PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL SARAIVA GARCIA

Inteligência de Processos de Negócio: Uma Proposta de Padronização entre as Etapas de Mineração de Dados e Visualização dos Resultados

Porto Alegre
2007

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Inteligência de Processos de Negócio: Uma Proposta de Padronização entre as Etapas de Mineração de Dados e Visualização dos Resultados

RAFAEL SARAIVA GARCIA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação, pelo Programa de Pós-Graduação da Faculdade de Informática.

Prof. Dr. Duncan Dubugras Alcoba Ruiz
Orientador

Porto Alegre
2007

Dados Internacionais de Catalogação na Publicação (CIP)

G216i Garcia, Rafael Saraiva.
Inteligência de processos de negócio : uma proposta de padronização entre as etapas de mineração de dados e visualização dos resultados / Rafael Saraiva Garcia. – Porto Alegre, 2007.
135 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Duncan Dubugras Alcoba Ruiz

1. Informática. 2. Descoberta de Conhecimento em Banco de Dados. 3. Mineração de Dados. 4. Visualização de Dados. 4. XML. I. Título.

CDD 005.74

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**Inteligência de Processos de Negócio: Uma Proposta de Padronização entre as Etapas de Mineração de Dados e Visualização dos Resultados**", apresentada por Rafael Saraiva Garcia, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Ciência da Computação, aprovada em 30/01/08 pela Comissão Examinadora:

Prof. Dr. Duncan Dubugras Alcoba Ruiz – Orientador — PPGCC/PUCRS

Prof. Dr. Marcelo Blois Ribeiro – — PPGCC/PUCRS

Prof. Dr. Carlos Alberto Heuser – — UFRGS

Homologada em 22/04/08, conforme Ata No. 008 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@inf.pucrs.br
www.pucrs.br/facin/pos

“All we have to decide is what to do with the time that is given us.”
Gandalf

“There is a difference between knowing the path and walking the path.”
Morpheus

“Memories fade, but a Google search never forgets!”
Perry White

Agradecimentos

Aos meus pais e avós pelo apoio, carinho e amor demonstrados ao longo de toda a minha vida e, principalmente, durante minha jornada acadêmica. Agradeço a eles a oportunidade que me deram de acesso ao ensino superior.

À minha namorada, Roberta Larratéa, por todo amor, carinho, dedicação e companheirismo ao longo desta jornada, sempre do meu lado, me auxiliando em tudo que era necessário. Sem ela esta dissertação não teria a mesma qualidade.

Aos coordenadores e gerente da Confederação SICREDI (Fábio Uggeri, Alejandro Halon e Leandro Castro), pela compreensão e apoio durante o período acadêmico, permitindo que pudesse me ausentar das minhas responsabilidades para dedicação ao curso de Mestrado.

Ao Prof. Duncan, meu orientador, pela paciência (e como foi necessária), apoio, compreensão e puxões de orelha.

Resumo

Diversos processos de negócio das organizações podem ser automatizados com o auxílio de sistemas de *Workflow*. Alguns deles, estrategicamente importantes, necessitam de ferramentas que permitam análises gerenciais e auxiliem os gestores no processo de tomada de decisão. Neste contexto, a aplicação das técnicas de descoberta de conhecimento sobre os registros de execução das instâncias dos processos de negócio mostra-se uma prática promissora.

No entanto, o ambiente computacional utilizado pelas aplicações de KDD pode ser significativamente complexo, tendo suas etapas executadas de forma independente como, por exemplo, em um ambiente orientado a serviços. Esta arquitetura possui um problema relacionado à troca de informações entre as etapas do processo, visto que cada serviço pode ter sido escrito em linguagens diferentes e necessitar que os dados estejam dispostos em um determinado formato. Neste caso, uma vez que este formato seja único, distintas aplicações podem trabalhar utilizando mesmos dados, agregando ao procedimento com diversidade de opções.

Seguindo esta problemática, este trabalho versa sobre uma abordagem que visa tornar independentes duas etapas do processo de descoberta de conhecimento: a mineração de dados e a visualização dos resultados. Para isto, a solução proposta está baseada no uso das tecnologias de XML e *XML Schema* para a definição de estruturas para as saídas e entradas dos algoritmos de mineração e técnicas de visualização. Além disto, o uso de técnicas de XSLT contribui para que a transformação entre estes formatos possa ser realizada de modo automatizado. Para a validação da solução, criada com base teórica, foram realizadas alguns testes utilizando as implementações de código livre. A principal contribuição deste trabalho está na criação de formatos únicos e genéricos para a troca de informações entre as etapas citadas, bem como sua transformação.

Palavras-chave: descoberta de conhecimento, técnicas de mineração de dados, visualização de dados, XML, *XML Schema*.

Abstract

Several business processes may be automated using Workflow systems. Some of these processes are strategically important and require tools to allow managerial analyses and assist process managers in decision making. Thus, the application of knowledge discovery techniques regarding records of steps of business processes shows great promise.

Yet, the computational environment adopted by KDD may be significantly complex, with steps executed independently such as seen in a service-oriented environment. This architecture poses a problem linked to the exchange of information between process steps, since each service may have been written in different languages and requires data to be allocated as a predefined format. In this case, if the format is exclusive, different applications may operate using the same data, adding an array of procedural options. In this sense, this paper addresses an approach to make independent two steps of the knowledge discovery process: data mining and visualization of results.

With this aim, the solution proposed is built on the use of XML and XML Schema technologies for the definition of data output and input structures of algorithms for data mining and visualization techniques. Moreover, the use of XLST techniques contributes to the automation of the transformation of formats. The validation of the solution, which was developed over the pertinent theory, was car out with experiments that used free code implementation. The main contributio this paper lies in the generation of exclusive and generic formats for information exchange between the steps mentioned and their transformation.

Keywords: knowledge discovery, data mining techniques, data visualization, XML, *XML Schema*.

Lista de Figuras

Lista de Siglas

BPEL	Business Process Execution Language
CSV	Comma Separated Values
DMG	Data Mining Group
DTD	Document Type Definition
EJB	Enterprise JavaBeans
ETL	Extract, Transform and Load
HTML	Hyper Text Markup Language
J2EE	Java2 Platform Enterprise Edition
KDD	Knowledge Discovery in Databases
PMML	Predictive Model Markup Language
SGML	Standard Generalized Markup Language
URL	Uniform Resource Locator
WfMC	Workflow Management Coalition
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XSD	XML Schema Definition
XSL	Extensible Style Sheet Language
XSLT	Extensible Style Sheet Language Transformation

Sumário

1 Introdução

Diversos processos presentes no dia-a-dia das grandes empresas podem ser automatizados com o auxílio de ferramentas computacionais. Tais processos vão desde a solicitação, aprovação e liberação de férias, viagens e compras de materiais até a definição das tarefas que uma determinada pessoa deve desenvolver dentro de um projeto. Estes fluxos de atividades são denominados processos de negócio e podem ser definidos como “um conjunto de um ou mais procedimentos ou atividades relacionados, os quais coletivamente atingem um objetivo de negócio, dentro do contexto de uma estrutura organizacional que define papéis funcionais e suas relações” [WfC99].

A necessidade de automação e controle dos processos de negócio nas organizações deu origem a sistemas conhecidos como *Workflow Management Systems* (WfMS). Estes sistemas são responsáveis pelo armazenamento dos dados dos processos de negócio, tanto os relacionados à definição dos modelos (automação), quanto à execução de suas atividades (gerência).

A tecnologia de *workflow* pode ser definida como a automação de processos de negócios, no todo ou em partes, na qual documentos, informações ou tarefas são passadas de um participante para outro, de acordo com um conjunto de regras procedimentais [Aal02], [Gol04], [WfC99]. Com o auxílio da tecnologia da informação diversas soluções e ferramentas foram desenvolvidas para o gerenciamento dos processos de negócio. Algumas delas são comercializadas por fabricantes importantes no mercado de sistemas como a *Oracle* que possui um WfMS denominado *Oracle Workflow* [Cha02]. Outras ferramentas, no entanto, estão baseadas em *software* livre, como o *Shark* [Enh07].

O uso sistemático desses sistemas acaba gerando grandes quantidades de dados que representam o estado e o comportamento real dos processos das organizações. Desta forma, o foco das empresas está se desviando do mero acompanhamento e controle desses processos para a medição, análise e monitoração dos mesmos [Cas05][Gol04]. Isto é realizado com o emprego de técnicas de descoberta de conhecimento (KDD) para análises baseadas em resultados obtidos por sumarizações de dados históricos e mineração de dados dos

processos. Assim, é possível identificar anomalias de execução dos processos, perceber as constantes modificações (tanto nos modelos quanto nas formas de execução de suas instâncias), verificar possíveis problemas de modelagem e sumarizar o comportamento dos processos de negócio, de acordo com os objetivos de cada organização [Aal03], [Aal05], [Gol04].

1.1 Caracterização do Problema

O autor deste trabalho apresenta em [Gar05], [Gar05a] e [Gar05b] uma ferramenta que tem por objetivo disponibilizar um ambiente computacional completo para a execução do processo de descoberta de conhecimento sobre bases de dados de processos de negócio. Esta ferramenta foi, inicialmente, implementada utilizando o modelo analítico multidimensional proposto por Casati em [Cas02] e, posteriormente, alterada para utilizar o modelo analítico proposto por Garcia em [Gar06] e [Gar06a].

Além disto, ainda em [Gar05], foi proposta, de forma superficial, uma maneira de se relacionar algoritmos de mineração de dados com técnicas de visualização. A proposta consistia em um mapeamento explícito entre classes de algoritmos e tipos de visualizações, realizado por um usuário com perfil de administrador do sistema. O objetivo deste mapeamento era tornar as etapas de mineração de dados e visualização dos resultados do processo de descoberta de conhecimento independentes.

A necessidade de padronizar mecanismos de mineração para serem utilizados na forma de componentes por outras ferramentas de KDD é bastante atual na comunidade cientifica. Esforços nesta direção podem ser categorizados nos seguintes aspectos [Gro02]: modelos, atributos, interfaces, APIs, configuração, processo e acesso remoto e distribuído.

1.2 Objetivos

O presente trabalho propõe uma abordagem para o mapeamento entre resultados de algoritmos de mineração de dados e formas de visualização destas informações. A partir de estudos sobre as técnicas de mineração é possível

identificar um conjunto principal de dados que são essenciais para a representação do resultado da execução de um algoritmo. Da mesma forma, uma análise sobre métodos de visualização proporciona a identificação do formato necessário para os seus dados de entrada. Assim, os resultados de execuções de algoritmos de mineração podem ser analisados com o auxílio de diferentes técnicas de visualização, mesmo que historicamente incompatíveis.

Para a validação desta proposta, são utilizados algoritmos e métodos de visualização fornecidos pelo *software* livre *Weka* [Wit05]. Esta escolha deve-se ao fato da ferramenta ser uma das mais utilizadas atualmente em contextos acadêmicos.

As contribuições desta pesquisa são:

- Identificação de padrões nos resultados de diferentes classes de algoritmos de mineração de dados;
- Identificação de padrões necessários para que seja possível a geração de visualizações de informações;
- Criação de um processo de transformação automático entre os formatos gerados.

1.3 Estrutura do Trabalho

Este documento encontra-se organizado em oito capítulos. O capítulo 2 discorre sobre alguns conceitos, definições e informações relacionadas à mineração de dados, além de apresentar um estudo sobre duas classes de algoritmos. O capítulo 3 trata sobre as técnicas de visualização de dados, apresentando uma análise sobre alguns métodos. Um estudo sobre a linguagem de marcação XML é apresentado no capítulo 4 e será utilizado como base para a criação dos formatos únicos para a troca de informações. O capítulo 5 trata de trabalhos relacionados à proposta ou utilizados como base. O capítulo 6, por sua vez, discorre sobre o problema que este trabalho visa resolver, fazendo sua contextualização e análise. O capítulo 7 apresenta os estudos de caso, os arquivos de padronização resultantes da pesquisa e descreve sobre a forma como foram desenvolvidos. Por fim, o capítulo

8 apresenta uma visão geral de todo o trabalho, expondo suas conclusões e sugestões para trabalhos futuros.

2 Mineração de Dados

Este capítulo apresenta uma visão geral sobre o processo de descoberta de conhecimento, e, após, relata um estudo sobre as duas das principais técnicas de mineração de dados: Associação e Classificação.

A descoberta de conhecimento em base de dados (também identificada como KDD) é um processo não trivial de identificação de padrões que sejam válidos, novos, potencialmente úteis e compreensíveis [Fay96]. Este processo é composto por várias etapas de manipulação de dados, definidas por Han [Han01] como: (a) limpeza, (b) integração, (c) seleção, (d) transformação, (e) mineração de dados, (f) validação dos resultados e (g) representação do conhecimento. A Figura 1 ilustra estas etapas e a hierarquia entre elas. Nela pode ser visualizado o ponto de partida para o processo de KDD, ou seja, uma base de dados contendo informações de execução de diversas aplicações salvas em formatos distintos (*databases* e *flat files*). Após a execução das etapas *(a)* e *(b)*, os dados consolidados são carregados para uma base multidimensional (*data warehouse*) onde ainda passam pelas etapas *(c)* e *(d)*. A execução da etapa *(e)* gera como resultado uma série de padrões que descrevem o comportamento dos dados. Estes padrões são validados através da etapa *(f)* e, após isto, na etapa *(g)*, podem ser dispostos aos usuários finais do sistema de KDD.

A mineração de dados é uma das principais etapas do processo de KDD e está relacionada ao uso de técnicas para extrair ou "minerar" conhecimentos a partir de uma base de dados analítica (ou multidimensional). Nela, métodos inteligentes são aplicados visando à busca de padrões. Dentre os diversos tipos de técnicas de mineração, as duas classes de algoritmos que, historicamente, recebem maior destaque são *Associação* e *Classificação.* Nas próximas seções é apresentado um estudo sobre estas duas técnicas que tem por objetivo identificar as suas principais características, focando na identificação da estrutura de como os resultados são gerados por estes algoritmos.

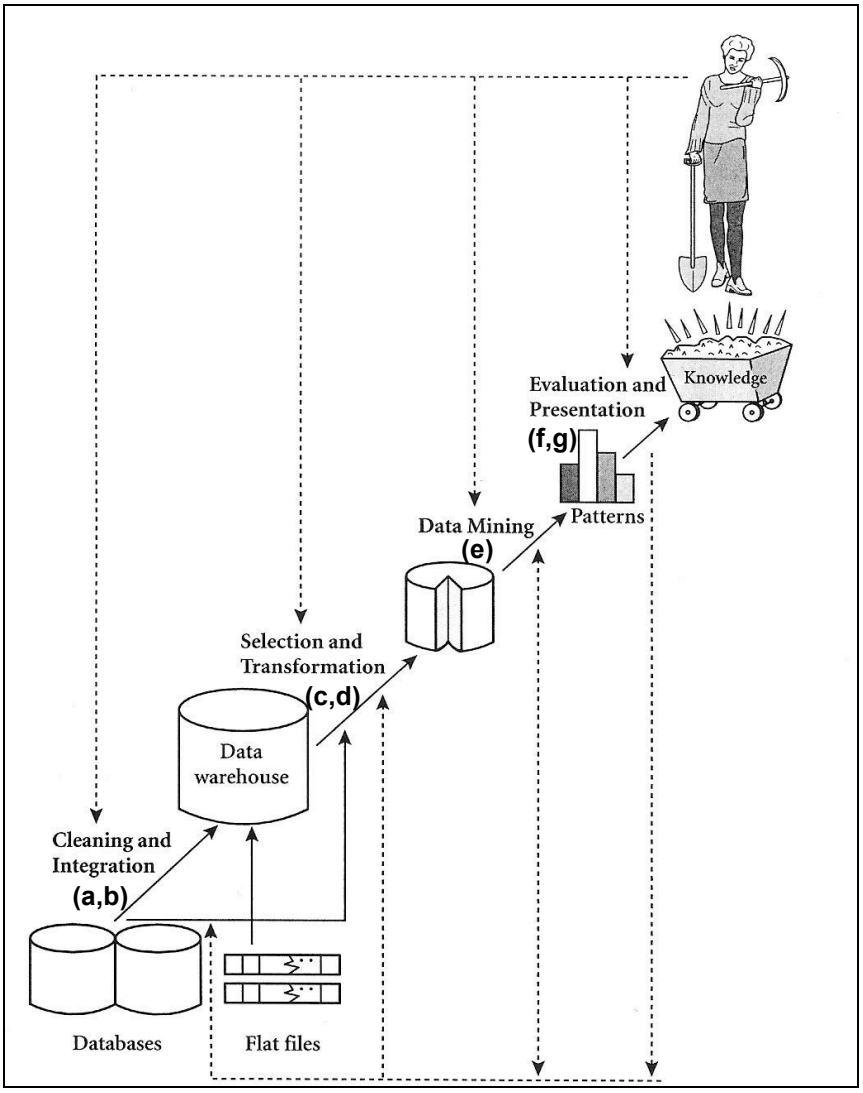


Figura 1 - Etapas do processo de descoberta do conhecimento [Han01]

2.1 Associação

A técnica de mineração de dados denominada *Associação* possui como objetivo a representação de padrões interessantes entre itens do domínio de uma aplicação, desde que eles possam ser verificados com freqüência na base de dados [Han01].

Um exemplo típico de uso desta classe de algoritmo é a análise de cestas de compras de supermercados, onde o processo analisa os hábitos de compra dos clientes buscando identificar associações entre os diferentes produtos comprados, conforme ilustrado pela Figura 2. Nela pode ser visualizado o trabalho de um analista de *marketing* tentando identificar as possíveis relações entre as compras

dos clientes, com base no histórico de quatro cestas de compras. Os resultados deste processo podem auxiliar estes profissionais a desenvolver novas estratégias de venda, contando com o conhecimento de quais produtos são freqüentemente comprados juntos. Dentre estas estratégias, pode-se citar a alteração da distribuição dos produtos no supermercado. Desta forma, produtos que, geralmente, são comprados juntos podem estar em prateleiras mais próximas, aumentando a venda de ambos. Outra opção é manter estes produtos localizados em prateleiras separadas com certa distância, forçando com que os clientes se desloquem mais pelo supermercado a sua procura. Conseqüentemente, isto poderá aumentar as vendas de outros produtos que estejam próximos dos mais procurados.

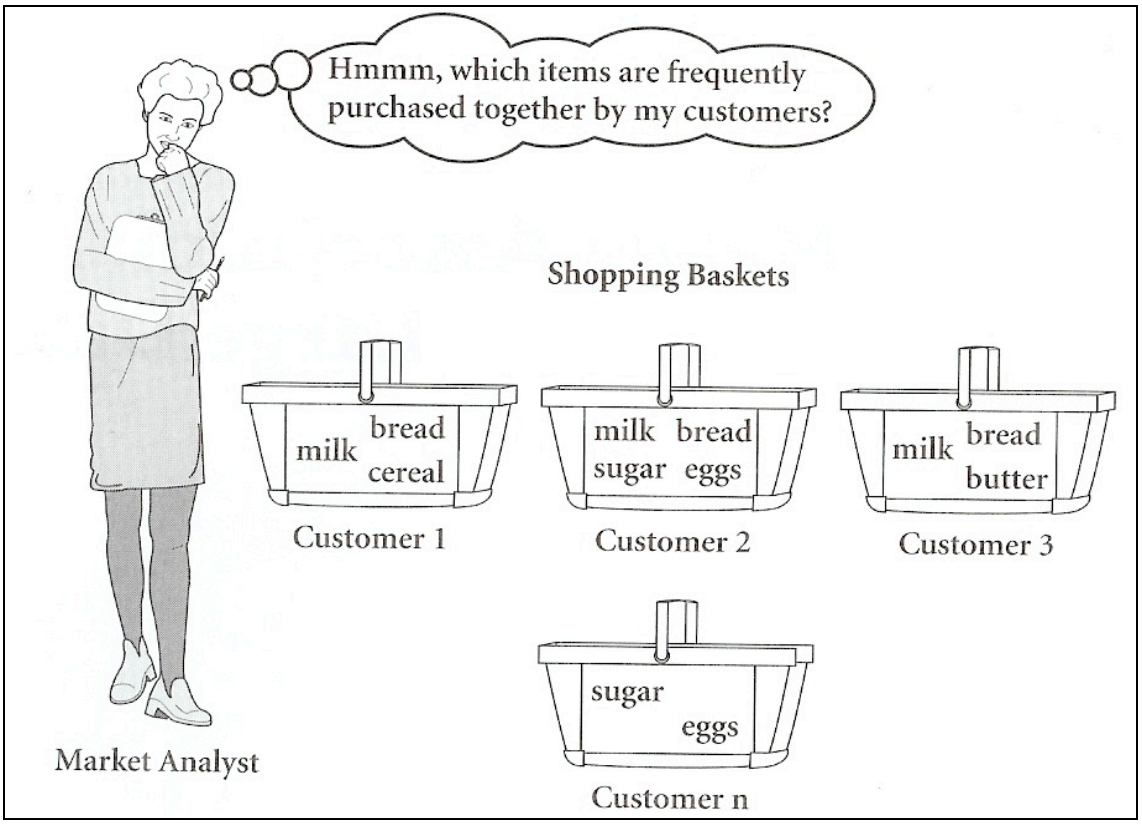


Figura 2 - Exemplo de um problema de associação [Han01]

Segundo Han [Han01], uma regra de *Associação* é representada através da implicação de um ou mais atributos em outros. Sendo assim, dada uma base de dados *D*, um conjunto de transações *T* e dois atributos quaisquer *A* e *B*, uma regra de *Associação* de *A* implicando em *B* é representada na forma $A \Rightarrow B$, onde $A \subset T$,

$B \subset T$ e $A \cap B = \phi$. Neste exemplo, o atributo *A* é definido como sendo a cabeça da regra e o atributo *B*, o corpo.

Para cada regra de *Associação* existe um fator de suporte (*support* ou *coverage*) que representa o percentual de todas as transações sob análise na base de dados, a qual contém todos os atributos especificados [Tan06]. Seguindo o exemplo supracitado, o fator de *suporte* é o percentual de transações *T* em *D* que contém $A \cup B$ (os dois atributos juntos). Este valor é dado pela probabilidade $P(A \cup B)$, ou seja, representa a razão entre transações onde a regra é verificada sobre todas as que estão sob análise na base de dados [Han01].

Além disto, toda regra de *Associação* também possui um fator de confiança (*confidence* ou *accuracy*). Este fator representa o percentual de transações sob análise na base de dados. Se estiver presente o atributo que compõe a cabeça da regra, o atributo que constitui o corpo também estará [Tan06]. Pensando novamente no exemplo anterior, o fator de *confiança* é o percentual de transações *T* em *D* que, se o atributo *A* estiver presente, também estará o atributo *B*. Este valor é dado pela probabilidade $P(B|A)$, ou seja, é a razão entre transações onde a regra é verificada sobre as que a cabeça está presente [Han01]. Uma regra que satisfaça um valor mínimo de suporte e de confiança é denominada *regra forte*. Estes valores são definidos, por convenção, como sendo um número entre o intervalo de 0% e 100% [Wit05].

O conjunto de atributos utilizado para a criação das regras recebe o nome de *itemset* e pode ser composto por *n* elementos. A sua freqüência é o número de transações sobre análise na base de dados no qual pode ser identificado. Os conjuntos compostos por apenas um atributo são denominados *one-item sets*, os compostos por dois são denominados *two-item sets*, e, assim, sucessivamente. Cada *itemset* satisfaz um suporte mínimo se a freqüência com que ocorre é maior ou igual ao produto do suporte mínimo com o número total de transações na base de dados, sendo, assim, chamado de *frequent itemset* [Wit05].

Os algoritmos de mineração da classe *Associação* executam, basicamente, duas etapas: encontrar os *frequent itemsets* e gerar *regras fortes* a partir deles. O conjunto de dados de entrada necessário é a base que contém todos os registros

desejados e os valores mínimos de suporte e confiança que deverão estar associados às regras resultantes do processo.

Após a sua execução, esta técnica produz como resultado *n* estruturas que correspondem às regras geradas, onde cada uma é composta por um conjunto de objetos que representam a cabeça da regra e outro que representa o corpo, conforme ilustrado na Figura 3. Além disto, conforme pode ser visualizado na figura, para cada regra, estão associados os seus respectivos valores de suporte e de confiança. Tanto a cabeça quanto o corpo de uma regra de *Associação* são um conjunto de um ou mais atributos, onde cada um é formado pelo seu nome, conforme a base de dados original, e pelo valor assumido por ele na regra correspondente [Han01].

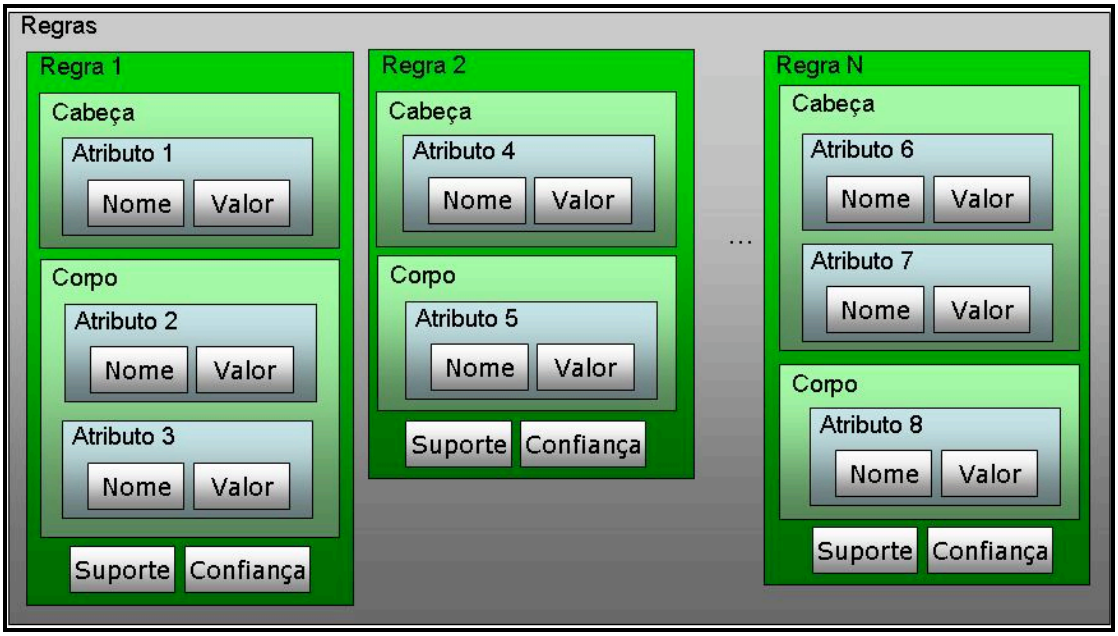


Figura 3 - Representação do resultado de algoritmos de *Associação*

2.1.1 Apriori

Dentre os algoritmos existentes para a classe *Associação*, um dos mais simples e conhecidos é o *Apriori*. Ele foi proposto por Agrawal [Agr94] com o objetivo de minerar regras associativas em bases de dados grandes e complexas. Esta técnica tornou-se bastante influente na mineração de dados, originando novos algoritmos, como o *Predictive Apriori*.

O nome *Apriori* está baseado no seu principal funcionamento, qual seja, utilizar conhecimentos identificados em execuções anteriores (*prior*) sobre as propriedades de conjuntos de atributos. Seu grande diferencial está na sua simplicidade original e versatilidade em bases de dados volumosas.

O *Apriori* aplica uma abordagem interativa conhecida como busca de nível inteligente (*level-wise*), onde *k* conjuntos de itens são utilizados para explorar *k+1* conjuntos. Na sua primeira execução, o algoritmo identifica os conjuntos de itens freqüentes compostos por um único atributo (*1-itemset*), denominados L_1. Estes conjuntos são utilizados para encontrar novos conjuntos, denominados L_2, ou seja, contendo todos os itens freqüentes compostos por dois atributos (*2-itemsets*). Este, por sua vez, é utilizado para encontrar L_3, que representa os conjuntos de três atributos (*3-itemsets*) e, desta forma, o algoritmo segue sucessivamente até que não seja possível a identificação de nenhum novo conjunto de *n* atributos. O *Apriori* possui como ponto negativo o fato de que a busca por cada um dos conjuntos exige uma leitura completa de todos os dados presentes no banco de dados [Han01].

O pseudocódigo deste algoritmo é ilustrado pela Figura 4. Nela, pode ser visualizada a especificação da função principal que é responsável por procurar por todos os *itemsets* que podem ser gerados a partir do conjunto de dados disponível, identificado por *D*. Para cada *itemset* gerado, denotado por L_{k-1}, é realizada uma análise para remover conjuntos que não atendem aos critérios estabelecidos, sendo realizada a chamada para a sub-rotina *apriori_gen*.

2.1.2 Predictive Apriori

O algoritmo denominado *Preditive Apriori* deriva do *Apriori*, o qual foi discutido na seção anterior. Ele foi criado por Scheffer [Sch01] e sua contribuição está fundamentada na importância que os valores de suporte e confiança possuem na geração de regas associativas.

Para que o *Apriori* possa ser executado, é necessária a definição de parâmetros que determinem os limites de suporte e confiança. Estes valores são utilizados para se tentar garantir a qualidade das regras geradas. Porém, ao selecionar somente as que superam este limite, nem sempre é possível obter como

resultado o conjunto com as melhores regras. Assim, a proposta do *Predictive Apriori,* consiste em buscar uma relação entre os valores de suporte e confiança que possam maximizar a chance de uma correta predição de dados não analisados (dados futuros ou que não foram utilizados no processo de mineração). Para isto, este algoritmo utiliza uma distribuição binomial onde a ocorrência do atributo analisado é classificada como correta ou incorreta.

```
Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on
candidate generation.
Input: Database, D, of transactions; minimum support threshold, min_sup.
Output: L, frequent itemsets in D.
Method:
(01) L1 = find_frequent_1-itemsets(D);
(02) for (k = 2; Lk-1 ≠ Ø; k++) {
(03)   Ck = apriori_gen(Lk-1,min_sup);
(04)   for each transaction t in D { // Scan D for counts
(05)     Ct = subset(Ck,t); // get the subsets of t that are candidates
(06)     for each candidate c in Ct
(07)       c.count++;
(08)   }
(09)   Lk = {c in Ck|c.count ≥ min_sup}
(10) }
(11) return L = UkLk;
procedure apriori_gen(Lk-1:frequent(k-1)-itemsets; min_sup: minimum support threshold)
(01) for each itemset l1 in Lk-1
(02)   for each itemset l2 in Lk-1
(03)     if (l1[1]=l2[1])^(l1[2]=l2[2])^…^(l1[k-2]=l2[k-2])^(l1[k-1]=l2[k-1]) then {
(04)       c=l1 join l2; //join step: generate candidates
(05)       if has_infrequent_subset(c,Lk-1) then
(06)         delete c; // prune step: remove unfruitful candidate
(07)       else add c to Ck;
(08)     }
(09) return Ck;
procedure has_infrequent_subset(c:candidate k-itemset; Lk-1: frequent (k-1)-itemsets);
(01) for each (k-1)-subset s of c
(02)   if s not in Lk-1 then
(03)     return TRUE;
(04) return FALSE;
```

Figura 4 - Algoritmo Apriori [Han01]

2.1.3 Tertius

Outro algoritmo existente para a classe *Associação* é o *Tertius*. Ele é descrito através de um sistema de descoberta do tipo *top-down* que faz uso de métricas de confirmação. Sua principal implementação, feita na linguagem de programação GNU C, livre para uso acadêmico, possui aproximadamente 7.500 linhas de código. Já a sua versão Java, disponibilizada juntamente com o *software* de mineração de dados Weka, é composta por doze classes e mais de 2.300 linhas de código [Fla01][Fla07].

O *Tertius* utiliza-se de lógica de primeira ordem para a geração da sua representação, o que o permite trabalhar com vários tipos de dados. Além disto, possibilita que os usuários possam escolher a forma de representação que lhe seja

mais conveniente, ou compreensível, dentre as disponíveis [Fla01]. O seu funcionamento é parecido com o *Apriori*, porém as regras são formadas utilizando a forma normal disjuntiva (*or*) ao invés da forma normal conjuntiva (*and*). O *Tertius* pode ser configurado para encontrar regras de predição de uma única condição ou de um atributo pré-determinado. Durante a personalização deste algoritmo, um parâmetro determina quando a negação é permitida no antecedente, conseqüente ou nos dois. Outros parâmetros determinam o número de regras a serem recuperadas, entre outros [Wit05].

Este algoritmo gera, como resultado, o padrão da sua categoria, ou seja, um conjunto de *n* regras de associação, cada uma composta por grupos de atributos que representam a cabeça e o corpo das regras. Os valores de confiança e suporte, no entando, estão disponíveis com os nomes de confirmação e freqüência observada.

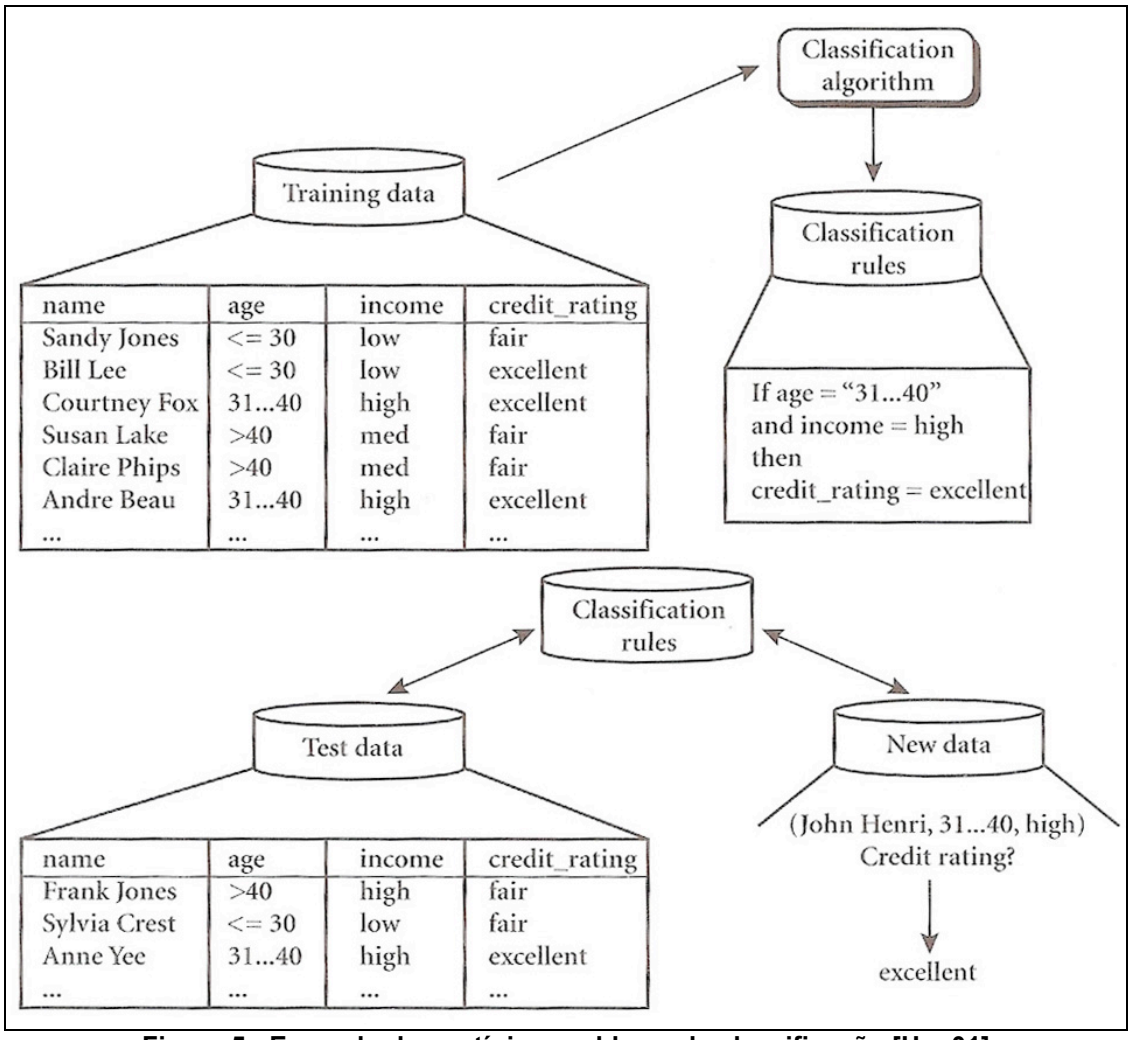
2.2 Classificação

A classe de mineração de dados denominada *Classificação* constitui uma forma de análise de informações que pode ser utilizada com dois objetivos, segundo Han [Han01] e Tan [Tan06]:

- Modelo descritivo: extrair modelos que descrevam as categorias dos dados. Estes modelos podem ser utilizados para fornecer um melhor entendimento sobre a organização da base de dados.
- Modelo Preditivo: realizar previsões de tendências futuras para registros de dados cujas classes ainda são desconhecidas.

Um típico exemplo de uso deste tipo de algoritmo é a criação de modelos de categorização para aplicações de empréstimo bancário que podem definir clientes com crédito “excelente” ou não, conforme ilustrado pela Figura 5. Nela, pode ser visualizada a existência de uma base de treinamento, contendo registros de clientes cuja classificação referente a empréstimos já é conhecida. A partir deste modelo, as regras de classificação são geradas. Uma delas, por exemplo, diz que clientes cuja taxa de crédito é excelente possuem, entre outros atributos, idade entre 31 e 40 anos. Na figura também pode ser observada a existência de uma base de testes,

onde as regras geradas são validadas. De acordo com estas regras, é possível definir qual a crédito “excelente” para um novo cliente chamado John Henri que possui, entre outros atributos, idade entre 31 e 40 anos. Assim, o uso deste modelo permite a categorização de futuros clientes, além de fornecer um melhor entendimento do conteúdo atual do banco de dados.



name	age	income	credit_rating
Sandy Jones	<= 30	low	fair
Bill Lee	<= 30	low	excellent
Courtney Fox	31...40	high	excellent
Susan Lake	>40	med	fair
Claire Phips	>40	med	fair
Andre Beau	31...40	high	excellent
...	...	...	...

name	age	income	credit_rating
Frank Jones	>40	high	fair
Sylvia Crest	<= 30	low	fair
Anne Yee	31...40	high	excellent
...	...	...	...

Figura 5 - Exemplo de um típico problema de classificação [Han01]

O processo de *Classificação* é realizado em duas etapas. Na primeira, um modelo é construído descrevendo um conjunto pré-determinado de classes ou conceitos. Este processo é realizado através da análise das tuplas de dados compostas por *n* atributos. Cada tupla é definida como pertencente a uma categoria, indicada pelo valor de um de seus atributos, mais conhecido como o atributo identificador de classe (*class label attribute*). Ao conjunto de tuplas utilizado para a construção do modelo, selecionado aleatoriamente a partir da amostra, dá-se o

nome de conjunto de treinamento (*training data set*). Esta etapa também é chamada de aprendizado supervisionado, vez que a classificação de cada item da amostra de treino é conhecida.

Na segunda etapa do processo, o modelo criado ao longo da primeira é testado para a validação da sua eficiência. Além disto, sua acurácia de predição é estimada utilizando o método de *holdout*. Esta é uma técnica simples e utiliza um grupo de tuplas de teste (*test set*) selecionado de forma aleatória e totalmente independente do conjunto de treinamento, podendo, inclusive, conter os mesmos registros. A acurácia final é o percentual de tuplas que são corretamente classificadas. Se ela estiver acima de um limiar considerado aceitável significa que o modelo gerado pode ser utilizado para futuras classificações de registros de categoria desconhecida.

Os algoritmos de *Classificação* possuem como entrada um conjunto de registros, compostos por *n* atributos, além da indicação de qual deles representa a sua classe. O resultado da sua execução é, tipicamente, uma estrutura em forma similar a de uma árvore de decisão, composta pelos atributos cujos testes sobre os valores são relevantes para a determinação da categoria do registro, conforme diagrama ilustrativo representado pela Figura 6. Nesta figura, pode ser visualizado o atributo principal, cujo teste separa o maior número de registros, identificado como o atributo raiz da árvore. A partir dele, existe uma lista de resultados possíveis e, para cada resultado, o seu relacionamento com um novo nodo, composto por outro atributo. Este, por sua vez, também poderá ter resultados associados ou não. No final, o valor dos nodos que não possuem resultados, ou seja, os nodos folha, representam a categoria dos registros classificados pela árvore.

2.2.1 ID3

Dentre os algoritmos que compõem esta classe, merecem destaque os que geram resultados na forma de árvores de decisão. Destes, o mais conhecido é o *ID3 (Iterative Dichotomiser 3*), um algoritmo guloso de indução. O seu funcionamento é baseado na criação de árvores de decisão utilizando a técnica recursiva de dividir e conquistar (*divide-and-conquer*) [Han01].

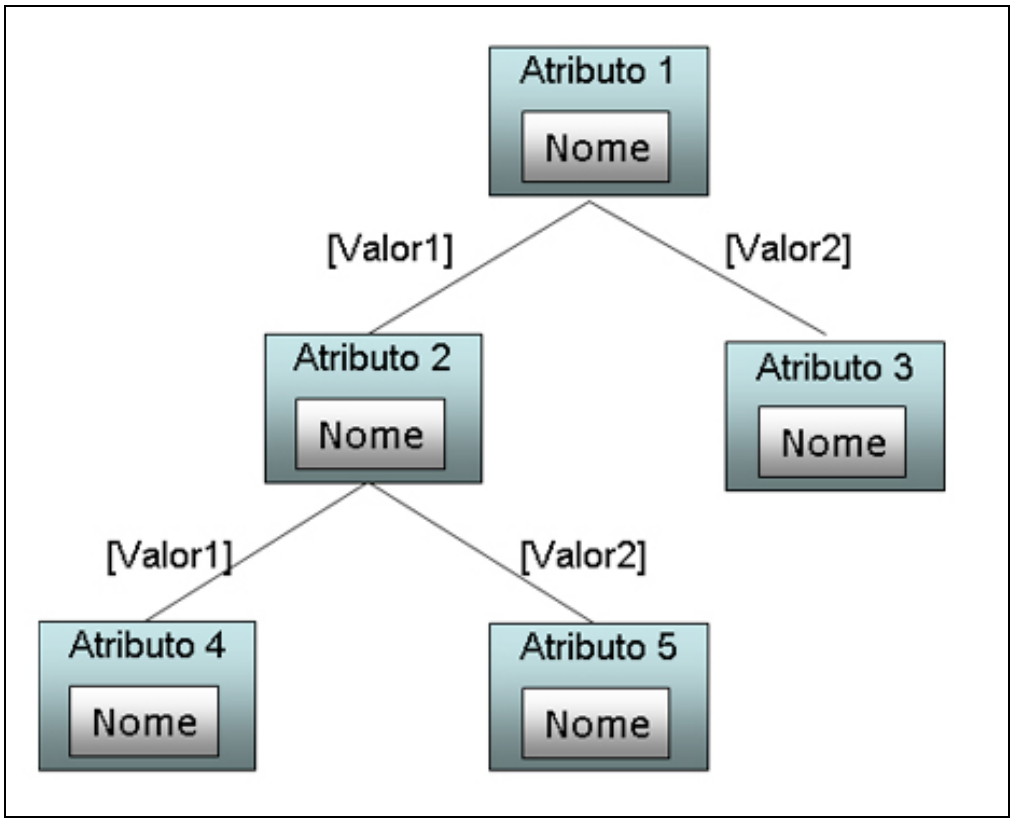


Figura 6 - Representação do resultado de algoritmos de *Classificação*

O *ID3*, cujo algoritmo sumarizado pode ser verificado na Figura 7, funciona da seguinte forma:

- Uma árvore é criada composta por um único nodo que representa as amostras de treinamento (passo 1).
- Se as amostras forem todas da mesma classe, então o nodo torna-se uma folha e é identificado com o valor desta classe (passos 2 e 3).
- Caso contrário, o algoritmo utiliza uma heurística conhecida como ganho de informação (*information gain*) para a seleção de um atributo que separa melhor as amostras em classes individuais (passo 6). Ele torna-se, então, o atributo de teste (ou decisão) para o nodo (passo 7). O algoritmo necessita que todos os seus valores sejam discretos. Desta forma, atributos com valores contínuos devem passar por um processo de discretização antes da sua execução.

- Um arco é criado para cada um dos valores conhecidos como resultado para atributo de teste. As amostras são separadas de acordo com estes valores (passos 8 a 10).

- O algoritmo utiliza este mesmo processo recursivamente para formar a árvore de decisão das amostras de cada partição. Uma vez que um atributo é selecionado para teste de um determinado nodo, não poderá mais ser utilizado nos nodos subseqüentes (passo 13).

- As repartições recursivas terminam apenas quando uma das seguintes condições for verdadeira:

- Todas as amostras para um determinado nodo pertencerem a uma mesma classe (passos 2 e 3); ou

- Não existirem mais atributos disponíveis para particionar as amostras (passo 4). Neste caso, a técnica de votação majoritária (*majority voting*) é empregada (passo 5). Isto envolve converter um determinado nodo em folha e identificá-lo com a classe principal das amostras; ou

- Não existirem mais amostras para o arco do atributo de teste (passo 11). Neste caso, uma folha é criada com a classe principal das amostras (passo 12).

```
Algorithm: Generate_decision_tree. Generate a decision tree from the giving training data.
Input: The training samples, samples, represented by discrete-valued attributes; the set of
candidates attributes, attribute-list.
Output: A decision tree.
Method:
(01) create a node N;
(02) if samples are all of the same class, C then
(03)   return N as a leaf node labeled with the class C;
(04) if attribute-list is empty then
(05)   return N as a leaf node labeled with the most common class in class; // majority voting
(06) select test-attribute, the attribute among attribute-list with the highest information
gain;
(07) label node N with test-attribute;
(08) for each know value a_i of test-attribute // partition the samples
(09)   grow a branch form node N for the condition test-attribute = a_i;
(10)   let s_i be the set of samples in samples for which test-attribute = a_i; // a partition
(11)   if s_i is empty then
(12)     attach a leaf labeled with the most common class in samples;
(13)    else attach the node returned by Generate_decision_tree(s_i, attribute-list-test-
attribute);
```

Figura 7 - Algoritmo ID3 [Han01]

2.2.2 C4.5

Outro algoritmo historicamente importante da categoria *Classificação* é o C4.5, também utilizado para a geração de árvores de decisão e, eventualmente, denominado como sendo um classificador estatístico.

Este algoritmo, provavelmente um dos mais utilizados atualmente [Wit05], é uma extensão do *ID3*, sendo que ambos foram propostos por Quinlan [Qui93]. O C4.5 apresenta uma série de melhorias em relação ao seu antecessor, merecendo destaque as seguintes:

- Uso de atributos contínuos e discretos: para que seja possível a manipulação de atributos contínuos, o C4.5 cria um limiar e, então, divide os valores dos atributos entre os que estão acima deste limiar e aqueles que estão abaixo ou são iguais e ele;
- Tratamento de dados de treino com valores perdidos: o C4.5 permite que valores de atributos que não estejam disponíveis sejam marcados como “?”. Os atributos que não tiverem valores simplesmente não são utilizados.
- Otimização das árvores após a criação: assim que uma árvore é criada o C4.5 faz uma revisão na tentativa de remover caminhos que não auxiliam no entendimento dos dados. Eles são substituídos por nodos folhas.

2.3 Considerações

O presente capítulo apresentou um estudo relacionado ao processo de descoberta de conhecimento, listando as suas principais etapas conforme definidas por Han [Han01]. Além disto, foi apresentada uma breve análise sobre duas das classes de algoritmos mais significativas: *Associação* e *Classificação.*

A classe de mineração de dados denominada *Associação*, apresentada na seção 2.1, almeja a busca de relacionamentos entre os diversos atributos de registros de uma base de dados. O resultado da sua execução é uma lista de regras de associação compostas de elementos denominados *cabeça* e *corpo*. Cada um deles é formado por um conjunto de um ou mais atributos, bem como pelos valores

assumidos para cada regra. Além disto, as regras de associação possuem, ainda, os valores de suporte e confiança.

A seção 2.1.1 discorreu sobre o algoritmo *Apriori*, que é um dos principais e mais simples desta classe. Ele influenciou na criação de outro algoritmo, um pouco mais eficiente, denominado *Predictive Apriori*, apresentação na seção 2.1.2. Além destes, existe, ainda, o *Tertius*, que foi analisado na seção 2.1.3 e utiliza lógica de primeira ordem para a geração das regras.

A seção 2.2 apresentou outra importante classe de algoritmos de mineração, a *Classificação.* Os algoritmos deste tipo possuem dois objetivos: extrair modelos para descrever as categorias dos dados e realizar previsões de tendências futuras utilizando os modelos criados. Dos diversos algoritmos existentes para esta classe, foram discutidos dois: o ID3 e o C4.5. O primeiro, apresentado na seção 2.2.1, é um algoritmo guloso de indução, cujo resultado é o mesmo que o da maioria dos demais algoritmos desta classe: é expresso através de árvores de decisão. Já o C4.5, discutido na seção 2.2.2, é, provavelmente, o mais utilizado atualmente, sendo uma evolução do algoritmo ID3. Ele apresenta uma lista de melhorias como, por exemplo, a possibilidade de se trabalhar com valores contínuos.

Este capítulo foi importante para o contexto deste trabalho por fornecer o referencial teórico sobre as principais classes de mineração de dados, buscando identificar o formato como os dados de saída das execuções dos algoritmos de mineração estão organizados. Com base nesta análise, foi possível a identificação de padrões nos resultados dos diferentes algoritmos de cada uma das classes estudadas, que serão utilizados na criação dos arquivos XML de definição de estruturas, apresentados no capítulo 0 deste documento.

Neste sentido, pôde-se observar que algoritmos da classe *Associação* geram como resultado *n* regras, onde cada uma possui os seguintes elementos: cabeça (composta por um ou mais atributos), corpo (também formado por uma lista de um ou mais atributos), fator de confiança e de suporte. Já as regras do tipo *Classificação* criam estruturas de decisão para categorização de instâncias. Dentre as opções de saída desta categoria, optou-se por trabalhar com a que produz árvores de decisão. Para esta opção, o resultado é uma estrutura que inicia a partir de um atributo onde

o teste sobre os seus possíveis valores assumidos indica novos testes ou o atributo responsável pela classificação da instância.

3 Técnicas de Visualização

Este capítulo apresenta uma visão geral sobre técnicas de visualização de informação, e, após, relata um estudo sobre algumas delas, como Árvores de Decisão e Redes Bayesianas.

A visualização de dados consiste em disponibilizar informações em uma forma textual, gráfica ou tabular, sendo que o seu principal objetivo é possibilitar a interpretação visual de informações por pessoas. Bons métodos de visualização necessitam que os dados sejam convertidos em um determinado formato, sendo que suas características e relacionamentos com itens ou atributos devem possibilitar boas análises e relatórios [Tan06].

A primeira etapa do processo de visualização é o mapeamento da informação para um formato visual. Assim, dados de objetos, seus atributos e seus relacionamentos são transformados em elementos gráficos como pontos, linhas, formas e cores. Os objetos são, geralmente, representados em uma das seguintes três maneiras [Tan06]:

- Se apenas um atributo objeto está sendo considerado para categorização, então são gerados aglomerados de dados organizados de acordo com os seus possíveis valores. Estas categorias são visualizadas como uma entrada de uma tabela ou como uma área na tela;
- Quando um objeto possui múltiplos atributos, então pode ser visualizado como uma linha (ou coluna) de uma tabela ou como uma linha de um gráfico;
- Um objeto que é representado como um ponto em um espaço bidimensional ou tridimensional pode ser representado como uma figura geométrica, como círculos e caixas, entre outros.

Existem diversas formas para a representação de padrões gerados por métodos de descoberta de conhecimento. Nas próximas seções é apresentado um estudo sobre algumas destas técnicas que tem por objetivo identificar as suas principais características, focando na identificação da estrutura de como os dados devem estar dispostos para que possam ser analisados visualmente.

3.1 Árvore de Decisão

A técnica de visualização de informações denominada *Árvore de Decisão* é um resultado natural da abordagem do tipo dividir e conquistar (*divide-and-conquer*) para um conjunto de instâncias independentes. Uma árvore de decisão é uma estrutura semelhante a um gráfico de fluxo em estrutura de árvore [Han01], conforme ilustrado pela Figura 8. Nela, pode ser observado o nodo principal, a raiz, que representa o primeiro atributo que é levado em consideração para a classificação de instâncias. Este atributo representa uma taxa de produção e possui dois possíveis resultados: reduzido ou normal. Para o valor reduzido existe um novo nodo que não possui nenhum teste, denominado nodo folha. Para o valor normal da raiz, existe um novo nodo, com um novo atributo para testes e novos resultados.

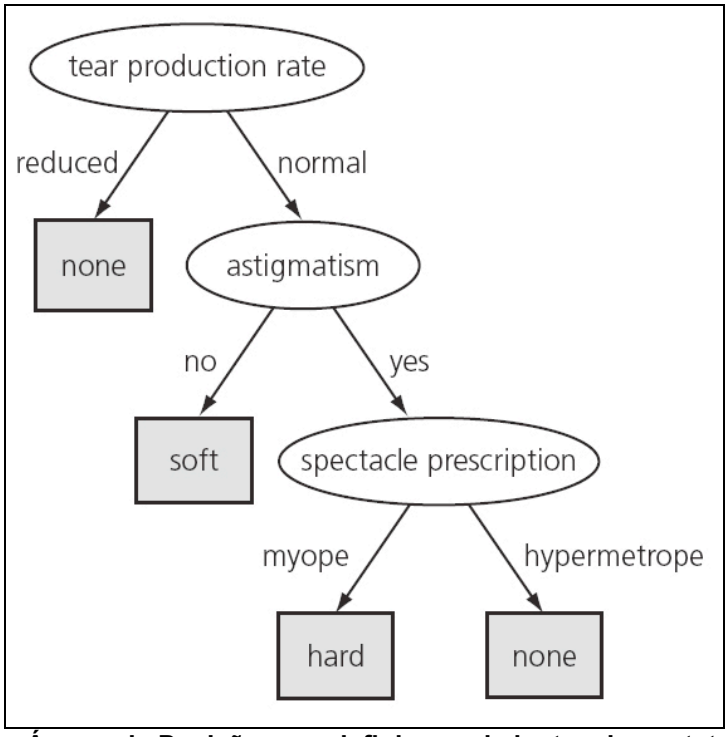


Figura 8 - Árvore de Decisão para definir uso de lentes de contato [Wit05]

Em uma árvore de decisão, os nodos internos representam testes para determinados atributos comparando-os com constantes pré-estabelecidas, com valores de outros atributos ou com o resultado da execução de funções. Quanto aos arcos da árvore, estes representam as possíveis saídas para a realização dos testes. Além disto, os nodos folhas representam a classificação que é aplicada a todas as instâncias que chegam até eles [Wit05].

Segundo Tan [Tan06], as principais características de árvores de decisão são:

- Interpretabilidade: os modelos gerados são de fácil compreensão;
- Robustez: a acurácia dos modelos não é afetada por eventuais ruídos ou dados redundantes.

Para que seja possível verificar a classificação uma determinada instância utilizando este técnica, é necessário percorrer a árvore, partindo da raiz e escolhendo os arcos correspondentes aos valores dos atributos representados pelos nodos até que se alcance os nodos folhas. Em outras palavras, partindo-se do nodo raiz, a árvore deve ser percorrida recursivamente, dividindo o conjunto de treinamento em subconjuntos de acordo com o critério de divisão (testes dos nodos) até que cada sub-árvore resulte em um nodo folha. Neste momento, a instância que está sob análise é classificada conforme o valor do nodo folha.

Se um determinado atributo, utilizado como teste em um nodo, for nominal, então o número de nodos filhos é, geralmente, o número de possíveis valores que o atributo pode conter. Nestes casos, como existe um arco para cada valor possível, o mesmo atributo não é utilizado novamente na estrutura árvore. Por outro lado, se o valor do nodo em teste for do tipo numérico, seu teste determina quando o valor é igual, maior ou menor que uma determinada constante, gerando três possíveis resultados. Outra possibilidade de uso de atributos numéricos inclui o teste do seu valor com um intervalo (gerando os arcos *abaixo*, *contidos* ou *acima*).

Um dos problemas óbvios, para a técnica de *Árvores de Decisão*, é a ocorrência de instâncias com atributos utilizados como testes em nodos sem valores, visto que não se sabe qual o arco deve ser selecionado durante a sua classificação. Uma abordagem para solucionar este problema é criar um arco no nodo que represente atributos sem valor. Porém, em alguns casos esta não é uma possibilidade e os atributos sem valores devem ser tratados como um caso especial. Uma alternativa é salvar o número de instâncias do conjunto de transações que são direcionadas para cada arco do nodo e, quando uma instância não possuir valor para o atributo correspondente, selecionar o arco mais utilizado até o momento.

Este técnica de visualização necessita como entrada para a sua correta execução de uma estrutura em forma de árvore. Nesta estrutura, composta por nodos, cada um deve conter o nome do atributo de teste que representa e, para cada um deles, deve existir uma lista de possíveis resultados, os quais formam os arcos. Cada arco é a representação do relacionamento entre dois nodos, ou seja, o resultado do teste para o atributo de origem que indica a próxima validação a ser realizada, no seu componente de destino. Por fim, nodos que não possuírem arcos, ou seja, nodos folhas, representam o valor resultante da classificação expressa na árvore.

3.2 Redes Bayesianas

Os classificadores *bayesianos* utilizam uma abordagem probabilística para modelar o relacionamento entre os atributos preditivos e a classe alvo. Uma *Rede Bayesiana* é um gráfico que representa o relacionamento probabilístico de um conjunto de variáveis [Hec97]. Uma Rede Bayesiana é dividida em duas partes:

- Qualitativa: grafo acíclico dirigido, que representa os relacionamentos de dependência entre um conjunto de variáveis;
- Quantitativa: Tabelas de probabilidades condicional, associadas a cada nodo e seu relacionamento com outros.

A Figura 9 ilustra uma *Rede Bayesiana*. Nela, é possível verificar um exemplo de classificação de fraudes, baseado na probabilidade de ocorrência de atributos como idade e sexo. Cada nodo da rede representa um atributo do domínio e os arcos representam as relações de dependência entre eles. A probabilidade entre dois nodos na rede representa a força do relacionamento causal entre eles [Hec97]. Desta forma, o relacionamento entre as variáveis do domínio é explorado através de probabilidades condicionais.

A abordagem probabilística das redes bayesianas é baseada no teorema de Bayes, o qual consiste em um princípio estatístico para combinar conhecimento *a priori* de classe com novas evidências, obtidas a partir dos dados [Tan06]. Este teorema pode ser traduzido da seguinte pela fórmula P(B|A) = P(A|B) * P(B) / P(A). Nela, P(A|B) significa a probabilidade do atributo *A*, tal que tudo que se tem

conhecimento é o atributo *B*. Através do uso de probabilidades é possível conhecer a influência de um atributo sobre o outro.

Dada uma distribuição de probabilidades para cada atributo, uma Rede Bayesiana permite predizer um atributo desconhecido (atributo classe) através do cálculo da distribuição de probabilidades a posteriori [Hec97]. Entre as principais características das Redes Bayesianas estão [Tan06]:

- Representatividade: representação das dependências causais entre atributos.
- Interpretabilidade: os modelos gerados são de fácil interpretação, sendo bem adaptados para situações contendo atributos sem valor.
- Robustez: a acurácia dos modelos não é afetada por eventuais ruídos ou dados redundantes.

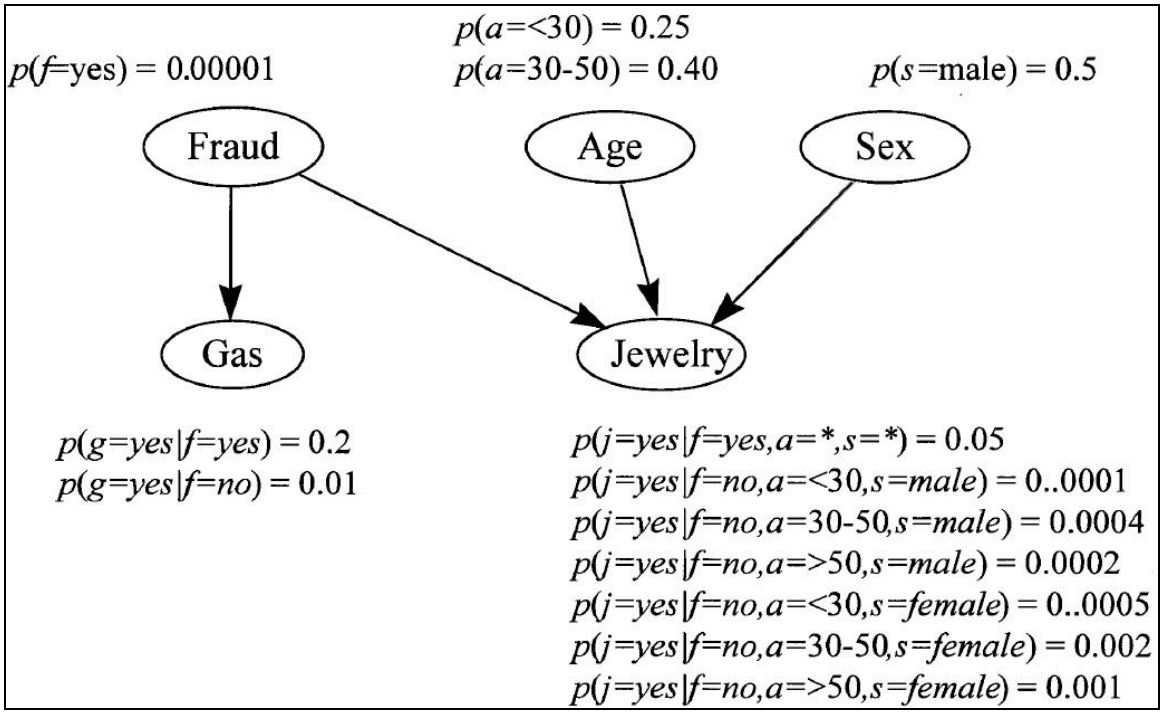


Figura 9 - Rede Bayesiana para detecção de fraudes [Hec97]

Sobre a técnica de representação do conhecimento de *Redes Bayesianas* pode-se concluir que é necessário um conjunto de nodos e suas dependências (quando houver), além da probabilidade associada ao nodo e o seu imediatamente relacionado.

3.3 Outras Visualizações

As técnicas de visualização são, tipicamente, especializadas para o tipo de dados analisado. Novos métodos de representação do conhecimento, bem como variações dos já existentes, são constantemente criados. Além das representações de dados apresentadas nas seções anteriores, duas outras técnicas também podem ser citadas: histogramas e gráficas de pizza.

3.3.1 Histogramas

Os histogramas são gráficos que exibem distribuições de valores para atributos. Eles funcionam através da divisão de possíveis valores em conjuntos e exibindo o número de objetos presentes em cada um. Para dados categóricos, cada valor do histograma representa um conjunto.

Após a criação dos grupos e da divisão dos objetos neles, um gráfico de barras é construído. Nele, cada conjunto é representado por uma barra e a área de cada barra é proporcional ao número de valores (objetos) que faz parte do intervalo correspondente. A Figura 10 exibe quatro histogramas para atributos da flor Íris, onde cada um é composto por dez conjuntos de valores diferentes.

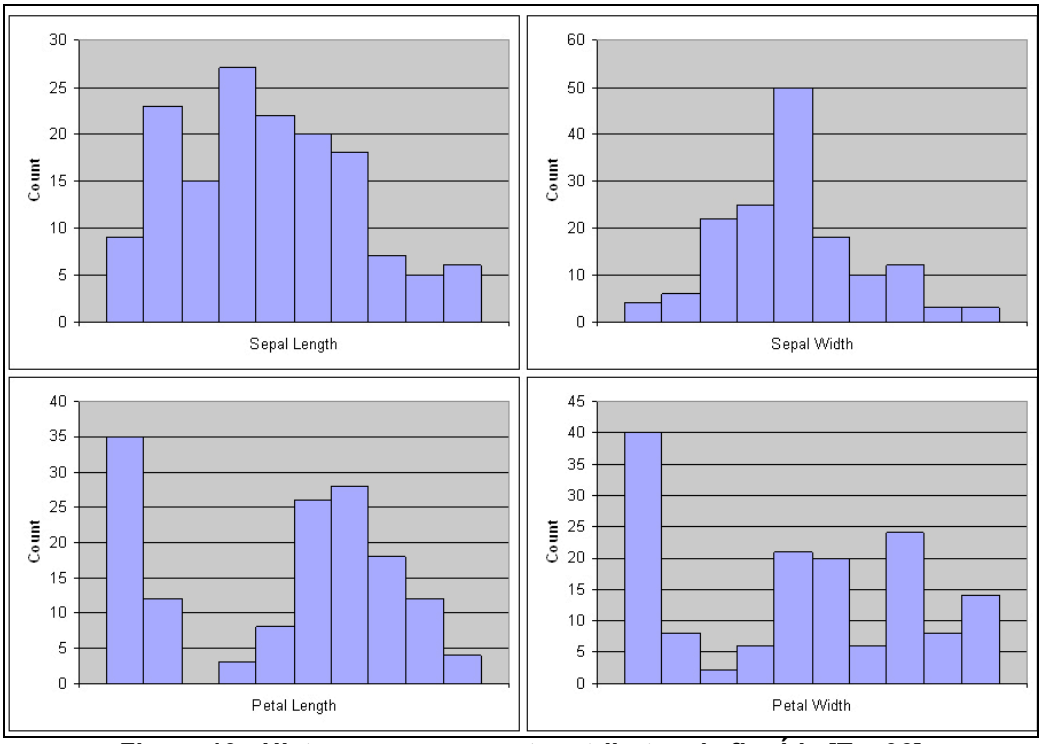


Figura 10 - Histograma para quatro atributos da flor Íris [Tan06]

3.3.2 Gráfico de Pizza

Um gráfico de pizza é uma visualização de dados semelhante a um histograma. Porém é tipicamente utilizado com atributos categóricos que possuem um número relativamente pequeno de valores. Ao invés de representar a freqüência relativa dos diferentes valores através de uma área ou altura de uma barra, esta visualização utiliza a área relativa de um circulo para indicar a freqüência. Mesmo que este tipo de gráfico seja popular no mundo dos negócios, eles são pouco utilizados em publicações técnicas porque o tamanho das áreas relativas pode ser difícil de julgar. Para esta finalidade, histogramas ainda são preferíveis.

A Figura 11 exibe um exemplo de um gráfico de pizza para as possíveis distribuições de cores da flor Íris para um determinado conjunto de dados.

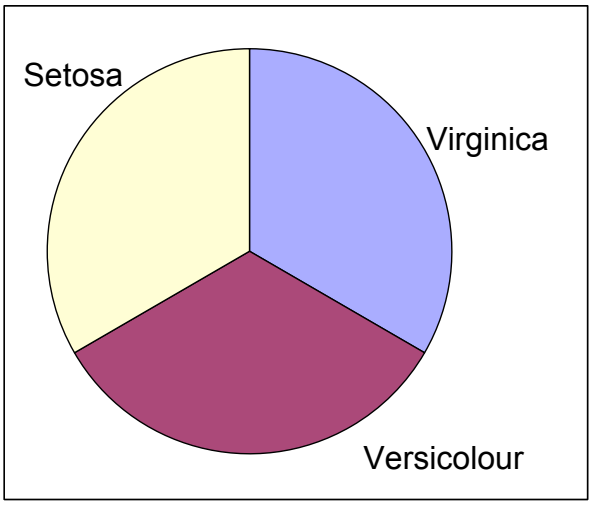


Figura 11 - Gráfico de pizza para cores da flor Íris [Tan06]

3.4 Considerações

Este capítulo discorreu sobre um estudo relacionado a algumas técnicas de visualização de dados. Este estudo apresentou alguns métodos importantes como *Árvores de Decisão* e *Redes Bayesianas*.

A técnica de visualização de informações denominada *Árvores de Decisão*, apresentada na seção 3.1, é o resultado natural da abordagem de dividir e conquistar. Para que a sua representação seja possível, os dados devem estar organizados em estruturas de nodos e arcos. A primeira estrutura representa o valor

de atributos do domínio de visualização, enquanto que a segunda representa o relacionamento entre eles.

A seção 3.2 discorreu sobre o método de geração de visualização *Redes Bayesianas*. Este técnica, baseada no teorema de Bayes, utiliza uma abordagem probabilística para modelar o relacionamento entre os atributos preditivos e a classe alvo. Para que a sua representação seja criada, é necessária a existência dos atributos, os relacionamentos entre eles, e a probabilidade condicional da sua ocorrência. Além destas técnicas, foram apresentadas na seção 3.3, as técnicas de histograma e gráficos de pizzas, que definem visualizações interessantes, dependendo do contexto dos dados que se deseja analisar.

Para o contexto deste trabalho, no entanto, foi selecionada para uso apenas a visualização de *Árvores de Decisão*, visto que é uma das mais comumente utilizadas. Neste sentido, pode-se observar que a técnica de representação de dados *Árvore de Decisão* necessita, como informações de entrada, estruturas que representam nodos. Cada uma deles deve conter o atributo do conjunto de dados correspondente e a sua lista de arcos relacionados, quando existirem. Os nodos que não possuírem relacionamentos são responsáveis por indicar as classificações do modelo. Estes devem ser apresentados de forma diferenciada, chamando a atenção dos usuários.

4 Linguagem de Marcação

Este capítulo apresenta um estudo sobre o XML, que é uma das principais linguagens de marcação, além de discorrer sobre os esquemas para definição de arquivos deste tipo e suas transformações.

Uma linguagem de marcação é um conjunto de códigos aplicados a um texto ou a dados. Sua principal finalidade é adicionar informações particulares a eles. As linguagens de marcação são usadas, por exemplo, na indústria editorial para marcar a formatação (exibição gráfica) de páginas. Se o código de marcação for padronizado, ou puder ser processado por um programa de computador, garante-se o intercâmbio de uma publicação complexa entre autores, editores e impressoras.

4.1 XML

A XML (*Extensible Markup Language*) é uma linguagem de marcação padrão criada pela W3C com o intuito de ser um formato universal para a troca de informações na Web [W3C07a]. Ela define um conjunto de regras para a definição de marcas semânticas que dividem um determinado documento em diversas partes [Har99].

Os documentos escritos utilizando a XML são compostos de uma mistura de dados e marcações (*tags*). É importante destacar que as marcas em XML descrevem estrutura e significado de documentos, mas não descrevem a formatação dos seus elementos. Para isto, podem ser utilizadas folhas de estilo. Com o uso de XML porque é possível fazer sistemas diferentes trabalharem juntos através da troca de informações, desde simples números até dados estruturados [Gol01].

4.2 XML Schema

Um dos principais aspectos que tornam a XML flexível decorre do fato de que as suas marcações são maleáveis, além de autodescritivas. Com isto, é possível atribuir nomes mais significativos ao conteúdo referenciado. Um documento escrito nesta linguagem pode conter, opcionalmente, uma descrição de sua gramática, que é utilizada por diversas aplicações para validar a sua estrutura. Esta descrição,

conhecida como esquema, possui, atualmente, duas propostas: *DTD* e *XML Schema* [W3C07].

A *DTD* (*Document Type Definition*) especifica um conjunto de regras para a estrutura dos documentos. Este formato define uma lista de elementos, atributos, notações e entidades que podem existir nos documentos XML, bem como o relacionamento entre eles. Os arquivos *DTD* podem estar incluídos no próprio documento XML ou vinculados através de uma URL externa [Har99]. A *DTD* possui uma notação própria para esta definição.

O formato *XML Schema*, por outro lado, é escrito como sendo um documento XML. Este padrão foi desenvolvido com o objetivo de ser um vocabulário compartilhado para a validação automática da estrutura de outros arquivos XML. Um documento deste formato, comumente denominado XSD (*XML Schema Definition*) é, tipicamente, um arquivo com a extensão "*xsd*", e pode ser utilizado para definir [W3S07]:

- Elementos que podem aparecer em um documento;
- Atributos que podem aparecer em um documento;
- Quais são os elementos filhos;
- A ordem dos elementos filhos;
- O número máximo de elementos filhos;
- Se um elemento é vazio ou poderá ter a inclusão de textos;
- Tipos de dados de elementos e atributos;
- Valores padrões e/ou fixos para elementos e atributos.

4.3 XSL Transformation

Além da definição de esquemas para os documentos XML, a W3C definiu alguns formatos para visualização dos dados contidos nos documentos. Estes padrões começaram a ser desenvolvidos porque existia uma demanda por linguagens do tipo folha de estilos para documentos XML. A XSL (*XML Style Sheets*) descreve como os arquivos XML devem ser exibidos [W3S07a]. Por exemplo, um

elemento do tipo <*table*> pode significar uma estrutura de tabela na linguagem HTML, uma peça de mobília ou qualquer outra coisa (e os navegadores não sabem como exibir isto).

O padrão XSL define tanto uma nova linguagem de marcação quanto uma linguagem de formatação. A transformação de um documento XML pode utilizar as marcações e esquemas originais do documento ou um conjunto de marcas completamente diferentes [Har99]. A XSL é composta por três partes [W3S07a]:

- XSLT: linguagem para transformação entre documentos XML;
- XPath: linguagem para navegação em documentos XML;
- XSL-FO: linguagem para formatação de documentos XML.

XSLT é considerada a parte mais importante da especificação do XSL. Ele permite que um arquivo XML possa ser transformado em um outro arquivo XML (com estrutura semelhante ou diferente), ou outros formatos, que possam ser reconhecidos pelos navegadores, como HTML e XHTML, sendo, este último, o seu uso mais freqüente. O XSLT permite a adição e/ou exclusão de elementos e atributos no arquivo de saída. Além disto, fornece meios para que os elementos possam ser organizados, ordenados, terem seus valores validados com testes para tomadas de decisões sobre quais elementos exibir e quais ocultar, entre outros [W3C07b].

No processo de transformação, a XSLT utiliza a tecnologia de XPath para definir partes do documento de origem que devem equiparar com um ou mais modelos pré-definidos. Quando uma igualdade é encontrada, o código de origem é transformado no código de destino, conforme as regras definidas no arquivo de transformação.

4.4 Considerações

O presente capítulo apresentou um estudo sobre a linguagem de marcação de dados denominada XML. Além disto, foi discutida a forma de definição da estrutura de documentos deste tipo e a linguagem de transformação dos dados.

A *XML Schema*, discutida na seção 4.2, é uma especificação utilizada para definir a estrutura que deve ser seguida por novos documentos criados com a linguagem XML, apresentada na seção 4.1. Ela, ao contrário da *DTD*, também é escrita utilizando-se o padrão XML, fato que facilita a sua utilização. Entre outros pontos positivos do seu uso, pode-se citar a flexibilidade para futuras melhorias na especificação, além da possibilidade de criação de esquemas mais ricos e poderosos, contando, inclusive, com a possibilidade de definição do tipo dos dados.

A seção 4.3 apresentou a *XSL*. Este formato, também definido pela W3C, é utilizado para definir como os documentos XML podem ser visualizados, servindo como uma espécie de folha de estilos. Com o seu uso, é possível transformar um documento XML em uma página HTML. A *XSL* é composta de três partes: *XSL Transformation*, *XPath* e *XSL-FO*.

Este capítulo foi importante para o contexto deste trabalho por fornecer o referencial teórico sobre a linguagem XML e os demais padrões que estão relacionados com esta tecnologia. Considerando que ela é uma das mais importantes linguagens de marcação em uso atualmente, tornou-se a opção mais lógica para a proposta deste trabalho, detalhada no capítulo 6.

5 Trabalhos Relacionados

Este capítulo apresenta um trabalho que está relacionado ao tema de pesquisa exposto por este documento, descrevendo suas principais características além de listar seus pontos positivos e negativos.

5.1 PMML

A PMML (*Predictive Model Markup Language*) é uma linguagem baseada em XML que proporciona uma forma de definição de modelos estatísticos e de mineração de dados para compartilhamento entre aplicações distintas. Além disto, esta linguagem fornece meios para o armazenamento de informações relacionadas às operações da etapa de limpeza, extração, transformação e carga dos dados, presentes no *data set* original. O órgão responsável pela definição e divulgação desta linguagem é o DMG (*Data Mining Group*), que é um grupo independente focado no desenvolvimento de formatos para mineração de dados [Dat07]. Uma vez que a PMML é descrita como sendo um documento no padrão XML, a sua especificação é fornecida na forma de um arquivo do tipo *XML Schema* que, atualmente, encontra-se na versão 3.0 [Dat07a].

A principal idéia desta especificação é fornecer uma forma independente para o intercâmbio de dados entre aplicações distintas. Através do seu uso, os usuários podem desenvolver seus modelos utilizando uma determinada aplicação (de licença proprietária ou não) e utilizar outras, de outros fornecedores ou baseadas em *software* livre, para a visualização, análise e avaliação [Dat07a]. Cada documento PMML pode conter a definição de um ou mais modelos de mineração de dados.

O elemento principal (*tag root*) é denominado com o nome da linguagem (*PMML)* e, a partir dele, todas as demais marcações são definidas. A estrutura geral de um documento, seguindo esta especificação, pode ser visualizada na Figura 12 [Dat07b].

Dentre os benefícios alcançados pelo uso deste formato para o intercâmbio de informações, um fator bastante positivo é que os documentos PMML são extremamente dinâmicos. A especificação da PMML, além de fornecer marcações

para armazenamento de informações da fase de ETL (extração, transformação e carga), também fornece *tags* especificas para a documentação de modelos de dados, incluindo diversas opções de técnicas. A definição dos elementos que são aceitos dentro da *tag* principal pode ser visualizada na Figura 13. Nela pode-se verificar que um documento seguindo a estrutura do PMML pode conter um cabeçalho, tarefas executadas durante a etapa de preparação dos dados, dicionários de dados e de transformação e, por fim, modelos os dados dos tipos suportados, tais como: Associação, *Clustering*, Redes Neurais, Seqüências e Árvores.

```
<?xml version="1.0"?>
<PMML version="3.0" xmlns="http://www.dmg.org/PMML-3_0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Header copyright="Example.com" />
    <DataDictionary>...</DataDictionary>
    <!-- ... a model ...  -->
</PMML>
```

Figura 12 - Estrutura principal de um documento PMML [Dat07b]

```
<xs:element name="PMML">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Header" />
            <xs:element ref="MiningBuildTask" minOccurs="0" />
            <xs:element ref="DataDictionary" />
            <xs:element ref="TransformationDictionary" minOccurs="0" />
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:choice>
                    <xs:element ref="AssociationModel" />
                    <xs:element ref="ClusteringModel" />
                    <xs:element ref="GeneralRegressionModel" />
                    <xs:element ref="MiningModel" />
                    <xs:element ref="NaiveBayesModel" />
                    <xs:element ref="NeuralNetwork" />
                    <xs:element ref="RegressionModel" />
                    <xs:element ref="RuleSetModel" />
                    <xs:element ref="SequenceModel" />
                    <xs:element ref="SupportVectorMachineModel" />
                    <xs:element ref="TextModel" />
                    <xs:element ref="TreeModel" />
                </xs:choice>
            </xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="version" type="xs:string" use="required" />
    </xs:complexType>
</xs:element>
```

Figura 13 - Estrutura da *tag* principal de documento PMML [Dat07b]

A especificação da PMML diz que um modelo de *Associação* representa regras onde alguns itens estão relacionados a outros. A Figura 14 ilustra a definição da marcação que representa o modelo de dados desta categoria de algortimo de mineração de dados. Analisando a estrutura representada pela figura, com base nas características de algoritmos de mineração de dados estudadas na seção 2.1, é

possível concluir que a especificação da PMML vai muito além dos atributos básicos, fornecendo informações adicionais que podem servir para melhorar a descrição do modelo, como, por exemplo, número de transações do conjunto de dados utilizadas ou o número de *itemsets* gerados.

```
<xs:element name="AssociationModel">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="MiningSchema"/>
            <xs:element ref="ModelStats" minOccurs="0"/>
            <xs:element ref="LocalTransformations" minOccurs="0" />
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="Item" />
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="Itemset" />
                <xs:element ref="AssociationRule" />
            </xs:choice>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="modelName" type="xs:string" />
        <xs:attribute name="functionName" type="MINING-FUNCTION" use="required" />
        <xs:attribute name="algorithmName" type="xs:string" />
        <xs:attribute name="numberOfTransactions" type="INT-NUMBER" use="required" />
        <xs:attribute name="maxNumberOfItemsPerTA" type="INT-NUMBER" />
        <xs:attribute name="avgNumberOfItemsPerTA" type="REAL-NUMBER" />
        <xs:attribute name="minimumSupport" type="PROB-NUMBER" use="required" />
        <xs:attribute name="minimumConfidence" type="PROB-NUMBER" use="required" />
        <xs:attribute name="lengthLimit" type="INT-NUMBER" />
        <xs:attribute name="numberOfItems" type="INT-NUMBER" use="required" />
        <xs:attribute name="numberOfItemsets" type="INT-NUMBER" use="required" />
        <xs:attribute name="numberOfRules" type="INT-NUMBER" use="required" />
    </xs:complexType>
</xs:element>
<xs:element name="Itemset">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element minOccurs="0" maxOccurs="unbounded" ref="ItemRef" />
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required" />
        <xs:attribute name="support" type="PROB-NUMBER" />
        <xs:attribute name="numberOfItems" type="xs:nonNegativeInteger" />
    </xs:complexType>
</xs:element>
<xs:element name="ItemRef">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="itemRef" type="xs:string" use="required" />
    </xs:complexType>
</xs:element>
<xs:element name="AssociationRule">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="antecedent" type="xs:string" use="required" />
        <xs:attribute name="consequent" type="xs:string" use="required" />
        <xs:attribute name="support" type="PROB-NUMBER" use="required" />
        <xs:attribute name="confidence" type="PROB-NUMBER" use="required" />
        <xs:attribute name="lift" type="xs:float" use="optional" />
        <xs:attribute name="id" type="xs:string" use="optional" />
    </xs:complexType>
</xs:element>
```

Figura 14 - Estrutura principal para modelos de associação em PMML [Dat07b]

O padrão PMML não especifica modelos da categoria *Classificação* de forma explícita. Em contrapartida a isto, descreve marcações que podem ser utilizadas para a definição de modelos de *Árvores de Decisão*. Estas *tags* podem ser utilizadas tanto na construção de estruturas de classificação quanto em estruturas de predição. A Figura 15 exibe a definição de algumas marcações que podem ser utilizadas para esta finalidade. Novamente, é possível verificar a grande diversidade de informações que podem estar presentes em documentos PMML, o que o torna extremamente versátil. Comparando estes dados com os estudos realizados pelas seções 2.2 e 3.1, pode-se verificar que o PMML possui suporte para os atributos principais, além de outros, como o valor de distribuição dos valores dos atributos de teste que representam um nodo.

```
<xs:element name="TreeModel">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="MiningSchema"/>
            <xs:element ref="Output" minOccurs="0" />
            <xs:element ref="ModelStats" minOccurs="0"/>
            <xs:element ref="Targets" minOccurs="0" />
            <xs:element ref="LocalTransformations" minOccurs="0" />
            <xs:element ref="Node"/>
            <xs:element ref="ModelVerification" minOccurs="0"/>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="modelName" type="xs:string" />
        <xs:attribute name="functionName" type="MINING-FUNCTION" use="required" />
        <xs:attribute name="algorithmName" type="xs:string" />
        <xs:attribute name="splitCharacteristic" default="multiSplit">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="binarySplit"/>
                <xs:enumeration value="multiSplit"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
</xs:complexType>
<xs:element name="Node">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
            <xs:group ref="PREDICATE" />
            <xs:choice>
                <xs:sequence>
                    <xs:element ref="Partition" minOccurs="0"/>
                    <xs:element ref="ScoreDistribution" minOccurs="0" maxOccurs="unbounded"/>
                    <xs:element ref="Node" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:group ref="EmbeddedModel"/>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="id" type="xs:string"/>
        <xs:attribute name="score" type="xs:string" use="required"/>
        <xs:attribute name="recordCount" type="NUMBER"/>
    </xs:complexType>
</xs:element>
```

Figura 15 - Estrutura principal para de árvores em PMML [Dat07b]

Uma das principais características da PMML é o fato de que toda a sua estrutura está descrita por um único esquema. Isto pode ser citado como sendo seu principal ponto forte e, também, o seu ponto fraco. A diversidade das marcações XML pode ser considerada como um fator positivo no sentido de que com apenas um documento de definição de esquemas, utilizando *XML Schema*, é possível criar arquivos com diferentes tipos de modelos de dados. Com isto, tem-se uma facilidade de uso, vez que não é necessário a existência de *n* diferentes esquemas, um para cada tipo de modelo.

Por outro lado, esta mesma característica também pode ser considerada como um fator negativo para o usuário final deste padrão. Isto pode ser concluído devido à dificuldade de manipulação do arquivo de definição do esquemas, composto por mais de 2.000 linhas. Esta característica, exigida pelos objetivos da linguagem, torna difícil a compreensão da forma de criação de algumas marcações, além de agregar um fator a mais de dificuldade no desenvolvimento de aplicações que interpretam este formato. Outro ponto negativo associado à esta característica é que não se pode, por exemplo, criar aplicações de visualização que utilizam como entrada uma estrutura fixa, visto que cada documento PMML pode conter um modelo de dados completamente diferente. Neste caso, as implementações devem ser flexíveis a ponto de manipular todos os formatos disponíveis ou gerar mensagens para uso de formatos não suportados.

Outro ponto que pode ser considerado como negativo é o fato de que a especificação não possui nenhum tipo de *framework* que facilite o seu manuseio. O uso de classes Java, por exemplo, que recebessem um conjunto de dados e retornassem o código do XML no formato PMML, ou fizesse o caminho contrário, poderia auxiliar a utilização do padrão, além de contribuir para a sua maior difusão e, conseqüentemente, uso.

5.2 Considerações

O presente capítulo apresentou o padrão PMML, que é uma especificação, descrita em XML, que proporciona uma forma de definição de modelos de dados resultantes de execuções de algoritmos de mineração. Além disto, permite a

documentação das operações realizadas durante a etapa de extração, transformação e carga. A principal idéia deste padrão é fornecer uma forma independente para intercâmbio de informações entre aplicativos distintos. Dentre os diversos modelos de dados aceitos pelo formato, descrito em um arquivo *XML Schema* de mais de 2.000 linhas, estão Associação, Seqüências, Árvores e *Clustering*.

Este capítulo foi importante para o contexto deste trabalho por fornecer uma visão de outro trabalho que está, direta ou indiretamente, buscando os mesmos objetivos propostos neste documento, ou seja, a padronização dos dados entre as etapas do processo de KDD. Após a análise da solução apresentada, foi possível perceber que o formato PMML está bem encaminhado rumo à padronização. Porém, como conseqüência da grande diversidade de modelos propostos, sua estrutura torna-se um tanto quanto difícil de ser utilizada. Com isto, aplicações que se propõem a trabalhar com este formato, devem se preocupar em tratar todos os tipos de modelos possíveis, ao invés de focar apenas uma funcionalidade.

6 Caracterização do Problema

Este capítulo apresenta a caracterização do problema de padronização entre as etapas do processo de descoberta de conhecimento, discorrendo sobre a sua importância.

6.1 Contextualização

O processo de descoberta de conhecimento é uma seqüência de sete etapas bem definidas, conforme descrito por Han [Han01] e discutido no capítulo 2 deste documento. Cada uma destas etapas é responsável por tarefas específicas, sendo que todas são importantes para o processo como um todo. Porém, o relacionamento entre elas não é algo trivial, vez que, para que se possa executar completamente o processo de KDD, pode ser necessário o desenvolvimento de todas estas etapas em um mesmo aplicativo, utilizando arquitetura e linguagem de programação idênticas, conforme ferramenta apresentada em [Gar05], [Gar05a] e [Gar05b].

Segundo [Bec06] e [Rui05], este ambiente computacional pode ser ainda mais complexo. Assim, as etapas do processo de KDD poderiam estar sendo executadas de forma independente, em um ambiente distribuído, orientado a serviços, com o uso da tecnologia Java J2EE ou, ainda, controlado por uma aplicação de orquestração de serviços (BPEL). Esta abordagem distribuída pode ser bastante benéfica para a melhoria de desempenho nas aplicações de KDD. Tanto o processo de mineração quanto o de geração da visualização podem possuir algoritmos complexos que, quando executados com uma grande quantidade de dados, demandem grande esforço computacional do servidor no qual estão instalados.

A Figura 16 ilustra uma possível arquitetura para implementação do processo de KDD de forma distribuída. Conforme pode ser visualizado na ilustração, a aplicação principal, instalada em um ou mais servidores, pode possuir todas as principais funcionalidades para interface com os usuários, além de regras para controle de acessos. A partir das operações disponíveis nos seus itens de menus, pode ser possível visualizar um resumo do conteúdo da base de dados e selecionar o conjunto desejado para o processo (*data sets*). Depois, pode ser possível optar por uma das técnicas de mineração de dados existentes e iniciar a sua execução. Ao

final deste processo, o algoritmo pode gerar um resultado a ser analisado utilizando um dos métodos de visualização acessíveis na ferramenta. Nesta arquitetura, as etapas de mineração e visualização são apresentadas como sendo duas aplicações distintas, implementadas com base nos conceitos da tecnologia J2EE (*Servlets* e *EJBs*). Para um melhor aproveitamento de recursos, cada um destes módulos pode estar instalado em um servidor independente. Em um mundo ideal, estes servidores encontram-se localizados na sede de uma mesma empresa. Porém podem estar em *data centers* separados em uma mesma cidade ou, ainda, em cidades, estados ou países diferentes.

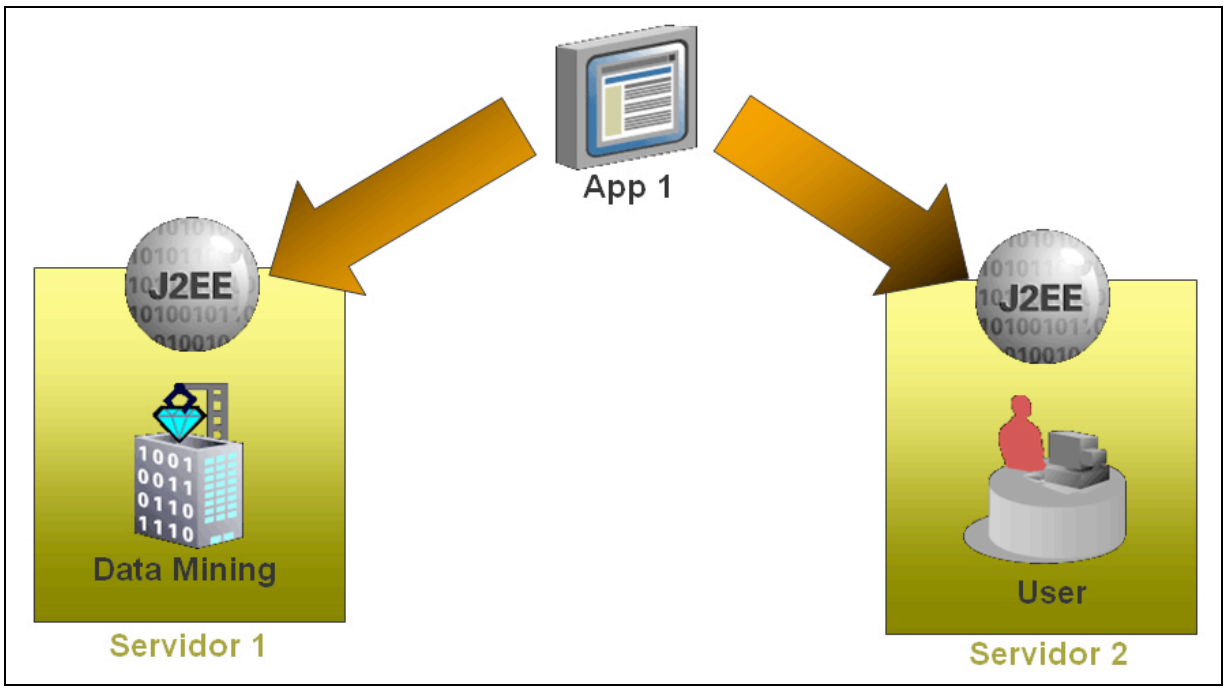


Figura 16 - Arquitetura ilustrativa de uma arquitetura distribuída

6.2 Trabalhos Anteriores

Em [Gar05], [Gar05a] e [Gar05b] é discutida uma ferramenta cuja proposta é uma arquitetura de suporte ao processo de descoberta de conhecimento sobre bases de dados de processos de negócio (*workflow*). Esta ferramenta, criada sob o conceito *web*, foi implementada seguindo uma arquitetura monolítica, ou seja, todas as etapas do processo de KDD são realizadas em uma mesma aplicação que se encontra disponibilizada em um servidor único. A Figura 17 ilustra a arquitetura da ferramenta, divida em três módulos: Importação, Análise e Visualização.

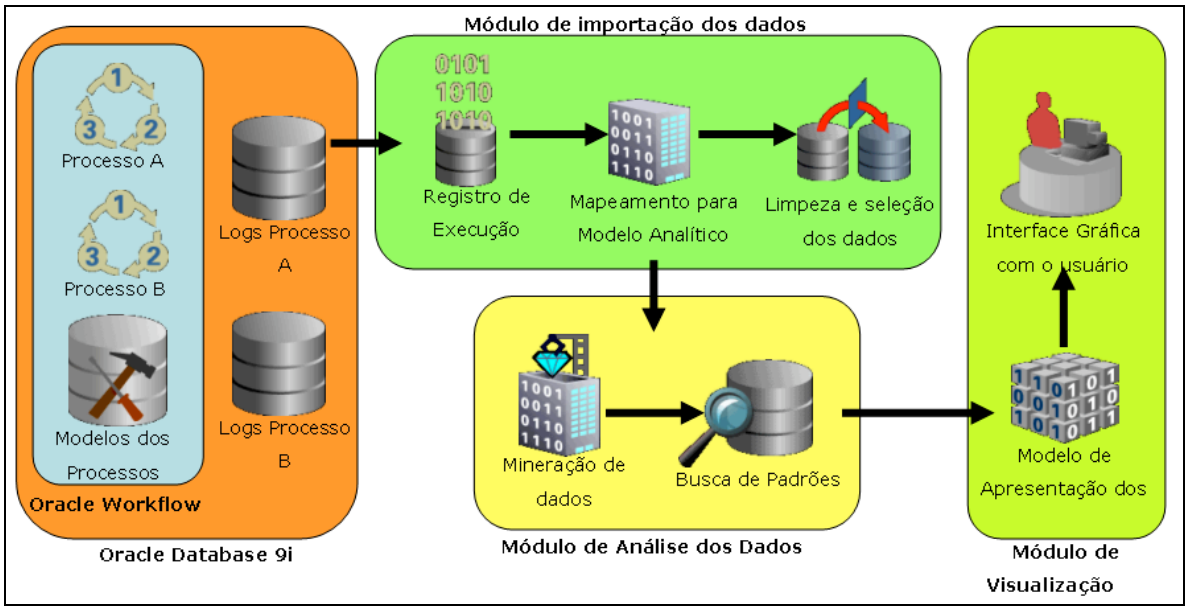


Figura 17 - Arquitetura da aplicação [Gar05]

Para que fosse possível a implementação desta ferramenta, foi utilizado o modelo analítico definido por Casati em [Cas02]. Este modelo, desenvolvido para o mapeamento de processos de negócio implementados na ferramenta *HP Process Manager 5.0* (HPPM 5.0), é composto da seguinte forma:

- Tabelas dimensão, responsáveis pelas definições dos modelos: *Proc_Defs_D* (modelos de processos), *Service_Defs_D* (serviços ou funções), *Node_Defs_D* (atividades), *Arcs_Defs* (transições entre as atividades), *Data_Defs_D* (dados), *Proc_Bhv_D* (comportamentos dos processos), *Time_D* (datas) e *Resources_D* (usuários e grupos).

- Tabelas fato, responsáveis pelas definições das instâncias: *Proc_Inst_F* (processos), *Proc_Data_Inst_F* (valores de dados para processos), *Proc_Bhv_Inst_F* (comportamentos das instâncias de processos), *Service_Inst_F* (atividades e serviços ou funções) e *Node_Inst_Data_F* (valores de dados para atividades).

O Modelo Casati é bastante abrangente e aborda diversos conceitos de processos de negócio. Porém, ele possui algumas características que, para a proposta de ferramenta discutida em [Gar05], não foram totalmente satisfatórias. Conforme analisado em [Gar06], alguns problemas deste modelo são:

- Falta de opções para o mapeamento de processos de negócio organizados na forma de subprocessos, uma prática de modelagem bastante comum;
- Ausência de forma de relacionar participantes com mais de um grupo sem que seja necessário repetir o cadastro do participante para cada grupo que ele faça parte;
- Falta de uma distinção clara entre a origem dos dados, vez que a definição do dado está associada ao modelo do processo e suas instâncias podem estar vinculadas tanto a processos como atividades.

Buscando a contínua melhoria da ferramenta e, conseqüentemente, do processo como um todo, em [Gar06] e [Gar06a] é descrito um novo modelo multidimensional denominado *Athena Model – Beta*. Este modelo é composto da seguinte forma:

- Tabelas dimensão (Figura 18), responsáveis pelas definições dos modelos: *Entities* (entidades), *Participants* (usuários e grupos), *Participants_Groups* (relacionamento entre usuários e grupos), *Entities_Participants* (relacionamento entre entidades e participantes), *Process_Groups* (grupos de processos), *Process_Definitions* (modelos de processos), *Activity_Definitions* (atividades e funções), *Process_Activities* (relacionamentos entre atividades e funções com processos e subprocessos), *Transitions* (transição entre as atividades), *Data_Definitions* (dados), *Process_Data_Definitions* (dados de processos), *Activity_Data_Definitions* (dados de atividades) e *Times* (datas).
- Tabelas fato (Figura 19), responsáveis pelas definições das instâncias: *Process_Instances* (processos), *Activity_Instances* (atividades e funções), *Data_Instances* (valores de dados), *Process_Data_Instances* (valores de dados de processos) e *Activity_Data_Instances* (valores de dados de atividades ou funções).

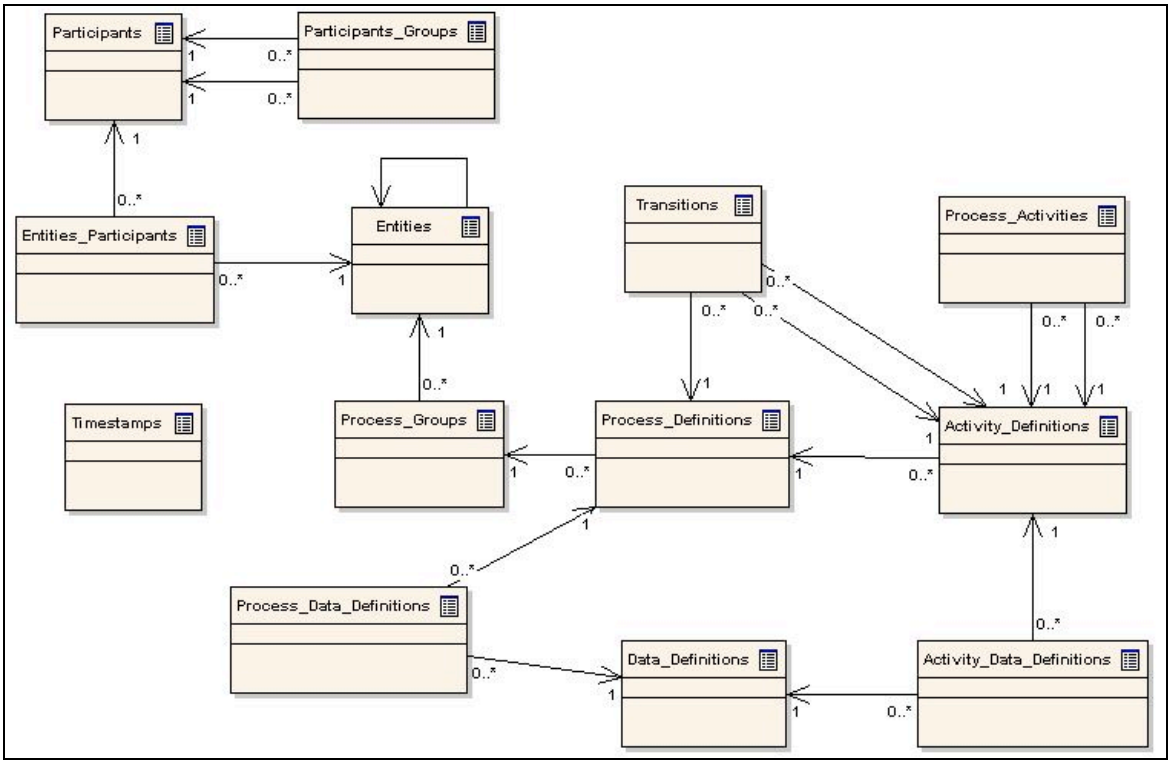


Figura 18 - Dimensões do Modelo Athena – Beta [Gar06a]

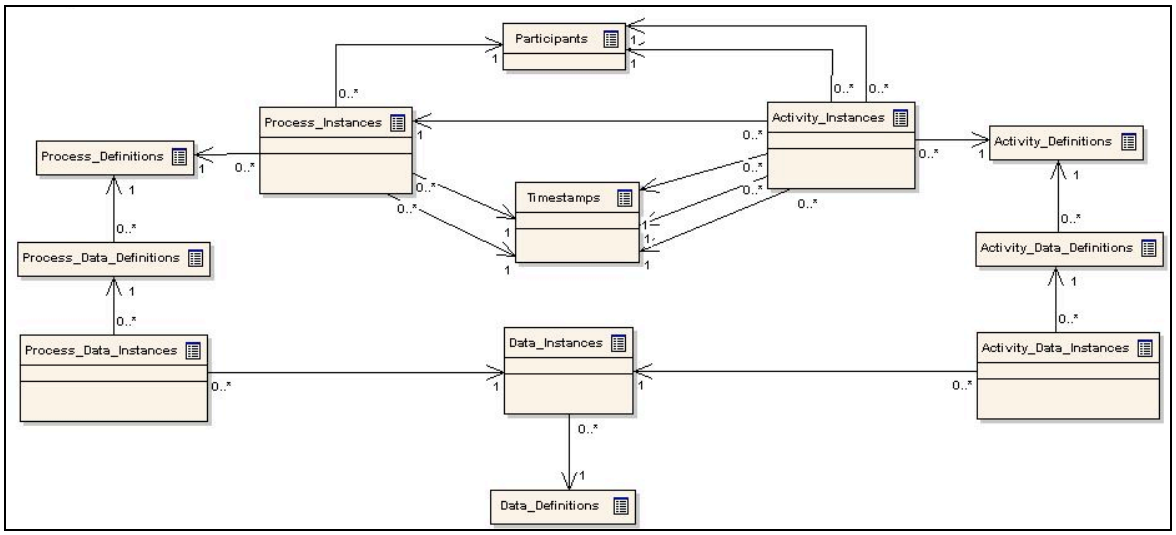


Figura 19 - Fatos do Modelo Athena – Beta [Gar06a]

Uma das propostas do modelo *Athena* era corrigir os problemas identificados com o uso do Modelo Casati [Cas02]. Para que fosse possível atendê-la, buscou-se uma maior adequação aos padrões definidos pela WfMC [Hol95]. Desta forma, foi realizado um estudo sobre a padronização de processos de negócio e, como resultado, criou-se uma lista de atributos essenciais e outra de opcionais totalizando, nas duas, 40 itens [Gar06][Gar06a]. Em um comparativo entre os dois modelos,

chegou-se a conclusão de que o modelo Casati atende a 26 atributos, enquanto que o modelo proposto, o *Model Athena - Beta*, atende a 30 atributos.

A partir do uso destes modelos analíticos o desenvolvimento da etapa de extração, transformação e carga ([Gar05a]) se deu através do uso de rotinas PL/SQL. A escolha por esta linguagem está baseada em questões de desempenho, aliado ao fato de que as bases original e analítica estão no mesmo banco de dados. Estas rotinas, então, são responsáveis por acessar a base que armazena as informações de execuções dos processos de negócio, selecionar os atributos desejados, executar a limpeza e padronização dos valores e, por fim, importar estes dados no modelo multidimensional.

O sistema de gerência de *workflow* utilizado é o *Oracle Workflow* [Cha02]. Para a base de dados foram utilizados dois processos de negócios de uma pequena empresa de desenvolvimento de *software*, contendo o registro de execução de mais de 2.500 instâncias de processos.

A ferramenta proposta em [Gar05] possui a opção de importação de arquivos contendo diferentes implementações para as etapas de mineração de dados e visualização das informações. Buscando a viabilidade deste módulo, foi proposto um *framework* de desenvolvimento utilizando a linguagem Java. Assim, para a importação de novos algoritmos de mineração de dados, o desenvolvedor do código deve definir uma classe principal que implementa uma determinada interface. Esta classe recebe os dados que devem ser utilizados, conforme contrato estabelecido pela especificação dos métodos da interface, e é responsável por executar o algoritmo e salvar seu resultado para posterior recuperação. Para as visualizações, o princípio é exatamente o mesmo, ou seja, todas elas devem possuir uma classe principal que implementa uma outra interface responsável por definir a forma como os resultados das execuções serão acessados. O autor da visualização deve então se preocupar em como o seu algoritmo se comporta e transmite suas informações.

Após a importação das implementações, a ferramenta ainda fornece uma forma de definir relacionamento entre elas, ou seja, é possível especificar quais as técnicas de visualização podem ser utilizadas para a análise dos resultados de um determinado algoritmo de mineração. Este relacionamento, no entanto, é realizado

por um usuário com perfil de administração do sistema e é livre, não existindo nenhuma restrição que impeça o uso de visualizações incompatíveis com determinados algoritmos. Assim, para que as implementações de técnicas de visualização possam funcionar corretamente, os seus autores devem conhecer a estrutura na qual os dados referentes aos resultados dos métodos de mineração estão dispostos, independente de formato e mídia. Esta característica faz com que as visualizações adicionadas ao sistema tenham a necessidade de serem projetadas de forma específica para as técnicas de mineração.

6.3 Descrição do problema

No contexto apresentado nas seções anteriores, pode-se perceber que é importante a organização da forma como a informação é trocada entre as etapas de mineração e de visualização, do processo de KDD. Esta comunicação, sendo realizada de forma única, pode facilitar o desenvolvimento de aplicações de mineração de dados, além da integração entre bases distintas. Para isto, torna-se necessário que as entradas e saídas dessas etapas do processo de descoberta de conhecimento possuam um determinado nível de padronização, de forma que consigam ser executadas de maneira independente, mas, ainda assim, estarem integradas. Além disto, as diferentes etapas podem estar sendo desenvolvidas por pessoas dispostas em várias localidades do mundo.

O problema abordado por este trabalho é, então, como realizar a padronização da troca de informações entre as etapas de mineração de dados e visualização dos resultados do processo de KDD.

Neste sentido, para que este problema possa ser resolvido, os requisitos de uma solução ideal são:

- Padronização da forma de geração dos resultados dos algoritmos de mineração de diferentes classes;
- Padronização da maneira como técnicas de visualização distintas lêem seus dados de entrada;
- Existência de um *framework* para a transformação automática entre os formatos de mineração para os de visualização.

6.4 Análise do Problema

O cenário descrito anteriormente é caracterizado como sendo um problema interessante de ser tratado, vez que existem diferentes classes de algoritmos de mineração de dados e técnicas de visualização de informação. Somado a isto, existe o problema de compatibilidade de uso entre determinadas técnicas de análise visual de dados com certas classes de algoritmos de mineração.

No capítulo 5 deste trabalho foi apresentada a especificação PMML. Este formato busca exatamente a citada padronização, sendo constituído por apenas um esquema e contendo definições para diversos tipos de algoritmos de mineração. Apesar de toda a sua diversidade, o PMML não resolve totalmente o problema descrito anteriormente, visto que não trata as técnicas de visualização. Esta característica faz com que as implementações de métodos deste tipo, para que possam trabalhar com o formato, tenham que entender a maneira como os resultados dos algoritmos de mineração estão organizados.

Por outro lado, mesmo com um foco maior na integração de diferentes algoritmos e visualizações, a ferramenta proposta em [Gar05] também não resolve o problema da padronização. Como a instalação e definição dos relacionamentos entre as implementações estão a cargo de um usuário com perfil de administrador, não existe uma forma de garantir uma troca de informações padrão, forçando as implementações a conhecerem as estruturas umas das outras.

Os capítulos 2 e 0 apresentaram estudos sobre as classes de mineração de dados e as técnicas de visualização de informação, respectivamente. Estes estudos foram importantes por fornecerem os requisitos necessários para a compreensão das estruturas de dados que são utilizadas. Tanto os resultados dos algoritmos de mineração quanto os dados necessários para as técnicas de visualização podem estar disponíveis em estruturas de dados e formatos diversos. Entre eles, pode-se destacar o uso de arquivos textuais e do tipo CSV, onde os dados encontram-se separados uns dos outros pelo uso de vírgulas. Algumas ferramentas de mineração, no entanto, possuem estruturas específicas para o armazenamento dos seus dados.

Conforme comentado anteriormente, este trabalho está utilizando o *software Weka* [Wit05] como base para os testes da solução. Esta ferramenta, implementada em linguagem de programação Java, armazena os resultados das execuções dos seus algoritmos de mineração em estruturas internas das suas classes. Para a visualização a ferramenta apenas imprime as informações de forma textual em uma determinada área da aplicação. A Figura 20 ilustra este comportamento, exibindo o resultado da execução do algoritmo J48, uma implementação do C4.5. Nela é possível perceber o formato padrão de como este *software* gera suas visualizações. Durante esta operação, nenhum arquivo é criado, nem mesmo que de forma temporária.

Algumas categorias de algoritmos, como o próprio J48 ou *Redes Bayesianas*, possuem áreas de análise de dados próprias, que podem exibir as informações de forma gráfica. A Figura 21 ilustra esta visualização para o mesmo resultado da execução do algoritmo J48 apresentado na figura anterior. Para que estas funcionalidades possam ser acionadas, é necessário que as informações sobre as execuções dos algoritmos, armazenadas em memória, sejam transformadas em textos seguindo um determinado formato, especifico de cada implementação. A partir disto, os algoritmos são responsáveis por criar suas estruturas internas e gerar o componente de visualização correspondente.

6.5 Considerações

Este capítulo discorreu sobre a caracterização do problema que o trabalho se propõe a resolver. Conforme apresentado, o processo de descoberta de conhecimento pode envolver algoritmos de mineração e técnicas de visualização complexos que, durante sua execução, necessitam de muitos recursos computacionais, prejudicando o rendimento das aplicações. Para estes casos, a melhor solução seria uma abordagem distribuída para as ferramentas de KDD.

A ferramenta descrita em [Gar05] propõe um ambiente computacional para a execução de processos de descoberta de conhecimento sobre bases de dados de *workflow*. Para o modelo analítico, foram utilizadas duas versões da mesma abordagem, sendo que a primeira foi desenvolvida por Casati [Cas02], enquanto que

a segunda, é o modelo proposto por Garcia [Gar06], [Gar06a]. Esta ferramenta fornece ainda uma maneira de importar e relacionar algoritmos de mineração e técnicas de visualização. Para a importação, é necessário que as implementações utilizem, como base, duas interfaces definidas através de um *framework* da própria aplicação. Já o relacionamento é realizado de forma livre por um usuário com perfil de administração.

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     weather-weka.filters.unsupervised.attribute.Discretize-B5-M-1.0-Rfirst-last
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
              play
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

outlook = sunny
|   humidity = '(-inf-71.2]': yes (2.0)
|   humidity = '(71.2-77.4]': no (0.0)
|   humidity = '(77.4-83.6]': no (0.0)
|   humidity = '(83.6-89.8]': no (1.0)
|   humidity = '(89.8-inf)': no (2.0)
outlook = overcast: yes (4.0)
outlook = rainy
|   windy = TRUE: no (2.0)
|   windy = FALSE: yes (3.0)

Number of Leaves  :     8

Size of the tree :      11
```

Figura 20 - Resultado da execução de algoritmo J48 no *Weka*

Assim, tem-se que a proposta principal deste trabalho é criar uma forma de relacionamento padrão entre duas importantes etapas do processo de KDD: mineração e visualização, sendo que serão utilizadas apenas as técnicas de classificação baseadas em *Árvores de Decisão*. Este problema será resolvido através da padronização dos resultados de algoritmos e das entradas de

visualizações. Além disto, a existência de um *framework* para a transformação automática entre eles é de extrema relevância.

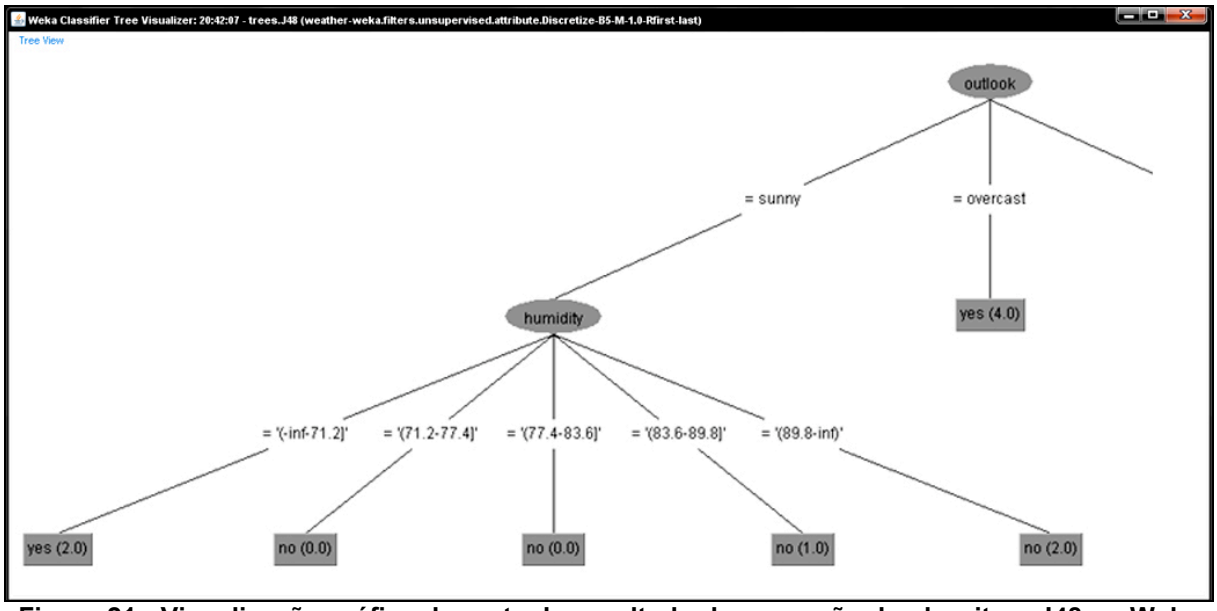


Figura 21 - Visualização gráfica de parte do resultado da execução do algoritmo J48 no Weka

7 Trabalhos Realizados

Este capítulo visa descrever a proposta para solução do problema apresentado no capítulo anterior, além de relatar todos os passos realizados para a criação da solução.

7.1 Solução Proposta

A partir do problema exposto no capítulo anterior, que não pode ser resolvido totalmente com o uso da linguagem PMML, pois a mesma limita-se a padronizar a saída de algoritmos de mineração, torna-se necessário a definição de uma solução que atenda as necessidades apresentadas. Assim, o foco principal da proposta deste trabalho é a definição de uma forma de padronização para troca de informações entre duas etapas do processo de descoberta de conhecimento: mineração de dados e visualização dos resultados.

7.2 Esquemas para Troca de Informações

A partir do estudo sobre a linguagem XML, apresentado no capítulo 4, foi possível concluir que esta tecnologia é a melhor escolha quando o assunto é troca de informações entre aplicações distintas. Desta forma, com base no problema descrito no capítulo 6 deste documento, a solução proposta é a criação de diferentes esquemas, utilizando a técnica de *XML Schema*. Para as classes de mineração de dados, os esquemas devem representar a estrutura como os resultados dos seus algoritmos são gerados, independente de implementação, sendo que deve existir um esquema para cada classe de algoritmo. Porém, para as técnicas de visualização, os esquemas indicam o formato como os dados de entrada devem estar organizados, existindo, também, um esquema para cada tipo de método.

7.2.1 Esquemas de Algoritmos de Mineração

Para a criação dos esquemas responsáveis por representar a estrutura dos algoritmos de mineração de dados (*XML Schemas*), foram utilizadas as duas classes estudadas no capítulo 2 deste trabalho: *Associação* e *Classificação*. O esquema tem um cabeçalho, com informações de identificação da base de dados, como o nome e descrição, além do algoritmo de mineração utilizado. A Figura 22 ilustra a definição

desta marcação, denominada *Info*, utilizando a notação *XML Schema*. Nela podem ser visualizadas as estruturas que representam o nome do conjunto de dados (*Name*), a sua descrição (*Description*) e o nome algoritmo de mineração responsável pelo resultado do arquivo (*Algorithm*).

```
<xs:complexType name="InfoType">
    <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1" />
        <xs:element name="Algorithm" type="xs:string" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
```

Figura 22 - Definição da *tag* de informações

7.2.1.1 Associação

A criação do esquema que representa a estrutura de saída de algoritmos de *Associação* foi realizada com base nas características estudadas na seção 2.1. Foi verificado que a saída desta classe apresenta uma lista composta por várias regras. Assim, a estrutura principal do seu esquema é uma marcação que pode conter uma lista de objetos, denominada *Rules*. Para representar cada uma das possíveis regras, foi criado um componente específico, chamado *Rule*.

Seguindo as definições da classe, tem-se que cada regra de associação é uma estrutura que possui quatro dados: cabeça, corpo, valor de confiança e valor de suporte. Desta forma, o esquema do componente *Rule* foi definido como sendo um tipo de dado complexo. Nele estão definidos dois novos elementos, um representando a cabeça da regra, denominado *Head*, e outro o corpo, definido como *Body*. Além deles, existem duas *tags* numéricas para os valores de suporte e confiança denominadas, respectivamente, *Support* e *Confidence*.

A definição das estruturas que representam a cabeça e o corpo de regras de associação especifica que os objetos devem ser compostos por um ou mais atributos. Desta forma, o tipo de dado complexo que representa as marcações *Head* e *Body,* é definido de forma semelhante ao utilizado para a lista de regras, ou seja, uma estrutura de dados que armazena uma lista de objetos. Porém, neste caso, esta lista é composta por marcações que representam atributos, denominada *Attribute*. Seguindo a definição de que cada atributo presente em uma regra é representado

pelo seu nome e valor assumido, tem-se que o conteúdo da marcação *Attribute* são duas *tags*: *Name* e *Value*.

Demonstrando o esquema especificado, a Figura 23 exibe um trecho do código de definição de arquivos XML para resultados de algoritmos de mineração da classe *Associação*. O esquema completo está disponível no Anexo I.

```
<xs:element name="Association" type="AssociationType" />
<xs:complexType name="AssociationType">
    <xs:sequence>
        <xs:element name="Info" type="InfoType" minOccurs="1" maxOccurs="1" />
        <xs:element name="Rules" type="RulesType" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="RulesType">
    <xs:sequence>
        <xs:element name="Rule" type="RuleType" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="RuleType">
    <xs:sequence>
        <xs:element name="Head" type="AttributeListType" minOccurs="1" maxOccurs="1" />
        <xs:element name="Body" type="AttributeListType" minOccurs="1" maxOccurs="1" />
        <xs:element name="Support" type="xs:double" minOccurs="1" maxOccurs="1" />
        <xs:element name="Confidence" type="xs:double" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="AttributeListType">
    <xs:sequence>
        <xs:element name="Attribute" type="AttributeType" minOccurs="1"
            maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="AttributeType">
    <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="Value" type="xs:string" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
```

Figura 23 - Esquema XML para a classe *Associação*

7.2.1.1.1 Estudo de Caso

Objetivando a validação do esquema proposto, foram realizadas alguns testes. Para isto, foram utilizadas as implementações, fornecidas pela ferramenta *Weka* [Wit05], de dois algoritmos da classe *Associação*: o *Apriori* e o *Tertius* apresentados, respectivamente, nas seções 2.1.1 e 2.1.3.

A comparação entre a saída fornecida originalmente pela aplicação *Weka* e o arquivo XML gerado com base no esquema proposto, foi realizada a partir da execução dos respectivos algoritmos para um mesmo conjunto de dados. Para isto, foi selecionado um dos arquivos de teste fornecido juntamente com a instalação desta ferramenta, denominado *weather*. O *weather* é composto por um total de 14

registros e, para cada um, estão listados quatro atributos que indicam características climáticas: condição do tempo (*outlook*), temperatura (*temperature*), umidade (*humidity*) e existência de ventos (*windy*). Com base nos seus valores, um quinto atributo (*play*) informa se um determinado esporte é praticado ou não.

O resultado da execução do algoritmo *Apriori* no *Weka*, ilustrado pela Figura 24, mostra que, para o conjunto de dados selecionado, foram criadas 10 regras de associação. Já a execução do algoritmo *Tertius,* para os mesmos dados, cujo resultado pode ser visualizado na Figura 25, gerou um total de 25 regras.

```
Apriori
=======

Best rules found:

 1. outlook=overcast 4 ==> play=yes 4    conf:(1)
 2. temperature='(80.8-inf)' 3 ==> windy=FALSE 3    conf:(1)
 3. outlook=rainy play=yes 3 ==> windy=FALSE 3    conf:(1)
 4. outlook=rainy windy=FALSE 3 ==> play=yes 3    conf:(1)
 5. humidity='(77.4-83.6]' 2 ==> outlook=rainy 2    conf:(1)
 6. temperature='(72.4-76.6]' 2 ==> play=yes 2    conf:(1)
 7. humidity='(83.6-89.8]' 2 ==> temperature='(80.8-inf)' 2    conf:(1)
 8. humidity='(77.4-83.6]' 2 ==> windy=FALSE 2    conf:(1)
 9. humidity='(77.4-83.6]' 2 ==> play=yes 2    conf:(1)
10. humidity='(83.6-89.8]' 2 ==> windy=FALSE 2    conf:(1)
```

Figura 24 - Resultado do algoritmo *Apriori* no *Weka*

7.2.1.1.2 Apriori

A implementação Java do algoritmo *Apriori*, fornecida juntamente com o *Weka*, foi criada pela Universidade de Waikato e sua classe principal é a *weka.associations.Apriori*. Para iniciar a sua execução, de acordo com a interface padrão do *Weka* para algoritmos, é necessário um objeto do tipo *weka.core.Instances*. Nele devem estar armazenados todos os registros recuperados do conjunto de dados que está sendo utilizado para o processo de mineração. Internamente esta classe contém um vetor com todos os atributos presentes no conjunto de dados e, para cada um deles, existe uma outra lista contendo todos os seus possíveis valores. Assim, um registro é armazenado como sendo uma estrutura composta por vários objetos, cada um com dois índices

numéricos. O primeiro representa a posição do vetor referente ao atributo constante no registro, enquanto o segundo indica o valor assumido pelo atributo, conforme definido na lista de valores correspondente.

```
Tertius
=======

 1. /* 0.727325 0.142857 */ play = yes ==> temperature = '(72.4-76.6]' or humidity = '(77.4-83.6]' or outlook = overcast
 2. /* 0.642658 0.000000 */ humidity = '(89.8-inf)' ==> temperature = '(68.2-72.4]' or play = no
 3. /* 0.642658 0.000000 */ humidity = '(89.8-inf)' ==> temperature = '(68.2-72.4]' or outlook = sunny
 4. /* 0.642658 0.000000 */ temperature = '(68.2-72.4]' ==> humidity = '(89.8-inf)' or outlook = sunny
 5. /* 0.642658 0.000000 */ outlook = sunny ==> humidity = '(-inf-71.2]' or play = no
 6. /* 0.641109 0.214286 */ windy = FALSE ==> temperature = '(80.8-inf)' or humidity = '(77.4-83.6]'
 7. /* 0.641109 0.214286 */ play = yes ==> temperature = '(72.4-76.6]' or outlook = overcast
 8. /* 0.641109 0.214286 */ play = yes ==> humidity = '(77.4-83.6]' or outlook = overcast
 9. /* 0.633754 0.071429 */ humidity = '(89.8-inf)' ==> temperature = '(68.2-72.4]'
10. /* 0.633754 0.071429 */ temperature = '(68.2-72.4]' ==> humidity = '(89.8-inf)'
11. /* 0.590214 0.000000 */ play = yes ==> temperature = '(72.4-76.6]' or windy = FALSE or outlook = overcast
12. /* 0.590214 0.000000 */ temperature = '(80.8-inf)' ==> humidity = '(83.6-89.8]' or outlook = overcast
13. /* 0.555556 0.000000 */ humidity = '(89.8-inf)' ==> temperature = '(68.2-72.4]' or windy = TRUE
14. /* 0.555556 0.000000 */ play = no ==> windy = TRUE or outlook = sunny
15. /* 0.538289 0.000000 */ humidity = '(-inf-71.2]' ==> temperature = '(-inf-68.2]' or outlook = sunny
16. /* 0.538289 0.000000 */ windy = TRUE and play = no ==> temperature = '(76.6-80.8]' or outlook = rainy
17. /* 0.538289 0.000000 */ windy = TRUE ==> temperature = '(68.2-72.4]' or humidity = '(-inf-71.2]' or play = no
18. /* 0.538289 0.000000 */ windy = TRUE ==> temperature = '(68.2-72.4]' or humidity = '(-inf-71.2]' or outlook = sunny
19. /* 0.535599 0.071429 */ windy = FALSE ==> temperature = '(80.8-inf)' or humidity = '(77.4-83.6]' or outlook = sunny
20. /* 0.531930 0.142857 */ windy = FALSE and play = yes ==> temperature = '(80.8-inf)' or humidity = '(77.4-83.6]'
21. /* 0.531930 0.142857 */ windy = TRUE ==> temperature = '(76.6-80.8]' or humidity = '(-inf-71.2]'
22. /* 0.531930 0.142857 */ play = yes ==> temperature = '(72.4-76.6]' or humidity = '(-inf-71.2]' or outlook = overcast
23. /* 0.531930 0.142857 */ play = yes ==> temperature = '(-inf-68.2]' or humidity = '(-inf-71.2]' or outlook = overcast
24. /* 0.503827 0.000000 */ humidity = '(83.6-89.8]' ==> temperature = '(80.8-inf)'
25. /* 0.503827 0.000000 */ windy = FALSE and outlook = overcast ==> temperature = '(80.8-inf)'

Number of hypotheses considered: 3588
Number of hypotheses explored: 1422
```

Figura 25 - Resultado do algoritmo *Tertius* no *Weka*

As regras resultantes da execução desta implementação são armazenadas em um vetor de objetos do tipo *weka.core.FastVector*. A primeira e a segunda posição deste vetor são compostas por novos componentes, os quais representam a cabeça e o corpo de todas as regras geradas. Tais componentes são constituídos por listas relacionadas entre si pelos valores dos seus índices. Exemplificando: o objeto presente na posição zero da primeira lista representa a cabeça da regra cujo corpo esta armazenado na posição zero da segunda lista.

Tanto a cabeça quanto o corpo das regras de associação armazenam seus dados dentro de uma instância da classe *weka.associations.AprioriItemSet*. Nela existem dois novos vetores contendo índices que referenciam os atributos e os valores salvos nas listas do objeto *Instances*.

A terceira posição da lista de objetos da classe *FastVector* armazena um vetor de valores numéricos que representam a confiança de cada uma das regras.

Esta lista segue a mesma lógica das anteriores, ou seja, a posição zero indica o valor de confiança da regra representada pelas posições zero das listas armazenadas nas duas primeiras posições do vetor. Por fim, o valor de suporte de cada regra é armazenado como sendo uma variável interna da classe *AprioriItemSet*, estando disponível apenas para os objetos que representam a cabeça das regras.

Para o mapeamento do resultado da execução do algoritmo *Apriori* para o arquivo XML, é necessário a execução dos seguintes passos:

- Criar a *tag* principal do arquivo, denominada *Rules*;
- Iterar sobre as listas das duas primeiras posições do objeto *FastVector*;
- Para cada par de posições destas listas que representam um regra, gerar a estrutura corresponde, ou seja, a marcação do tipo *Rule*;
- Gerar para cada regra, por padrão, as *tags Head* e *Body*;
- Percorrer os vetores do objeto *AprioriItemSet,* correspondentes aos indíces que representam a cabeça e corpo de cada regra no objeto *FastVector*;
- Para cada posição destes vetores, criar a estrutura da *tag Attribute*;
- Criar, dentro da *tag Attribute*, as marcações *Name* e *Value* e preenchê-las com o nome e o valor dos atributos recuperados da classe *Instances*, conforme os índices informado nos vetores;
- Após a marcação que define o corpo, criar a *tag Confidence*, preenchendo seu valor com o número armazenado do índice correspondente à regra na terceira posição do vetor da classe *FastVector*;
- Para cada regra, com base no valor da variável *m_counter* do objeto *AprioriItemSet* da estrutura que representa a cabeça, gerar uma *tag* do tipo *Support*.

A Figura 26 apresenta uma parte do arquivo XML resultante deste mapeamento, utilizando, como entrada para a execução do algoritmo, o conjunto de dados descrito na seção 7.2.1.1.1. Nesta imagem, pode ser visualizada apenas uma

das 10 regras geradas, sendo que o arquivo completo está disponível no Anexo III. Estas são as mesmas 10 regras de associação resultantes da execução do algoritmo no *software Weka*, exibidas na Figura 24

```
<Rules>
    <Rule>
        <Head>
            <Attribute>
                <Name>outlook</Name>
                <Value>overcast</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>play</Name>
                <Value>yes</Value>
            </Attribute>
        </Body>
        <Support>4.0</Support>
        <Confidence>1.0</Confidence>
    </Rule>
</Rules>
```

Figura 26 - Resultado do algoritmo *Apriori* exportado como arquivo XML

7.2.1.1.3 Tertius

A implementação Java do algoritmo *Tertius*, fornecida juntamente com o *software Weka*, é de criação de Amelie Deltour e foi desenvolvida com base na versão original, escrita na linguagem C por Flach [Fla01]. O componente principal desta implementação é a classe *weka.associations.Tertius*. Para iniciar a execução, também é necessário um objeto do tipo *Instances,* contendo todos os registros do conjunto de dados, conforme descrito na seção anterior. A estrutura de armazenamento de resultados do *Tertius* é bastante diferente da utilizada na implementação do *Apriori*. Enquanto que o *Apriori* armazena diversas listas de índices para os atributos e valores salvos na classe *Instances*, o *Tertius* utiliza melhor os conceitos de orientação a objetos e, cada estrutura, possui uma classe distinta com seus devidos atributos.

Assim, no *Tertius* as regras de associação são todas salvas em uma instância da classe *weka.associations.tertius.SimpleLinkedList*, que possui uma implementação do conceito de listas encadeadas. Nela, cada nodo é composto por um objeto do tipo *weka.associations.tertius.Rule*, representando uma regra.

Tanto a cabeça quanto o corpo das regras, por terem estruturas semelhantes, são armazenados como sendo instâncias da classe

weka.associations.tertius.LiteralSet. Nela, existe uma lista de objetos que representa todos os atributos que fazem parte do componente, sendo que cada um deles é um objeto do tipo *weka.associations.tertius.AttributeValueLiteral*. Esta classe permite que o nome do atributo seja recuperado através da execução do método *getPredicate*, enquanto que o valor correspondente deve ser extraído de uma variável interna, denominada *m_value*.

Os valores de confiança e suporte devem ser recuperados a partir dos valores de confirmação e freqüência observada para cada regra, acessíveis pelos métodos *getConfirmation* e *getObservedFrequency* da classe *Rule*. Como o Tertius trabalha de forma um pouco diferente do padrão de algoritmos de associação, ele não possui valores explícitos para confiança e suporte, sendo necessário um mapeamento das suas informações semelhantes.

Para o mapeamento do resultado da execução do algoritmo *Tertius* para o arquivo XML, é necessário a execução dos seguintes passos:

- Criar a *tag* principal do arquivo, denominada *Rules*;
- Iterar sobre a lista encadeada de objetos do tipo *Rule*, armazenados na classe *SimpleLinkedList*;
- Para cada objeto recuperado, criar uma *tag* do tipo *Rule*.
- Gerar para cada regra, por padrão, as *tags Head* e *Body*;
- Para cada objeto do tipo *LiteralSet*, que representa a cabeça ou o corpo da regra, percorrer a lista de atributos correspondente;
- Cada instância da classe *AttributeValueLiteral* recuperada deve resultar na criação da marcação *Attribute*;
- Executar o método *getPredicate* da classe *AttributeValueLiteral* e salvar o valor resultante na marcação *Name* do atributo correspondente;
- Extrair o valor da variável *m_value* da classe *AttributeValueLiteral* e salvar na marcação *Value* do atributo correspondente;

- Após a marcação que define o corpo, criar a *tag Confidence* e preencher seu valor com o resultado da execução do método *getConfirmation* da instância que representa a regra;
- Da mesma forma, o valor para a *tag Support* é o retorno da execução do método *getObservedFrequency* do objeto da classe *Rule*.

A Figura 27 apresenta o arquivo XML resultante deste mapeamento, utilizando, como entrada para a execução do algoritmo, o mesmo conjunto de dados descrito na seção 7.2.1.1.1. Nesta imagem, pode ser visualizada apenas uma das 25 regras geradas, sendo que o arquivo completo está disponível no Anexo III. Estas são as mesmas 25 regras de associação resultantes da execução do algoritmo do *software Weka*, exibidas na Figura 25. É importante destacar nesta ferramenta as regras estão dispostas ao contrário, ou seja, primeiro é mostrado o corpo da regra e, depois, a cabeça.

```
<Rules>
    <Rule>
        <Head>
            <Attribute>
                <Name>temperature</Name>
                <Value>(72.4-76.6]</Value>
            </Attribute>
            <Attribute>
                <Name>humidity</Name>
                <Value>(77.4-83.6]</Value>
            </Attribute>
            <Attribute>
                <Name>outlook</Name>
                <Value>overcast</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>play</Name>
                <Value>yes</Value>
            </Attribute>
        </Body>
        <Support>0.727325</Support>
        <Confidence>0.142857</Confidence>
    </Rule>
</Rules>
```

Figura 27 - Resultado do algoritmo *Tertius* exportado como arquivo XML

7.2.1.1.4 Generalização

Com base nas experimentações apresentadas nas seções anteriores, é possível chegar a uma conclusão quanto ao procedimento padrão para o mapeamento de resultados de execução de algoritmos de mineração da classe *Associação* para um arquivo XML. Assim, independente das estruturas de dados

utilizadas ou da forma como foi realizada a implementação do algoritmo, para a criação do arquivo para uma nova implementação, é necessário seguir os seguintes passos:

- Identificar qual a estrutura que armazena a lista de regras de associação e criar no arquivo XML a marcação *Rules*;
- Percorrer esta lista e, para cada objeto recuperado criar, no arquivo XML, uma nova *tag Rule*;
- Para cada objeto que representa uma regra, identificar os componentes que armazenam as informações referentes à cabeça e ao corpo, criando as marcações *Head* e *Body* no arquivo XML;
- Percorrer a estrutura que representa a cabeça e o corpo da regra identificando as estruturas correspondentes aos atributos;
- Para cada atributo, na marcação correspondente (*Head* ou *Body)* do arquivo XML, criar uma nova *tag Attribute*;
- Identificar a forma como o nome e o valor dos atributos são salvos em cada objeto, bem como recuperá-los e salvar seus dados nas marcações *Name* e *Value* correspondentes;
- Identificar, no objeto que representa a regra, os valores de suporte e confiança criando, dentro da marcação correspondente, as *tags Confidence* e *Support* com seus valores respectivos.

7.2.1.1.5 Comparação entre Formatos

Após a definição do formato para o armazenamento de resultados de algoritmos do tipo *Associação*, é possível traçar uma comparação entre este formato e o PMML, apresentado no capítulo 5.

O PMML utiliza a marcação *AssociationRule* para definir a sua lista de regras, enquanto que o formato proposto por este documento utiliza uma *tag* identificada por *Rule*. A primeira possui uma série de outros atributos, utilizando para identificação de parâmetros do modelo, enquanto que a segundo não possui outras definições.

Nos dois formatos, cada regra é composta pelos quatro elementos básicos: cabeça, corpo, confiança e suporte. No PMML, está estrutura é bastante complexa pois deve ser definida uma marcação do tipo *ItemSet* que irá conter os dados de todos os atributos que compõem a cabeça e o corpo. Cada atributo é definido como uma *tag* do tipo *ItemRef*. O relacionamento entre as marcações é realizado a partir de identificadores. Desta forma, a marca *AssocationRule* deve possuir os identificadores dos componentes do tipo *ItemSet* que representam a cabeça e corpo da regra. Estes, por sua vez, devem possuir o identificador de todos os elementos do tipo *ItemRef* que definem os pares de atributos e valores. No formato proposto por este documento, esta estrutura é mais simples pois basta a definição de marcações específicas para os quatro elementos. Dentro da cabeça e corpo, existem os atributos dispostos sem relacionamentos.

Com base neste breve comparativo, pode-se perceber que o novo formato proposto é mais simples e de fácil manuseio que o PMML. Isto acontece pois não existe a necessidade de definição de diversas marcações com identificadores diferentes que devem estar relacionados entre si.

7.2.1.2 Classificação

A criação do esquema que representa a estrutura de saída de algoritmos de *Classificação* foi realizada com base nas características estudadas na seção 2.2. Foi verificado que a principal forma de apresentação dos resultados desta classe utiliza uma estrutura de árvores de decisão, composta por nodos e arcos. Assim, optou-se por utilizar esta forma de organização dos resultados.

Os algoritmos de *Classificação* utilizam os valores de um dos atributos do conjunto de dados para representar a informação de categorização dos registros. Deste modo, esta informação deve estar presente nos arquivos XML resultantes do processo. Assim, a estrutura da marcação *Info*, descrita na seção 7.2.1, também é definida para esta categoria, recebendo, além das informações principais, um elemento, denominado *ClassLabelAttribute*. A definição desta marcação, em *XML Schema,* pode ser visualizada na Figura 28.

O componente principal deste esquema possui a denominação de *Attribute* para manter compatibilidade com as nomenclaturas utilizadas pelos algoritmos e representa o nodo raiz. Seguindo as definições da classe, tem-se que cada nodo de uma árvore de decisão é composto pelo nome do atributo de teste e por uma lista de arcos com outros nodos, representando seus possíveis valores. Desta forma, o esquema do componente *Attribute* foi definido como sendo um tipo de dado complexo. Nele, estão definidos dois novos elementos: um correspondente ao nome do atributo, denominado *Name*, e outro equivalente à lista de resultados, denominada *Results*.

```
<xs:complexType name="InfoType">
    <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1" />
        <xs:element name="Algorithm" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="ClassLabelAttribute" type="xs:string" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
```

Figura 28 - Definição da *tag* de informações para algoritmos de *Classificação*

Pela definição da categoria de *Classificação*, existem dois tipos de nodos:

- Internos: representam um atributo de testes e possuem uma lista de arcos de saída;
- Folha: representam o atributo de classificação dos registros possuindo apenas o seu valor.

A definição da estrutura *Results*, então, apresenta uma lista de elementos do tipo *Result*. Cada resultado é composto pelo valor e por um componente de escolha. O valor é representado pela marcação *Value* e corresponde ao arco de saída de um nodo. Logo, se um determinado atributo possuir 5 valores no conjunto de dados original, pela definição do esquema do arquivo XML, será transformado em um elemento do tipo *Attribute,* contendo uma lista de cinco elementos do tipo *Result*, cada um mantendo um dos seus possíveis valores.

O componente de escolha da marcação *Result* indica se o nodo é do tipo interno ou folha. Os nodos do tipo folha possuem o valor do atributo de classificação mapeado para o elemento *Label*. Por outro lado, os nodos internos possuem a definição de qual atributo representa o próximo teste, definido por uma nova *tag* do

tipo *Attribute*. A especificação desta marcação segue exatamente a mesma lógica exposta anteriormente.

Demonstrando o esquema especificado, a Figura 29 exibe um trecho do código de definição de arquivos XML para resultados de algoritmos de mineração do tipo *Classificação*. O esquema completo está disponível no Anexo I.

7.2.1.2.1 Estudo de Caso

Objetivando a validação do esquema proposto, foram realizadas alguns testes. Para isto, foram utilizadas as implementações, fornecidas pela ferramenta *Weka* [Wit05], de dois algoritmos de *Classificacao*: o *ID3* e o *C4.5*, apresentados, respectivamente, nas seções 2.2.1 e 2.2.2.

A comparação entre a saída fornecida originalmente pela aplicação *Weka* e o arquivo XML gerado com base no esquema proposto, foi realizada a partir da execução dos respectivos algoritmos para o mesmo conjunto de dados apresentado na seção 7.2.1.1.1. O atributo *play* foi selecionado como sendo o responsável pela identificação da categoria dos registros.

```
<xs:element name="Classification" type="ClassificationType" />
<xs:complexType name="ClassificationType">
    <xs:sequence>
        <xs:element name="Info" type="InfoType" minOccurs="1" maxOccurs="1" />
        <xs:element name="Attribute" type="AttributeType" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="AttributeType">
    <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="Results" type="ResultsType" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="ResultsType">
    <xs:sequence>
        <xs:element name="Result" type="ResultType" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="ResultType">
    <xs:sequence>
        <xs:element name="Value" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:choice>
            <xs:element name="Label" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Attribute" type="AttributeType" minOccurs="1"maxOccurs="1"/>
        </xs:choice>
    </xs:sequence>
</xs:complexType>
```

Figura 29 - Esquema XML para a categoria *Classificação*

O resultado da execução do algoritmo *ID3* no *Weka*, ilustrado pela Figura 30, mostra que, para o conjunto de dados selecionado, foi gerada uma árvore de

decisão cujo nodo raiz é representado pelo atributo *outlook*. A partir dele, dependendo do resultado, os atributos *humidity* e *windy,* foram utilizados como nodos internos. Já a execução do algoritmo *C4.5*, cuja implementação é denominada *J48*, resultou na árvore de decisão ilustrada pela Figura 31. Casualmente, este algoritmo gerou uma árvore de decisão muito similar a do *ID3*, onde também foram utilizados o atributo *outlook* como raiz, e os atributos *humidity* e *windy* como nodos internos.

```
Id3


outlook = sunny
|   humidity = '(-inf-71.2]': yes
|   humidity = '(71.2-77.4]': null
|   humidity = '(77.4-83.6]': null
|   humidity = '(83.6-89.8]': no
|   humidity = '(89.8-inf)': no
outlook = overcast: yes
outlook = rainy
|   windy = TRUE: no
|   windy = FALSE: yes
```

Figura 30 - Resultado do algoritmo *ID3* no *Weka*

7.2.1.2.2 ID3

A implementação Java do algoritmo *ID3*, fornecida juntamente com a instalação do *software Weka*, foi criada pela Universidade de Waikato e sua classe principal é a *weka.classifiers.trees.Id3*. A exemplo do que acontece com as classes principais de outros algoritmos de mineração, discutidas nas seções anteriores, a execução do *ID3* também é iniciada a partir do objeto padrão *Instances,* que representa o conjunto de dados a ser utilizado. Cada instância da classe *Id3* possui dois objetos: um representando o atributo de testes de um determinado nodo e outro representando a lista de relacionamentos, denominada de sucessores.

```
J48 pruned tree
------------------

outlook = sunny
|   humidity = '(-inf-71.2]': yes (2.0)
|   humidity = '(71.2-77.4]': no (0.0)
|   humidity = '(77.4-83.6]': no (0.0)
|   humidity = '(83.6-89.8]': no (1.0)
|   humidity = '(89.8-inf)': no (2.0)
outlook = overcast: yes (4.0)
outlook = rainy
|   windy = TRUE: no (2.0)
|   windy = FALSE: yes (3.0)
```

Figura 31 - Resultado do algoritmo *C4.5* no *Weka*

O objeto que representa o atributo, do tipo *weka.core.Attribute*, armazena o seu nome e uma lista contendo todos os seus possíveis valores, conforme definido no conjunto de dados inicial. A lista de relacionamentos é composta por um vetor de novos objetos do tipo *ID3*, sendo que a posição destes elementos possui relação direta com a lista de valores do atributo. Assim, o objeto da primeira posição deste vetor forma o arco com o atributo, sendo que a identificação do relacionamento é fornecida pela primeira posição da lista de valores. Com o uso destas estruturas, o *ID3* consegue, recursivamente, montar toda a árvore de decisão.

Para o mapeamento do resultado da execução do algoritmo *ID3* para o arquivo XML, é necessário a execução dos seguintes passos:

- Recuperar o objeto do tipo *Id3*, que representa o atributo principal da árvore;
- Gerar a *tag* principal do arquivo, denominada *Attribute*;
- Recuperar o nome do atributo armazenado no objeto *Id3* e gerar a marcação *Name* para o atributo principal;

- Gerar a marcação responsável pela definição da lista de resultados, denominada *Results*;
- Percorrer a lista de valores da classe *Attribute*;
- Para cada posição desta lista, gerar a marcação *Result*, salvando na *tag Value* o valor da iteração, recuperado a partir da execução do método *value*;
- Verificar se a lista de sucessores do atributo possui elementos ou está vazia;
- Se esta lista estiver vazia, gerar a marcação *Label* para o resultado e preencher com o valor recuperado a partir da variável interna *m_ClassAttribute* da classe *Id3*;
- Se a lista possuir elementos, criar uma nova estrutura da marcação *Attribute* e, para cada um dos itens da lista, executar todo o processo de forma recursiva até chegar a um atributo de classificação.

A Figura 32 apresenta uma parte do arquivo XML resultante deste mapeamento, utilizando, como entrada para a execução do algoritmo, o mesmo conjunto de dados abordado na seção 7.2.1.1.1. Nesta imagem, pode ser visualizada apenas a definição do atributo principal (*outlook*) e alguns dos seus resultados, sendo que o arquivo completo está disponível no Anexo III. Esta árvore é exatamente a mesma gerada pelo *software Weka*, conforme pode ser visualizado na Figura 30.

7.2.1.2.3 C4.5

A implementação em Java do algoritmo *C4.5,* fornecida juntamente com a ferramenta *Weka,* é denominada *J48,* e, a exemplo do *ID3*, também foi desenvolvida pela Universidade de Waikato. A classe principal desta implementação é a *weka.classifiers.trees.J48* e, cumprindo o contrato estabelecido pela interface padrão do *Weka*, também necessita de uma instância da classe *Instances*.

Após a sua execução, o *J48* armazena os resultados em um objeto do tipo *weka.classifiers.trees.j48.ClassifierTree.* A partir dele, de forma recursiva, é possível acessar toda a estrutura que compõe a árvore de decisão resultante. O atributo

principal está armazenado como uma variável interna desta classe, identificada por *m_root*. Cada instância do tipo *ClassifierTree,* possui três objetos que armazenam, entre outros dados, o nome do atributo de teste, a lista de componentes filhos e o conjunto original de dados, representado pelo objeto *Instances*.

```
<Attribute>
    <Name>outlook</Name>
    <Results>
        <Result>
            <Value>sunny</Value>
            <Attribute>
                <Name>humidity</Name>
                <Results>
                    <Result>
                        <Value>(-inf-71.2]</Value>
                        <Label>yes</Label>
                    </Result>
                </Results>
            </Attribute>
        </Result>
    </Results>
</Attribute>
```

Figura 32 - Resultado do algoritmo *Id3* exportado como arquivo XML

Tanto o nome quanto a lista de valores dos atributos são armazenados, na classe *ClassifierTree,* como instâncias do objeto *weka.classifiers.trees.j48.ClassifierSplitModel*. Para a recuperação do nome do atributo, é necessário a execução do método *leftSide*. Ao contrário da forma como foi desenvolvido o *ID3*, onde um atributo folha não possui sucessores, na implementação do *J48,* o componente responsável pela representação dos atributos (classe *ClassifierTree*), sempre possui uma lista de filhos. O valor de cada relacionamento é recuperado a partir do mesmo objeto da classe *ClassifierSplitModel* que contém o nome do atributo. Porém, neste caso, é utilizado o método *rightSide* juntamente com o índice do resultado que está sendo processado.

A classe *ClassifierTree* possui uma variável interna denominada *m_isLeaf*. Esta variável indica se um determinado nodo é uma folha ou não. Assim, para cada elemento da lista de filhos do atributo deve-se testar o conteúdo desta variável. Se ele for verdadeiro, o valor da classificação dos registros é extraído utilizando-se o método *dumpModel* da instância da classe *ClassifierSplitModel.* Esta classe é a mesma utilizada anteriormente para a recuperação do nome e dos valores do atributo. Porém, sendo falso este teste, significa que o elemento representa um atributo interno.

Para o mapeamento do resultado da execução do algoritmo *J48* para o arquivo XML, é necessário a execução dos seguintes passos:

- Recuperar o objeto do tipo *ClassifierTree* armazenado pela variável *m_root* da classe *J48*;
- Gerar a *tag* principal do arquivo, denominada *Attribute*;
- Recuperar o objeto *ClassifierSplitModel* armazenado na classe *ClassifierTree*;
- Executar o método *leftSide* e salvar o resultado na *tag Name* da marcação de atributo;
- Gerar a marcação responsável pela definição da lista de resultados, denominada *Results*;
- Iterar sobre a lista de filhos do classe *ClassifierTree*;
- Para cada posição desta lista, gerar a marcação *Result*, salvando na *tag Value* o valor da iteração, recuperado a partir da execução do método *rightSide*;
- Testar o valor da variável *m_isLeaf* da classe *ClassifierTree*;
- Se o valor for verdadeiro, executar o método *dumpModel* da instância da classe *ClassifierSplitModel* e mapear o resultado para a *tag Label*;
- Se o valor for falso, criar uma nova estrutura da marcação *Attribute* e, executar todo o processo de forma recursiva para a lista de objetos filhos até que se chegue a um atributo de classificação.

A Figura 33 apresenta uma parte do arquivo XML resultante deste mapeamento, utilizando, como entrada para a execução do algoritmo, o mesmo conjunto utilizado anteriormente. Nesta imagem, pode ser visualizado que o resultado é bastante semelhante ao gerado pelo algoritmo *Id3*. Este fato, no entanto, não é uma regra e sim apenas uma casualidade devido ao conjunto de dados reduzido. O arquivo completo está disponível no Anexo III. Esta árvore é exatamente a mesma gerada pelo *software Weka*, conforme pode ser visualizado na Figura 31.

```
<Attribute>
    <Name>outlook</Name>
    <Results>
        <Result>
            <Value>sunny</Value>
            <Attribute>
                <Name>humidity</Name>
                <Results>
                    <Result>
                        <Value>(-inf-71.2]</Value>
                        <Label>yes</Label>
                    </Result>
                </Results>
            </Attribute>
        </Result>
    </Results>
</Attribute>
```

Figura 33 - Resultado do algoritmo *C4.5* exportado como arquivo XML

7.2.1.2.4 Generalização

Com base nos estudos de caso apresentados nas seções anteriores, é possível concluir como deve ser o procedimento padrão para o mapeamento dos resultados da execução de algoritmos de mineração do tipo *Classificação* para o arquivo XML correspondente. Assim, independente das estruturas de dados utilizadas ou da forma como foi implementado o algoritmo, para a criação do arquivo, é necessário seguir os seguintes passos:

- Identificar qual a estrutura que armazena os dados do atributo principal, ou seja, o nodo raiz da árvore de decisão e criar no arquivo XML a marcação *Attribute;*
- Verificar a forma como o nome do atributo é salvo no objeto correspondente e mapear para a marca *Name*;
- Recuperar todos os atributos filhos do nodo atual, criando a *tag Results* no arquivo XML;
- Iterar sobre a lista de atributos filhos recuperada e, para cada objeto selecionado, criar uma estrutura correspondente à marcação *Result* no arquivo;
- Recuperar o valor do arco formado entre o atributo pai e o filho, salvando esta informação na marca *Value* que compõe o componente *Result*.
- Verificar o tipo atributo filho, se representa um nodo folha um novo atributo de teste;

- Se o atributo representar um nodo folha, deve ser criada a marcação *Label* para o componente *Result* correspondente do arquivo XML contendo o valor do atributo;
- Se o atributo representar um nodo de teste, então deve ser criada uma nova marcação do tipo *Attribute*, dentro do resultado correspondente, e todo o processo deve ser executado novamente de forma recursiva.

7.2.1.2.5 Comparação entre Formatos

Após a definição do formato para o armazenamento de resultados de algoritmos do tipo *Classificação*, é possível traçar uma comparação entre este formato e o PMML, apresentado no capítulo 5. O PMML não possui um esquema para modelos de classificação. Desta forma, uma vez que este trabalho optou por utilizar somente árvores de decisão como saída deta categoria, será utilizado a especificação da estrutura de árvores para a comparação.

O PMML utiliza a marcação *TreeModel* para definir modelos do tipo árvores. Dentre as diversas marcações de controle e parametrização do modelo, existe a *tags Node*. Este elemento define uma lista de novos itens deste tipo, representando o relacionamento entre os nodos. O nome do atributo fica armazenado através de uma estrutura do tipo *Partition*.

O formato proposto nesta seção mostra-se, novamente, mais simples, visto que devem ser apenas definidos os nodos, seus nomes e a sua lista de relacionamento. Estas listas podem estar relacionadas com novos nodos os conter o valor de classificação dos registros, representando nodos folhas.

7.2.2 Esquemas para Métodos de Visualização

Para a criação dos esquemas responsáveis por representar a estrutura das técnicas de visualização de informações, foi utilizada apenas um dos métodos estudados no capítulo 0 deste trabalho: *Árvores de Decisão*. Esta técnica foi escolhida por ser a mais comum para a visualização de resultados de algoritmos de *Classificação*.

Durante a elaboração da estrutura principal deste método, foi identificado que, a exemplo do que aconteceu para as técnicas de mineração, seria interessante que os arquivos XML possuíssem um cabeçalho de identificação. Nele poderiam estar descritas as mesmas informações apresentadas na marcação *Info,* explicada na seção 7.2.1. Além destes dados, uma outra informação, que pode ser importante estar presente no arquivo XML, é o nome da visualização que se está utilizando. A Figura 34 ilustra a definição desta marca, usando a notação *XML Schema.* Nela pode ser visualizado, além dos dados já existentes, na definição do esquema da *tag Info*, o elemento que representa o nome da técnica de visualização (*Visualization*).

```
<xs:complexType name="InfoType">
    <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1" />
        <xs:element name="Algorithm" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="Visualization" type="xs:string" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
```

Figura 34 - Definição da *tag* de informações para técnicas de visualização

7.2.2.1 Árvores de Decisão

A criação do esquema que representa a estrutura de entrada necessária para a técnica de visualização de *Árvores de Decisão* foi realizada com base nas características identificadas na seção 3.1. Foi verificado que a estrutura de dados necessária para a geração da visualização deve, obrigatoriamente, iniciar com um elemento que represente o nodo raiz da árvore. Assim, a estrutura principal do seu esquema é composta por uma marcação definida como um tipo complexo, denominada *Node*.

Seguindo as definições da classe, tem-se que cada nodo possui o nome do atributo presente no conjunto de dados inicial que está sendo representado. Desta forma, o esquema do componente *Node* recebe a definição de um elemento denominado *Name*, responsável por armazenar o nome atributo de teste representado por ele.

Um nodo pode possuir uma lista de arcos que define seu relacionamento com outros nodos. Estes, por sua vez, podem representar novos testes sobre valores de atributos ou armazenarem o valor de classificação dos registros, sendo

denominados nodo folhas. Por isso, a marcação *Node* recebe a definição de um elemento que pode conter uma lista de objetos, denominado *Children*.

Para representar os relacionamentos de um nodo com seus filhos, foi criada a marcação *Child*. Ela possui a definição do elemento que representa o arco, ou seja, um dos valores do atributo de teste, representado pela *tag Value*. Além disto, a marca *Child* possui a especificação de uma estrutura de escolha responsável por indicar se o próximo nodo do relacionamento é uma folha ou outro nodo intermediário.

No caso do próximo nodo ser uma folha, a marcação *Child* passa a possuir um elemento denominado *Result,* o qual apenas armazena o valor do atributo de classificação. Porém, se for outro atributo de testes, a marcação que deve estar presente é a *Node*, sendo que ela é do mesmo tipo da utilizada para definir o nodo raiz. Desta forma, o arquivo XML que utiliza esta estrutura como base poderá conter diversos nodos internos, seguindo o mesmo formato, até que um deles seja uma folha.

Demonstrando o esquema especificado, a Figura 35 exibe um trecho do código de definição de arquivos XML para dados de entrada da técnica *Árvores de Decisão*. O esquema completo está disponível no Anexo I.

```
<xs:element name="DecisionTree" type="DecisionTreeType" />
<xs:complexType name="DecisionTreeType">
    <xs:sequence>
        <xs:element name="Info" type="InfoType" minOccurs="1" maxOccurs="1" />
        <xs:element name="Node" type="NodeType" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="NodeType">
    <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="Children" type="ChildrenType" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="ChildrenType">
    <xs:sequence>
        <xs:element name="Child" type="ChildType" minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="ChildType">
    <xs:sequence>
        <xs:element name="Value" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:choice>
            <xs:element name="Result" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Node" type="NodeType" minOccurs="1" maxOccurs="1" />
        </xs:choice>
    </xs:sequence>
</xs:complexType>
```

Figura 35 - Esquema XML para a visualização *Árvores de Decisão*

7.2.2.2 Estudos de Caso

A visualização fornecida para a análise dos resultados do algoritmo *J48*, implementação do padrão *C4.5*, é apresentada na Figura 21. Nela podem ser verificados todos os tipos nodos existentes, ou seja, raiz, internos e folhas. Já o algoritmo *ID3,* não possui uma opção gráfica para apresentação dos resultados, sendo utilizada apenas a notação textual, conforme ilustrada pela Figura 30.

Objetivando a validação do esquema proposto, foram realizadas alguns estudos de caso. Para isto, foi utilizada a técnica de visualização fornecida pelo *Weka* para o algoritmo de *Classificação J48*. A comparação entre a visualização gerada originalmente pela aplicação *Weka* e pelo arquivo XML foi realizada a partir dos resultados da execução dos algoritmos *ID3* e *C4.5* apresentados nas seções 7.2.1.2.2 e 7.2.1.2.3. Estes resultados foram transformados, manualmente, em um arquivo XML, correspondente ao esquema de visualização explicado na seção anterior. O resultado deste mapeamento, para resultados do algoritmo *J48*, está ilustrado na Figura 36.

Como a estrutura de resultado dos algoritmos de *Classificação* é bastante similar à visualização de *Árvores de Decisão*, o mapeamento entre os arquivos foi realizado na proporção de um para um, ou seja, cada marcação de resultado possui a sua correspondente no esquema de visualização.

7.2.2.3 Visualização de Árvore do *J48*

A visualização para árvores fornecida pelo *Weka* também foi desenvolvida, a exemplo das implementações dos algoritmos, pela Universidade de Waikato. Ela é constituída por uma aplicação *Java Swing*, ou seja, é executada no próprio computador, ao invés de ser uma aplicação disponibilizada em um servidor acessado via navegador.

Para a sua execução, é necessário que os dados a serem visualizados sejam transformados em uma estrutura especifica, conforme ilustrado pela Figura 37. Nela pode ser visualizado que existe a definição de todos os nodos que fazem parte da árvore, identificados por *Nx*, onde *x* é um numérico seqüencial. Cada nodo possui a definição do seu nome de exibição através do atributo identificado por *label*. No caso

de nodos folhas, mais duas informações devem ser definidas, indicando que a forma geométrica a ser gerada deve ser um quadrado (atributos *shape* e *style*). Os arcos entre os atributos, por sua vez, devem estar descritos em estruturas do tipo *Nx→Ny*, onde *x* e *y* representam os identificadores seqüenciais que definem o nodo que faz parte do relacionamento. Além disto, os arcos devem possuir o valor do atributo de teste, definido através do atributo *label*.

Assim, para transformar o arquivo XML, apresentado na Figura 37, na entrada necessária para a visualização, é preciso proceder a recuperação do nodo raiz e a sua transformação no componente *N0*. A partir deste componente, faz-se necessária a navegação nos seus filhos para a criação das estruturas que irão representar os demais nodos. Durante este processo, são utilizados identificadores numéricos seqüenciais para a especificação de cada nodo. O valor encontrado deve ser armazenado e relacionado com o seu nodo correspondente no arquivo XML para que, posteriormente, seja realizada a referência.

Após a criação das estruturas que definem os nodos, é necessário criar as estruturas que definem os arcos. Isto não pode ser realizado junto da execução da etapa anterior, pois a ordem das definições é importante. Logo, todo o processo de recuperar o nodo raiz e percorrê-lo deve ser executado novamente, buscando, desta vez, o valor do relacionamento (marcação *Value* da *tag Result*). Para cada par de nodos que compõe o arco, os identificadores gerados na etapa anterior devem ser utilizados junto com o valor do relacionamento para a criação da estrutura necessária para a visualização.

Seguindo este processo, foi possível utilizar a visualização padrão de árvores, empregada pelo *J48* para, além da análise dos resultados do próprio algoritmo, analisar as informações geradas pela implementação *ID3*. É importante destacar que este algoritmo não possui uma técnica de visualização correspondente, e a visualização, que foi gerada a partir do uso do arquivo XML, não é possível, atualmente, no *software Weka*. A Figura 38 ilustra a visualização dos resultados do algoritmo *J48,* enquanto que a Figura 39 exibe as informação referentes a execução do *ID3*.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<DecisionTree xmlns:pos="http://www.pucrs.br/inf/pos"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.pucrs.br/inf/pos ../xsd/DecisionTree.xsd">
    <Info>
        <Name>weather</Name>
        <Description></Description>
        <Algorithm>J48</Algorithm>
        <Visualization>Decision Tree</Visualization>
    </Info>
    <Node>
        <Name>outlook</Name>
        <Children>
            <Child>
                <Value>sunny</Value>
                <Node>
                    <Name>humidity</Name>
                    <Children>
                        <Child>
                            <Value>(-inf-71.2]</Value>
                            <Result>yes</Result>
                        </Child>
                        <Child>
                            <Value>(71.2-77.4]</Value>
                            <Result>no</Result>
                        </Child>
                        <Child>
                            <Value>(77.4-83.6]</Value>
                            <Result>no</Result>
                        </Child>
                        <Child>
                            <Value>(83.6-89.8]</Value>
                            <Result>no</Result>
                        </Child>
                        <Child>
                            <Value>(89.8-inf)</Value>
                            <Result>no</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>overcast</Value>
                <Result>yes</Result>
            </Child>
            <Child>
                <Value>rainy</Value>
                <Node>
                    <Name>windy</Name>
                    <Children>
                        <Child>
                            <Value>TRUE</Value>
                            <Result>no</Result>
                        </Child>
                        <Child>
                            <Value>FALSE</Value>
                            <Result>yes</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
        </Children>
    </Node>
</DecisionTree>
```

Figura 36 - Resultado do algoritmo *J48* transformado em *Árvore de Decisão*

7.2.2.4 Generalização

Com base nos estudos de caso apresentados na seção anterior, é possível concluir como deve ser o procedimento padrão para o mapeamento dos resultados de execução de algoritmos de mineração para o arquivo XML que pode ser utilizado pela técnica *Árvores de Decisão*. Assim, independente das estruturas de dados utilizadas, ou da forma como foi implementada a visualização, para a criação do arquivo é necessário seguir os seguintes passos:

```
N0 [label="outlook" ]
N1 [label="humidity" ]
N2 [label="yes" shape=box style=filled ]
N3 [label="no" shape=box style=filled ]
N4 [label="no" shape=box style=filled ]
N5 [label="no" shape=box style=filled ]
N6 [label="no" shape=box style=filled ]
N7 [label="yes" shape=box style=filled ]
N8 [label="windy" ]
N9 [label="no" shape=box style=filled ]
N10 [label="yes" shape=box style=filled ]
N0->N1 [label="sunny"]
N1->N2 [label="(-inf-71.2]"]
N1->N3 [label="(71.2-77.4]"]
N1->N4 [label="(77.4-83.6]"]
N1->N5 [label="(83.6-89.8]"]
N1->N6 [label="(89.8-inf)"]
N0->N7 [label="overcast"]
N0->N8 [label="rainy"]
N8->N9 [label="TRUE"]
N8->N10 [label="FALSE"]
```

Figura 37 - Estrutura de dados para a visualização de árvores no Weka

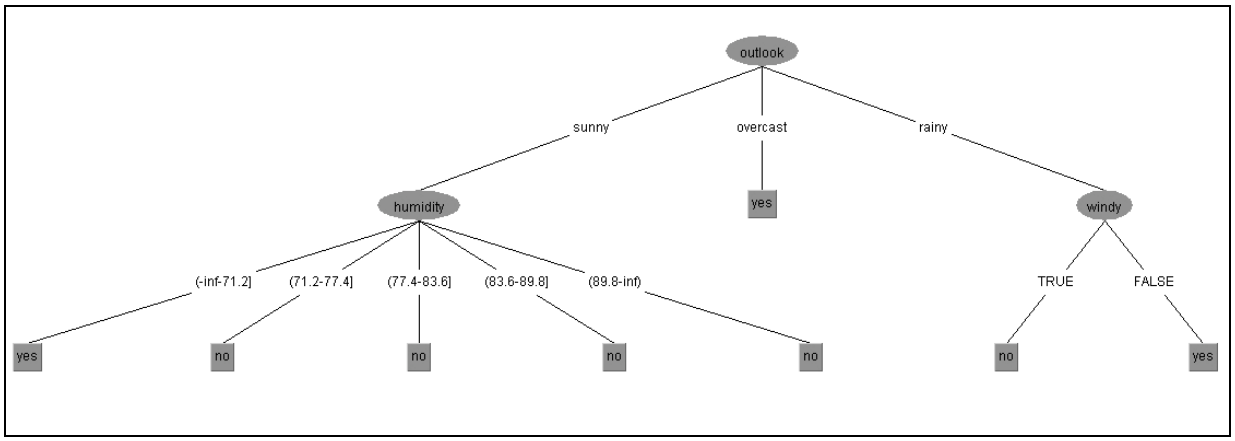


Figura 38 - Visualização do *J48* a partir do uso do arquivo XML

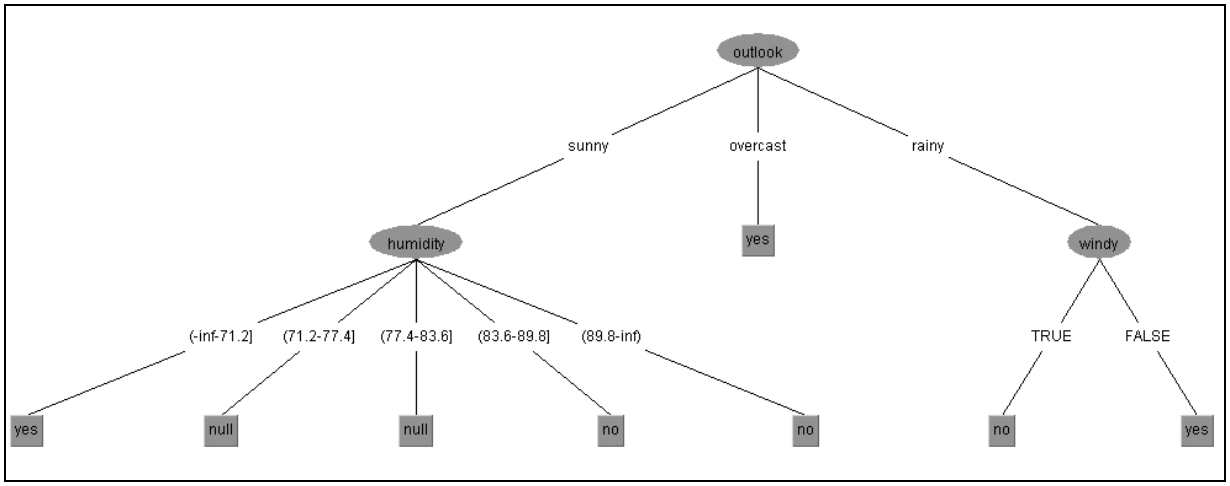


Figura 39 - Visualização do *Id3* a partir do uso do arquivo XML

- Identificar a estrutura, no arquivo XML de origem, que representa o ponto principal para entendimento dos dados, mapeando como sendo o nodo raiz da árvore utilizando a marcação *Node*, no arquivo de destino;

- Recuperar o valor que representa a identificação deste dado, como o nome do atributo, e mapear para a *tag Name* dentro da marcação *Node*

- Criar a marca *Childen* dentro da marcação *Node*;

- Procurar por elementos que possuam relacionamentos com a estrutura principal, no arquivo XML de origem, criando, para cada um, uma marcação do tipo *Child,* no arquivo de destino;

- Identificar o valor que representa o relacionamento entre os elementos, como o valor do atributo definido na *tag Node*, e mapear para a marcação *Value* da marca *Child*;

- Se estes elementos não possuírem novos relacionamentos significa que irão representar os nodos folhas da visualização, sendo que deverá ser identificado o valor que representa a classificação, salvando os dados na marcação *Result* da *tag Child*;

- Por outro lado, se os elementos possuírem outros relacionamentos, deve ser criada a marcação *Node* e o processo seguirá como exposto anteriormente.

7.3 Esquema para transformação de informações

A partir do estudo da linguagem XML e das demais estruturas que compõe esta linguagem, apresentado no capítulo 4, foi possível definir o uso desta tecnologia para a criação dos arquivos de padronização entre algoritmos de mineração e técnicas de visualização. Estes arquivos, cuja estrutura é definida através de documentos do tipo *XML Schema*, possuem estruturas próprias, sendo bem diferentes uns dos outros.

Afim de que haja a transformação automática dos dados dos arquivos XML, criados com base nos esquemas de mineração, para a estrutura de visualização, pode ser empregado o uso da técnica de *XSLT*, descrita na seção 4.3 deste documento. Neste caso, é necessário que exista um arquivo de transformação para cada par de algoritmos de mineração e técnica de análise de informações.

7.3.1 Transformação para Árvores de Decisão

7.3.1.1 Classificação

A criação do documento de transformação que manipula a estrutura de arquivos XML, constituídos no formato de *Classificação,* para gerar arquivos do tipo *Árvores de Decisão,* foi realizada com base nas estruturas dos esquemas apresentados nas seções 7.2.1.2 e 7.2.2.1, respectivamente. A partir dos esquemas, é possível verificar que as estruturas de dados são bastante similares, conforme visualizado no código da linguagem *XML Schema,* ilustrado pela Figura 29 e pela Figura 35. Isto ocorre uma vez que a técnica de visualização de *Árvores de Decisão* é, tipicamente, a mais indicada para representar os resultados de algoritmos do tipo *Classificação.*

O mapeamento entre as marcações do arquivo de origem para o de destino é realizado quase que na proporção de um para um, ou seja, para cada marcação do documento XML do tipo *Classificação* deve existir uma similar no arquivo de *Árvores de Decisão*. Para este mapeamento, é necessário a execução dos seguintes passos:

- Recuperar, no arquivo de origem, a marcação *Info,* onde podem ser encontradas as informações referentes ao nome e descrição do conjunto de

dados, algoritmo de mineração que foi utilizado e nome do atributo cujo valor representa a classe;

- Criar, no arquivo de destino, a marcação *Info* copiando os dados referentes ao nome e descrição do conjunto de dados e algoritmo de mineração que foi utilizado;
- Criar a marcação que representa o nome da visualização com um valor fixo que representa a técnica de *Árvores de Decisão* (*Decision Tree*);
- Recuperar, recursivamente, todos as *tags* do tipo *Attribute* no arquivo de origem, criado a marcação *Node* no arquivo de destino;
- Copiar o nome do atributo, armazenado na marcação *Name* dos dois arquivos;
- Recuperar, no arquivo de origem, a marcação *Results*, que contém a lista de arcos e criar, no arquivo de destino, a *tag Children*;
- Para cada resultado, identificado no arquivo de entrada pela marcação *Result*, criar no documento de saída a estrutura da *tag Child*;
- Copiar o valor que identifica o arco, armazenado na marcação *Value* dos dois arquivos;
- Se o resultado definir outro atributo no arquivo de origem, através da marcação *Attribute*, criar uma nova estrutura da *tag Node* no arquivo de destino;
- Para a identificação da classificação dos registros, recuperar o valor da marcação *Label*, presente nos resultados do arquivo de origem, e copiá-la para a *tag Result*, definida dentro da marcação *Child*, no arquivo de destino.

A Figura 40 apresenta uma parte do arquivo XSLT resultante deste mapeamento, sendo que o arquivo completo está disponível no Anexo II.

7.3.1.2 Associação

Tipicamente, os resultados dos algoritmos da classe *Associação* não podem ser visualizados como estruturas de *Árvores de Decisão*. Para este tipo de análise

podem ser criados diferentes tipos de visualizações. Assim, foram utilizados os esquemas apresentados nas seções 7.2.1.1 e 7.2.2.1, onde é possível verificar as estruturas nas quais os dados estão dispostos, conforme ilustrado pela Figura 23 e pela Figura 35.

```
<xsl:template match="pos:Classification">
    <DecisionTree xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.pucrs.br/inf/pos ../xsd/DecisionTree.xsd">
        <xsl:apply-templates select="pos:Info" />
        <xsl:apply-templates select="pos:Attribute" />
    </DecisionTree>
</xsl:template>
<xsl:template match="pos:Info">
    <Info>
        <Name>
            <xsl:value-of select="pos:Name" />
        </Name>
        <Description>
            <xsl:value-of select="pos:Description" />
        </Description>
        <Algorithm>
            <xsl:value-of select="pos:Algorithm" />
        </Algorithm>
        <Visualization>Decision Tree</Visualization>
    </Info>
</xsl:template>
<xsl:template match="pos:Attribute">
    <Node>
        <Name>
            <xsl:value-of select="pos:Name" />
        </Name>
        <xsl:if test="count(pos:Results) > 0">
            <Children>
                <xsl:for-each select="pos:Results/pos:Result">
                    <Child>
                        <Value>
                            <xsl:value-of select="pos:Value" />
                        </Value>
                        <xsl:if test="count(pos:Label) = 1">
                            <Result>
                                <xsl:value-of select="pos:Label" />
                            </Result>
                        </xsl:if>
                        <xsl:if test="count(pos:Attribute) >= 1">
                            <xsl:apply-templates select="pos:Attribute" />
                        </xsl:if>
                    </Child>
                </xsl:for-each>
            </Children>
        </xsl:if>
    </Node>
</xsl:template>
```

Figura 40 - Arquivo de transformação entre *Classificação* e *Árvores de Decisão*

O tipo de visualização definido possui um atributo principal genérico. Os atributos que compõem a cabeça das regras de associação são a identificação dos arcos. Estes, estão ligados a nodos internos identificados pelos valores dos atributos. Os nomes dos atributos que compõem o corpo das regras estão definidos como os arcos que ligam os nodos internos com as folhas. Estes últimos são

representados pelos valores dos atributos que definem o corpo. Para este mapeamento, é necessária a execução dos seguintes passos:

- Recuperar, no arquivo de origem, a marcação *Info,* onde podem ser encontradas as informações referentes ao nome e descrição do conjunto de dados e algoritmo de mineração que foi utilizado;
- Criar, no arquivo de destino, a marcação *Info* copiando os dados referentes ao nome e descrição do conjunto de dados e algoritmo de mineração que foi utilizado;
- Criar a marcação que representa o nome da visualização com um valor fixo que representa a técnica de *Árvores de Decisão* (*Decision Tree*);
- Como as regras de associação não possuem relacionamentos entre si (não sendo possível identificar qual deve representar o nodo raiz da árvore de decisão), deve-se criar a estrutura da marcação *Node* com um valor genérico (*Default Association Root*) para a *tag Name*.
- Criar a marcação *Children*;
- Recuperar, no arquivo de entrada, a lista de regras de associação, identificada pela marcação *Rules*;
- Para cada regra recuperada desta lista (*tag Rule*), criar, no arquivo de destino, a marcação *Child*;
- Recuperar a estrutura que representa a cabeça da regra de associação no arquivo de entrada (*tags Head -> Attribute*);
- Concatenar o nome de todos os atributos que representam a cabeça da regra (*tag Name*) e salvar na marcação *Value*;
- Criar o componente que representa um nodo interno no documento de saída, através de uma nova estrutura da marcação *Node*;
- Concatenar o valor de todos os atributos que representam a cabeça da rega (*tag Value*) e salvar na marcação *Name*;
- Recuperar a estrutura que representa o corpo da regra de associação no arquivo de entrada (*tags Body -> Attribute*);

- Criar, no arquivo de saída, a estrutura das marcações *Children* e *Child*;
- Concatenar o nome de todos os atributos que representam o corpo da regra (*tag Name*) e salvar na marcação *Value*;
- Concatenar o valor de todos os atributos que representam o corpo da regra (*tag Value*) e salvar na marcação *Result*.

A Figura 41 apresenta uma parte de código do arquivo XSLT resultante deste mapeamento, sendo que o arquivo completo está disponível no Anexo II.

7.3.1.3 Estudo de Caso

Objetivando a validação dos documentos de transformação propostos, foram realizados alguns testes. Assim, os resultados da execução de dois algoritmos de mineração de dados apresentados na seção 7.2.1, *Apriori* (Figura 26) e *C4.5* (Figura 33), foram utilizados como arquivos de entrada. Após a execução dos arquivos de transformação, para cada um dos arquivos de origem, foi gerado um novo arquivo disposto na estrutura de *Árvores de Decisão*.

A Figura 42 exibe um resultado bastante interessante, visto que a análise de resultados de algoritmos de *Associação* como *Árvores de Decisão* não é uma visualização usual. Por outro lado, a Figura 43, que exibe o resultado da transformação dos dados de um algoritmo de *Classificação* mapeado para a estrutura de *Árvores de Decisão* que, pela convergência das estruturas, criou um resultado já conhecido, conforme exposto pela Figura 36. As duas figuras não ilustram os arquivos completos, que estão disponíveis no Anexo III.

A Figura 44 ilustra a execução da técnica de visualização de dados *Árvore de Decisão*. Como fonte de dados, foi utilizado o documento XML criado a partir do processo de transformação para o resultado fornecido pelo algoritmo *Apriori*. Esta figura é interessante por exibir uma opção de análise de dados não disponível até então. Nela, cada uma das regras de associação está mapeada como um nodo interno da árvore, onde os nomes dos atributos que compõem a cabeça das regras são os identificadores dos arcos e os valores destes atributos estão dispostos como os valores dos nodos internos. Da mesma forma, os nomes e valores dos atributos que compõem o corpo das regras são utilizados para formar os nodos folhas. Já a

Figura 45 apresenta a execução da mesma técnica de visualização, porém, para o documento XML gerado pela transformação de resultados de algoritmos de *Classificação*. Esta visualização é exatamente a mesma gerada pela execução do algoritmo *C4.5* no *software Weka*.

```
<xsl:template match="pos:Association">
    <DecisionTree xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.pucrs.br/inf/pos ../xsd/DecisionTree.xsd">
        <xsl:apply-templates select="pos:Info" />
        <xsl:apply-templates select="pos:Rules" />
    </DecisionTree>
</xsl:template>
<xsl:template match="pos:Info">
    <Info>
        <Name>
            <xsl:value-of select="pos:Name" />
        </Name>
        <Description>
            <xsl:value-of select="pos:Description" />
        </Description>
        <Algorithm>
            <xsl:value-of select="pos:Algorithm" />
        </Algorithm>
        <Visualization>Decision Tree</Visualization>
    </Info>
</xsl:template>
<xsl:template match="pos:Rules">
    <Node>
        <Name>Default Association Root</Name>
        <Children>
            <xsl:for-each select="pos:Rule">
                <Child>
                    <xsl:apply-templates select="current()" />
                </Child>
            </xsl:for-each>
        </Children>
    </Node>
</xsl:template>
<xsl:template match="pos:Rule">
    <Value>
        <xsl:for-each select="pos:Head/pos:Attribute/pos:Name">
            |<xsl:value-of select="." />|
        </xsl:for-each>
    </Value>
    <Node>
        <xsl:element name="Name">
            <xsl:for-each select="pos:Head/pos:Attribute/pos:Value">
                |<xsl:value-of select="." />|
            </xsl:for-each>
        </xsl:element>
        <Children>
            <Child>
                <xsl:element name="Value">
                    <xsl:for-each select="pos:Body/pos:Attribute/pos:Name">
                        |<xsl:value-of select="." />|
                    </xsl:for-each>
                </xsl:element>
                <xsl:element name="Result">
                    <xsl:for-each select="pos:Body/pos:Attribute/pos:Value">
                        |<xsl:value-of select="." />|
                    </xsl:for-each>
                </xsl:element>
            </Child>
        </Children>
    </Node>
</xsl:template>
```

Figura 41 - Arquivo de transformação entre *Classificação* e *Árvores de Decisão*

```
<Node>
    <Name>Default Association Root</Name>
    <Children>
        <Child>
            <Value>|outlook|</Value>
            <Node>
                <Name>|overcast|</Name>
                <Children>
                    <Child>
                        <Value>|play|</Value>
                        <Result>|yes|</Result>
                    </Child>
                </Children>
            </Node>
        </Child>
    </Children>
</Node>
```

Figura 42 - Saída da transformação de *Associação* para *Árvores de Decisão*

```
<Node>
    <Name>outlook</Name>
    <Children>
        <Child>
            <Value>sunny</Value>
            <Node>
                <Name>humidity</Name>
                <Children>
                    <Child>
                        <Value>(-inf-71.2]</Value>
                        <Result>yes</Result>
                    </Child>
                </Children>
            </Node>
        </Child>
    </Children>
</Node>
```

Figura 43 - Saída da transformação de *Classificação* para *Árvores de Decisão*

7.4 Framework de desenvolvimento

Conforme apresentado no capítulo 5, o formato PMML também possui como objetivo a padronização dos resultados dos algoritmos de mineração, além de fornecer maneiras de documentar informações referentes à base de dados e à etapa de pré-processamento. Um dos aspectos negativos apontados para o uso deste padrão é a ausência de um *framework* de desenvolvimento que facilite algumas operações.

Assim, visando complementar o uso dos esquemas para arquivos XML, propostos neste capítulo, desenvolveu-se um *framework* para manipulação destes formatos. A idéia principal deste componente de *software* é fornecer uma maneira fácil e ágil para que os responsáveis pela programação de algoritmos de mineração e técnicas de visualização de dados possam manipular os documentos XML sem a necessidade de conhecer detalhadamente suas estruturas. Este *framework* está

dividido em duas partes, uma contendo classes utilitárias para o processo de mineração de dados e outra para o processo de geração da visualização.

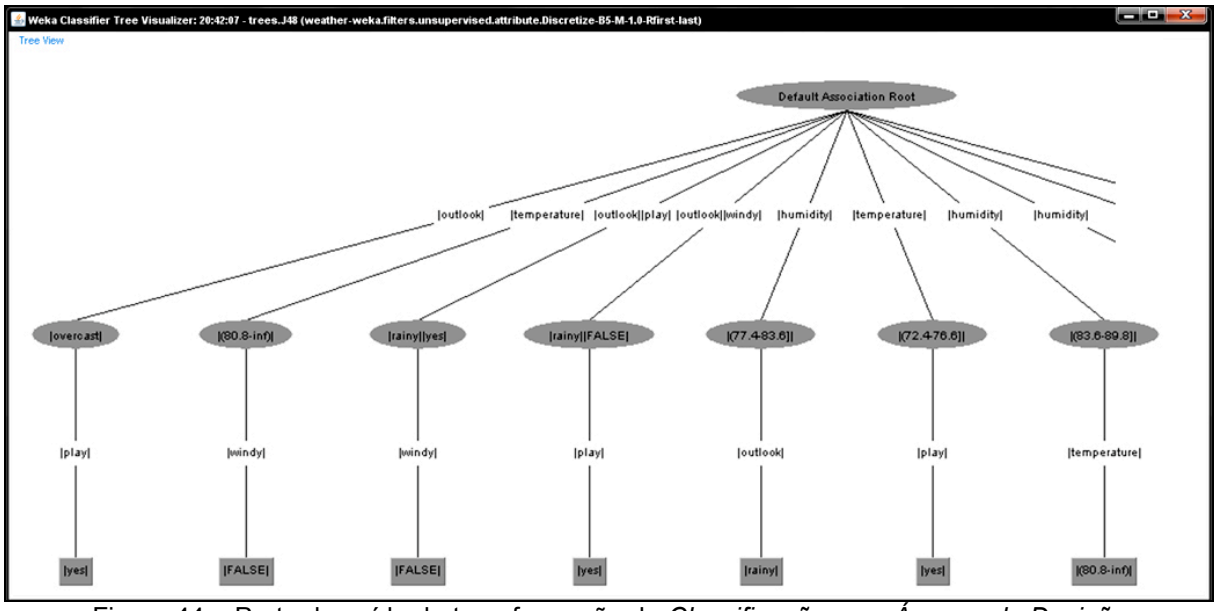


Figura 44 – Parte da saída da transformação de *Classificação* para *Árvores de Decisão*

A Figura 46 apresenta o conjunto de classes responsáveis por armazenar os dados de execução dos algoritmos. O conjunto de classes disponíveis no *framework* está relacionado às categorias de algoritmos apresentadas neste documento, ou seja, *Associação* e *Classificação*.

Conforme pode ser visualizado no diagrama, cada categoria de algoritmo possui uma classe principal, sendo que esta extende o objeto abstrato identificado como *Mining*. Ele é utilizado para que o processo de execução dos algoritmos possa ser definido de forma genérica e não possui nenhuma propriedade ou método.

Para os resultados provenientes de algoritmos de *Associação*, a classe principal para o armazenamento dos dados é denominada *Association*. Ela possui a mesma estrutura definida para os arquivos XML correspondentes a esta técnica de mineração, ou seja, define uma propriedade que representa as informações do modelo e outra que armazena uma lista de regras. A primeira é representada pela classe *AssociationInfo*, enquanto que a segunda é composta por objetos do tipo *Rule*. Cada regra, conforme já discutido anteriormente, possui duas propriedades do tipo *Attribute* que representam a cabeça e o corpo.

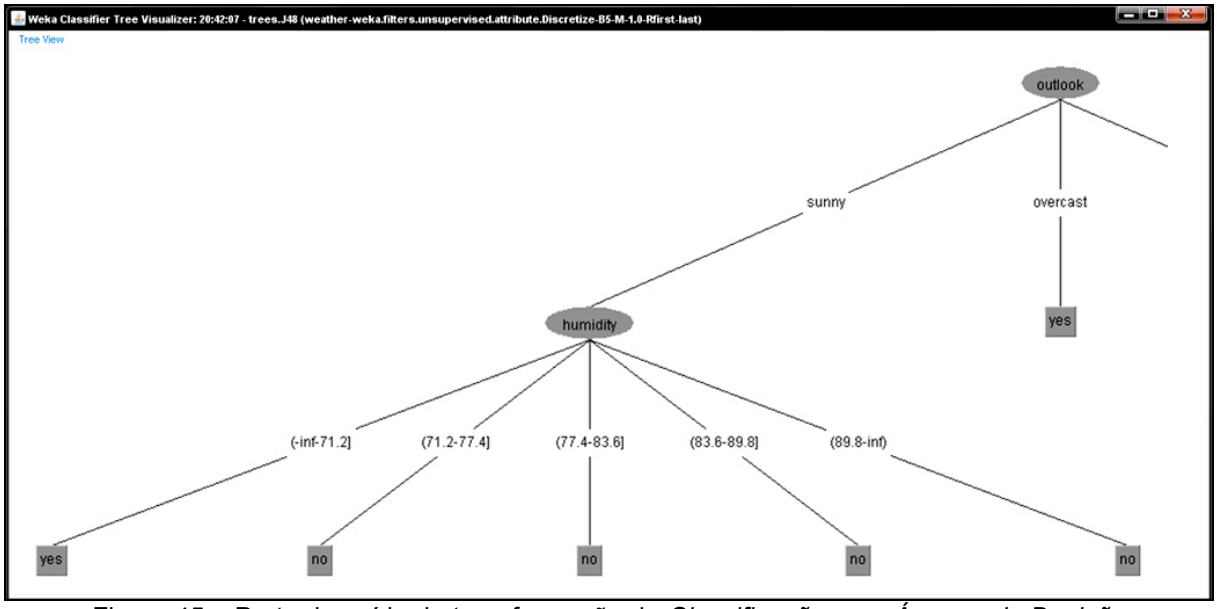


Figura 45 – Parte da saída da transformação de *Classificação* para *Árvores de Decisão*

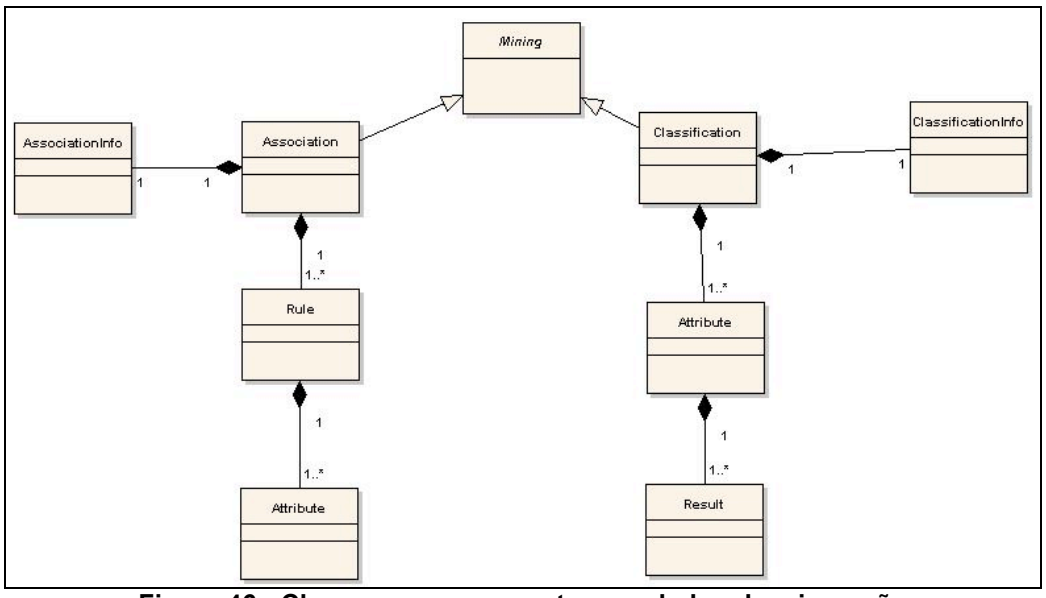


Figura 46 - Classes que representam os dados de mineração

Da mesma forma, a classe principal para os algoritmos de *Classificação* é denominada *Classification*. Na tentativa de recriar a estrutura dos arquivos XML seguindo o conceito de orientação a objetos, ela também define uma propriedade que representa as informações do modelo, denominada *ClassificationInfo*. Além disto, possui uma segunda propriedade responsável por mapear o atributo principal do modelo de classificação, representado por um objeto do tipo *Attribute*. Este define uma uma lista com os possíveis resultados do atributo, composta por instâncias da classe *Result*.

A Figura 47 ilustra a estrutura de classes responsável pela execução dos algoritmos de mineração de dados. A classe *AbstractMining* define todos os métodos principais para o uso de qualquer categoria de algoritmo, como a função de iniciar a execução do algoritmo e recuperar seus resultados. Por outro lado, as classes abstratas específicas *AbstractAssociation* e *AbstractClassification,* definem métodos que são responsáveis pela extração dos dados seguindo a categoria do algoritmo. Assim, a primeira define um método para recuperar a lista de regras de associação, enquanto que a segunda busca o atributo principal da classificação. Estas classes são as responsáveis por criar o arquivo XML contendo o resultado obtido para o esquema correspondente.

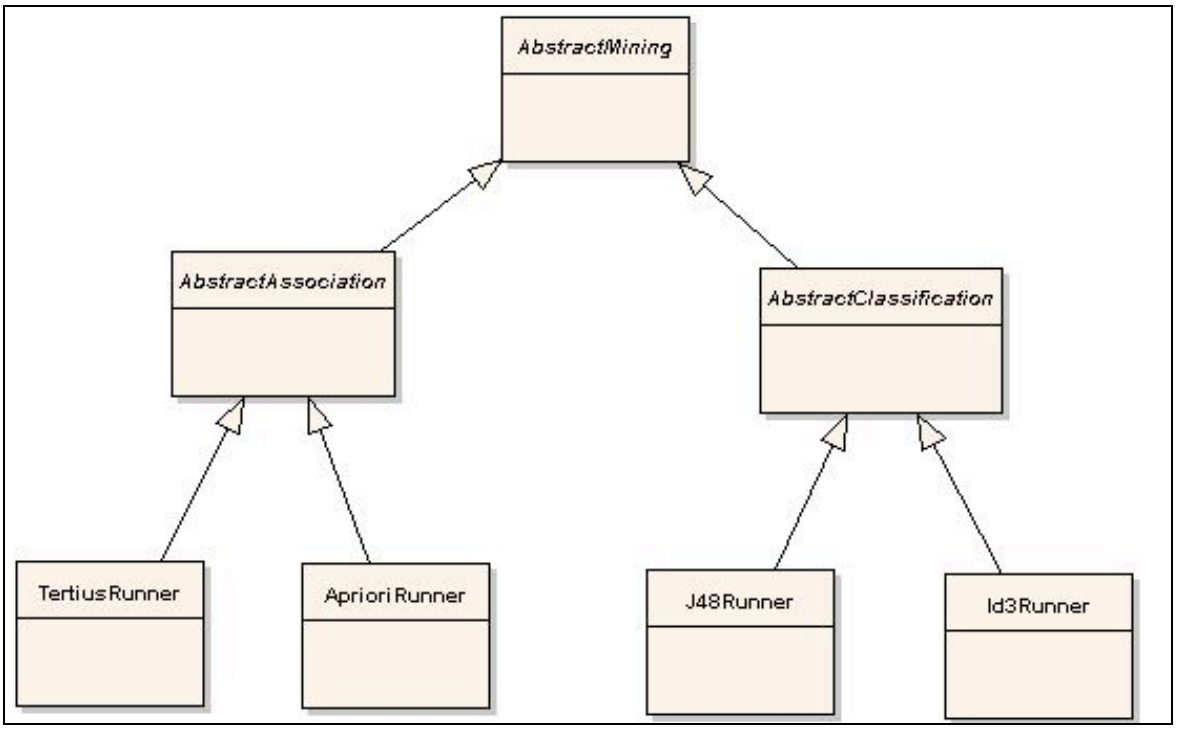


Figura 47 - Modelo de classes para algoritmos de mineração

Para adicionar novos algoritmos utilizando este *framework*, os autores devem criar novas classes filhas de uma das duas classes abstratas apresentadas anteriormente, de acordo com seu tipo. Na Figura 47, são apresentadas quatro exemplos de objetos que implementam as classes abstratas: *TertiusRunner, AprioriRunner, J48Runner* e *Id3Runner*. Elas são responsáveis pela execução, respectivamente, dos algoritmos *Tertius, Apriori, J48* e *ID3*, detalhados na seção 7.2.1.

Seguindo a mesma lógica das classes de mineração de dados, a Figura 48 apresenta o conjunto de classes responsáveis por armazenar os dados necessários para a geração de visualizações. O conjunto de classes disponíveis no *framework* também está relacionado à categoria *Árvores de Decisão* abordada neste documento.

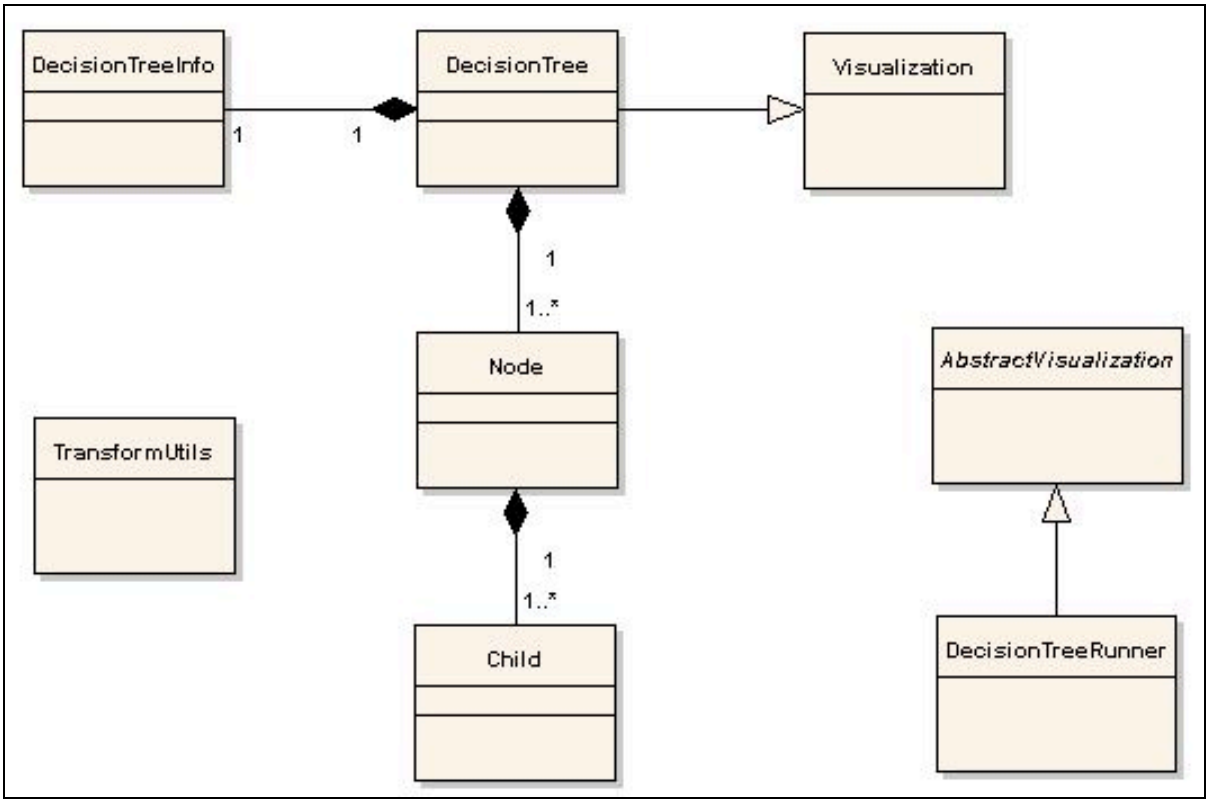


Figura 48 - Modelo de classes para visualização e transformação

A classe principal para o armazenamento de dados de visualização é a *DecisionTree*. Sua estrutura é bastante similar à classe *Classification* discutida anteriormente, ou seja, também possui uma propriedade responsável por armazenar as informações de identificação da visualização e outra que define uma lista de nodos. A primeira é representada pela classe *DecisionTreeInfo*, enquanto que a segunda é composta por objetos do tipo *Node*. Além disto, a classe *Node* define uma outra lista, responsável por armazenar todos os arcos de relacionamentos do nodo com outros, utilizando instâncias da classe *Child*.

Para a geração da visualização, deve ser utilizada como base a classe abstrata *AbstractVisualization*. Ela define os métodos principais que devem ser

utilizados para a leitura do arquivo XML, o qual contém os dados seguindo a estrutura definida para a técnica correspondente e posterior execução da visualização. Os autores das técnicas de visualização, então, devem apenas implementar os métodos abstratos desta classe e, a partir do objeto contendo os dados, gerar a sua visualização.

Conforme também é ilustrado pela Figura 48, é apresentado juntamente com o *framework* a classe *DecisionTreeRunner*. Ela é responsável pela execução da visualização de árvores fornecida juntamente com o *software Weka* [Wit05], conforme detalhado na seção 7.3. Além disto, o diagrama apresenta a classe *TransformUtils*. Ela é responsável por realizar a conversão entre os formatos de documentos XML, com base nos arquivos XSLT especificados anteriormente.

Para que novas classes de algoritmos de mineração sejam adicionadas ao *framework*, é necessário seguir os seguintes passos:

- Criar uma nova classe abstrata que implemente a classe *AbstractMining;*
- Implementar o método que é responsável por criar o arquivo XML para a categoria de algoritmo a ser adicionada;
- Definir novos métodos abstratos para que as classes que irão executar realmente os algoritmos possam retornar os dados necessários para a criação dos arquivos;
- Criar uma nova classe principal para o armazenamento dos dados que extenda a classe *Mining*.

Para que novos algoritmos de mineração de dados possam ser utilizados, é necessário seguir os seguintes passos:

- Criar uma nova classe que implemente a abstrata correspondente à categoria de algoritmos desejada;
- Implementar o método abstrato que responsável pelo retorno da execução do algoritmo;
- Implementar o processo de execução do algoritmo.

A adição de novas técnias de visualização de dados segue a mesma lógica descrita para algoritmos de mineração. A diferença, naturalmente, é referente à classe que devem ser implementada. Neste caso, deve ser utilizada a classe *AbstractVisualization*.

7.5 Considerações

Este capítulo discorreu sobre a solução proposta para a resolução do problema descrito anteriormente. O primeiro passo para a criação desta solução foi a definição do formato no qual está baseada a idéia de padronização entre as etapas do processo de descoberta de conhecimento. Seguindo a mesma proposta do formato PMML, estudada no capítulo 5.1, foi definido que a melhor estrutura para atender a padronização seria o uso de *XML Schema*.

Inicialmente, trabalhou-se com a padronização do resultado das técnicas de mineração de dados. Para isto, foram utilizados os estudos referentes aos algoritmos das classes *Associação* e *Classificação*. Cada uma destas categorias teve a identificação do formato padrão de como o seu resultado é gerado e, a partir disto, os esquemas dos documentos correspondentes foram criados. A validação desta proposta foi realizada a partir de testes utilizando os algoritmos fornecidos pelo *software Weka*. O resultado destes testes foi importante no sentido de fornecer uma visão de um processo padrão para o mapeamento dos resultados para os arquivos XML.

Seguindo a mesma idéia, a geração das visualizações também tem a entrada de dados padronizada com o uso de arquivos XML. Assim, a partir do estudo da técnica de *Árvores de Decisão*, foi identificada a forma principal como os dados devem ser passados para o componente de visualização, sendo que o resultado disto foi a criação de um novo esquema, em *XML Schema*. Para a validação, novos testes foram realizados a partir da conversão dos arquivos que representam os resultados dos algoritmos de mineração para a estrutura de entrada da técnica de visualização. Este novo documento XML serviu de base para a visualização de árvores fornecida com a ferramenta *Weka*. Novamente, foi feita a definição de um processo padrão para o mapeamento.

Além disto, para que a transformação entre os formatos acontecesse de forma mais independente possível, foram criados documentos do tipo *XSLT*, responsáveis por ler um arquivo XML e gerar um novo documento. A conversão entre os formatos de *Classificação* e *Árvores de Decisão* aconteceu naturalmente, visto que as duas técnicas são complementares. Porém, o mapeamento entre *Associação* e *Árvores de Decisão* necessitou de uma estruturação própria, gerando uma árvore na qual os arcos representam os atributos que compõem a cabeça e o corpo das regras de associação, enquanto que os nodos representam os valores destes atributos.

Por fim, para facilitar o uso destes formatos, foi criado um *framework* de desenvolvimento. A partir do uso das classes definidas, novos algoritmos de mineração de dados e técnicas de visualização podem passar a utilizar os arquivos XML. Todavia, para isso, é necessário que sigam o contrato estabelecido através das interfaces padrões fornecidas. Este *framework* foi criado contendo as classes principais para manipulação dos algoritmos e visualizações estudados.

8 Conclusão e Trabalhos Futuros

Conforme discutido ao longo deste trabalho, com o aumento do uso de ferramentas computacionais tem se observado uma grande geração de dados que representam o estado e comportamento dos processos diários das organizações. Este fato está contribuindo para que o foco das empresas se desvie do mero acompanhamento e controle para ações de medição, análise e monitoração.

Neste sentido, o uso de técnicas de descoberta de conhecimento sobre estas bases de dados se mostrou uma prática bastante promissora, vez que as empresas começam a aprender sobre a forma de execução dos seus processos e, com isto, atuar diretamente em pontos problemáticos. Porém, ferramentas deste tipo podem necessitar de muitos recursos computacionais e, dependendo da operação que está sendo executada, impactar em problemas para os usuários. Desta forma, a opção que traz melhores desempenhos passa por um ambiente distribuído. Este fato leva, então, ao problema que este trabalho se propôs a resolver: como realizar a troca de informações entre as etapas do processo de KDD, focando nas saídas do processo de mineração de dados e entradas para a geração da visualização?

O estudo do principal trabalho relacionado, apresentado no capítulo 5, permitiu uma visão sobre a melhor forma de criar esta padronização, ou seja, através do uso de documentos XML. Esta tecnologia, apresentada no capítulo 4, é a mais indicada para troca de informações entre aplicações.

A proposta dos documentos de esquema, utilizando o padrão *XML Schema*, demonstrou, por meio de estudos de caso envolvendo resultados de algoritmos de mineração fornecidos gratuitamente pela ferramenta *Weka* [Wit05], e por técnicas de visualização deste mesmo aplicativo, que os documentos XML gerados proporcionam uma forma de integração entre as etapas citadas. Além da simplicidade da proposta, o fato dos esquemas terem sido criados com base nos padrões das técnicas de mineração e visualização facilita a compreensão para a sua utilização.

A transformação entre os formatos, automatizada pelo uso da técnica de XSLT, propicia a manipulação automatizada entre os arquivos XML, beneficiando

desenvolvedores das aplicações. Outro ponto importante a ser destacado é que, com o uso dos documentos propostos, técnicas que historicamente não eram compatíveis, como *Associação* e *Árvores de Decisão*, puderam ser utilizadas para a visualização de mesmos conjuntos de dados.

Esta proposta permite que, futuramente, novas classes de algoritmos de mineração sejam adicionadas para uso. Para isto, basta que seja criado o esquema de definição da estrutura dos dados e o documento de transformação para as técnicas de visualização existentes. Além disto, novos algoritmos das classes utilizadas poderão ser incorporados à solução, agregando em diversidade de opções para a manipulação dos conjuntos de dados.

Outra proposta seria a melhoria da visualização de resultados de algoritmos de *Associação* utilizando a técnias de *Árvores de Decisão*. O arquivo XSLT correspondente poderia identificar regras que possuem cabeças com mesmos atributos e criar um único nodo interno. Os diferentes resultados, ou valores dos atributos, poderiam direcionar para nodos folhas contendo o corpo da regra ou nodos internos seguindo a mesma idéia.

Da mesma forma, outra possibilidade de trabalho futuro é a adição de novas técnicas de visualização de informações, tanto de novos tipos como *Redes Bayesianas*, como outras versões da categoria estudada. Novamente, para que isto possa ser possível, é necessária a criação de um novo esquema referente à estrutura de dados da visualização. Em complemento, devem ser criados tantos documentos de transformação quantas opções de algoritmos de mineração disponíveis.

Em complemento a esta tese de dissertação, foi publicado mais um trabalho cientifico: um artigo na III Escola Regional de Banco de Dados (ERBD 2007). Para dar continuidade ao trabalho, novos artigos estão sendo previstos para futuras submissões.

Referências

[Aal02] AALST, Wil M. P. van der; HEE, Kees van. "***Workflow Management: models, methods and systems***". Cambridge: MIT Press, 2002. 368p.

[Aal03] AALST, Wil M. P. van der; HOFSTEDE, Arthur H. M. ter; WESKE, Mathias. "Business Process Management: A survey". In: International Conference on Business Process Management (BPM2003), 2003, Berlin – Germany. ***Proceedings...*** Berlin: Springer-Verlag, 2003, p.1-12.

[Aal05] AALST, Wil M. P. van der. "Business Alignment: Using Process Mining as a Tool for Delta Analysis and Conformance Testing". ***Requirements Engineering Journal***, London, v. 10, n. 3, p. 198-211, nov. 2005.

[Agr94] AGRAWAL, Rakesh; SRIKANT, Ramakrishnam. "Fast algorithms for mining association rules in large databases". In: International Conference on Very Large Data Bases (VLDB), 20, 1994, Santiago – Chile. ***Proceedings ...*** Hove: Morgan Kaufmann, 1994, p.478-499.

[Bec06] BECKER, Karin; RUIZ, Duncan Dubugras; CUNHA, Virgínia Silva da; NOVELLO, Taisa Carla; SOUZA, Franco Vieira. "SPDW: a Software Development Process Performance Data Warehousing Environment". In: Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop (SEW-06), 30, 2006, Columbia – USA. ***Proceedings ...***. Los Alamitos: IEEE Press, 2006, p.107-118.

[Cas02] CASATI, Fabio. "***Intelligent Process Data Warehouse for HPPM 5.0***". Palo Alto: HP Company, 2002. 40p.

[Cas05] CASATI, Fabio. "Industry Trends in Business Process Management: Getting Ready for Prime Time". In: 16th International Workshop on Database and Expert Systems Applications (DEXA 2005), 16, 2005, Copenhagen - Denmark. ***Proceedings ...*** Los Alamitos: IEEE Press, 2005, p.903-907.

[Cha02] CHANG, Siu; JAECKEL, Clara. "***Oracle Workflow Guide***". San Francisco: Oracle Corp., 2002, vol. 1, 1896p.

[Dat07] DATA MINING GROUP. "***Home***". Capturado em: http://www.dmg.org, Setembro 2007.

[Dat07a] DATA MINING GROUP. "***PMML Version 3.0***". Capturado em: http://www.dmg.org/pmml-v3-0.html, Abril 2007.

[Dat07b] DATA MINING GROUP. "***PMML3.0 – General Structure of a PMML Document***". Capturado em: http://www.dmg.org/v3-0/GeneralStructure.html, Setembro 2007.

[Enh07] ENHYDRA. "***Shark: Open Source Java XPDL Workflow***". Capturado em: http://www.enhydra.org/workflow/shark, Junho 2007.

[Fay96] FAYYAD, Usama; SHAPIRO, Gregory Piatetsky; SMYTH, Padhraic. "Knowledge Discovery and Data Mining: Towards a unifying framework". In: 2nd International Conference on Knowledge Discovery and Data Mining, 2, 1996, Oregon – Portland. ***Proceedings …*** Menlo Park: AAAI Press, 1996, p.82-88.

[Fla01] FLACH, Peter A.; LACHICHE, Nicolas. "Confirmation-Guided Discovery Of First-Order Rules with Tertius". ***Machine Learning***, New York, v. 42, n. 1/2, p. 61-95, jan. 2001.

[Fla07] FLACH, Peter A.; LACHICHE, Nicolas. "***Tertius: a system for rule discovery in first-order logic***". Capturado em: http://www.cs.bris.ac.uk/Research/MachineLearning/Tertius, Setembro 2007.

[Gar05] GARCIA, Rafael Saraiva. "***Descoberta de Conhecimento em Processos de Workflow***". 2005. 144p. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação), Faculdade de Informática, PUCRS, 2005.

[Gar05a] GARCIA, Rafael Saraiva; RUIZ, Duncan Dubugras. "Pré-Processamento de Dados para Descoberta de Conhecimento em Processos de Workflow Modelados Sobre Plataforma Oracle". In: 1ª

Escola Regional de Banco de Dados – SBC, 1, 2005, Porto Alegre – Brasil. ***Anais ...*** Porto Alegre: UFRGS, 2005, p. 25-30.

[Gar05b] GARCIA, Rafael Saraiva; Ruiz, Duncan Dubugras. “Uma Abordagem Para a Descoberta de Conhecimento em Processos de Workflow em Oracle”. In: II Simpósio Brasileiro de Sistemas de Informação, 2, 2005. ***Anais ...*** Florianópolis: INE-UFSC, 2005, p.1-8.

[Gar06] GARCIA, Rafael Saraiva. “***Padrões de Troca de Resultados de Mineração Entre Algoritmos e Ferramentas de Visualização***”. 2006. 90p. Trabalho Individual I (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2006.

[Gar06a] GARCIA, Rafael Saraiva; RUIZ, Duncan Dubugras. “Athena Model – Beta: Um Modelo Multidimensional para Armazenar Logs de Execuções de Processos de Workflow”. In: 3ª Escola Regional de Banco de Dados – SBC, 3, 2007, Caxias do Sul – Brasil. ***Anais ...*** Caxias do Sul: UCS, 2007, p.164-173.

[Gol01] GOLDGARB, Charles F.; PRESCOD, Paul. “***The XML Handbook***”. Upper Saddle River: Prentice Hall, 2001, 3rd Ed, 988p.

[Gol04] GOLFARELLI, Mateo; RIZZI, Stefano; CELLA, Iuris. “Beyond data warehousing: what's next in business intelligence?”. In: 7th ACM international workshop on Data warehousing and OLAP, 7, 2004, Washington – USA. ***Proceedings …*** New York: ACM, 2004, p.1-6.

[Gro02] GROSSMAN, Robert; HORNICK, Mark; MEYER, Gregor. “Data Mining Standards Initiatives”. ***Communications of the ACM***, New York, v. 45, n. 8, p. 59-61, ago. 2002.

[Har99] HAROLD, Elliote Rusty. “***XML Bible***”. Chicago, IL: IDG Book Worldwide, 1999. 1015p.

[Han01] HAN, Jiawei; KAMBER, Micheline. “***Data Mining: Concepts and Techniques***”. San Francisco: Morgan Kaufmann Publishers, 2001. 550p.

[Hec97] HECKERMAN, David. “Bayesian Networks for Data Mining”. ***Data Mining and Knowledge Discovery Journal***, v. 1, n. 1, p. 79-119, mar. 1997.

[Hol95] HOLLINGSWORTH, David. “***The Workflow Reference Model***”. Document Number TC-00-1003. Hampshire: WfMC Specification, 1995, vol. 1.1, 55p.

[Qui93] QUINLAN, J. Ross. “***C4.5: Programs for Machine Learning***”. San Francisco, CA: Morgan Kaufmann Publishers, 1993, 302p.

[Rui05] RUIZ, Duncan Dubugras; BECKER, Karin; Novello, TAISA Carla; CUNHA, Virgínia Silva da. “A Data Warehousing Environment to Monitor Metrics in Software Development Process”. In: 1st International Workshop on Business Process Monitoring & Performance Management, 1, 2005, Copenhagen – Denmark. ***Proceedings ...*** Los Alamitos: IEEE Computer Society Press, 2006, p.936-940.

[Sch01] SCHEFFER, Tobias. “Finding association rules that trade support optimally against confidence”. In: 5th European Conference on Principles of Data Mining and Knowledge Discovery, 5, 2001, Freiburg – Germany. ***Proceedings …*** London: Springer-Verlag, 2001, p.424-435.

[Tan06] TAN, Pang-Ning; STEINBACH, Michael; KUMAR, Vipin. “***Introduction to data mining***”. Boston: Addison-Wesley, 2006, 796p.

[W3C07] WORLD WIDE WEB CONSORTIUM. “***XML Schema Part 0: Primer Second Edition***”. October, 2004. Capturado em: http://www.w3.org/TR/xmlschema-0, Maio 2007.

[W3C07a] WORLD WIDE WEB CONSORTIUM. “***Extensible Markup Language (XML) 1.0***”. 4th Edition. August, 2006. Capturado em: http://www.w3.org/TR/REC-xml, Maio 2007.

[W3C07b] WORLD WIDE WEB CONSORTIUM. “***XSL Transformations (XSLT) Version 2.0***”. January, 2007. Capturado em: http://www.w3.org/TR/xslt20, Abril 2007.

[W3S07] W3SCHOOLS. "***XML Schema Tutorial***". Capturado em: http://www.w3schools.com/schema/schema_intro.asp, Abril 2007.

[W3S07a] W3SCHOOLS. "***XSLT Tutorial***". Capturado em: http://www.w3schools.com/xsl, Abril 2007.

[WfC99] WORKFLOW MANAGEMENT COALITION. "***Terminology & Glossary***". Document Number WFMC-TC-1011. Hampshire: WfMC Specification, 1999, vol. 3, 65p.

[Wit05] WITTEN, Ian H.; FRANK, Eibe. "***Data Mining: Practical Machine Learning Tools and Techniques***". San Francisco, CA: Morgan Kaufmann Publishers, 2005, 2nd ed., 525p.

Apêndice I – Esquemas

Esquema para resultados de algoritmos de *Associação*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.pucrs.br/inf/pos" xmlns="http://www.pucrs.br/inf/pos"
        elementFormDefault="qualified">
    <xs:element name="Association" type="AssociationType" />
    <xs:complexType name="AssociationType">
        <xs:sequence>
            <xs:element name="Info" type="InfoType" minOccurs="1" maxOccurs="1" />
            <xs:element name="Rules" type="RulesType" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="InfoType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1" />
            <xs:element name="Algorithm" type="xs:string" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="RulesType">
        <xs:sequence>
            <xs:element name="Rule" type="RuleType" minOccurs="1" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="RuleType">
        <xs:sequence>
            <xs:element name="Head" type="AttributeListType" minOccurs="1" maxOccurs="1" />
            <xs:element name="Body" type="AttributeListType" minOccurs="1" maxOccurs="1" />
            <xs:element name="Support" type="xs:double" minOccurs="1" maxOccurs="1" />
            <xs:element name="Confidence" type="xs:double" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="AttributeListType">
        <xs:sequence>
            <xs:element name="Attribute" type="AttributeType" minOccurs="1"
                maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="AttributeType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Value" type="xs:string" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Esquema para resultados de algoritmos de *Classificação*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.pucrs.br/inf/pos" xmlns="http://www.pucrs.br/inf/pos"
        elementFormDefault="qualified">
    <xs:element name="Classification" type="ClassificationType" />
    <xs:complexType name="ClassificationType">
        <xs:sequence>
            <xs:element name="Info" type="InfoType" minOccurs="1" maxOccurs="1" />
            <xs:element name="Attribute" type="AttributeType" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="InfoType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1" />
            <xs:element name="Algorithm" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="ClassLabelAttribute" type="xs:string" minOccurs="1"
                maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="AttributeType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Results" type="ResultsType" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ResultsType">
        <xs:sequence>
            <xs:element name="Result" type="ResultType" minOccurs="1" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ResultType">
        <xs:sequence>
            <xs:element name="Value" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:choice>
                <xs:element name="Label" type="xs:string" minOccurs="1" maxOccurs="1" />
                <xs:element name="Attribute" type="AttributeType" minOccurs="1"
                    maxOccurs="1" />
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Esquema para entrada para visualizações de *Árvore de Decisão*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.pucrs.br/inf/pos"
        xmlns="http://www.pucrs.br/inf/pos"
        elementFormDefault="qualified">
    <xs:element name="DecisionTree" type="DecisionTreeType" />
    <xs:complexType name="DecisionTreeType">
        <xs:sequence>
            <xs:element name="Info" type="InfoType" minOccurs="1" maxOccurs="1" />
            <xs:element name="Node" type="NodeType" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="InfoType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1" />
            <xs:element name="Algorithm" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Visualization" type="xs:string" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="NodeType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:element name="Children" type="ChildrenType" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ChildrenType">
        <xs:sequence>
            <xs:element name="Child" type="ChildType" minOccurs="1" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ChildType">
        <xs:sequence>
            <xs:element name="Value" type="xs:string" minOccurs="1" maxOccurs="1" />
            <xs:choice>
                <xs:element name="Result" type="xs:string" minOccurs="1" maxOccurs="1" />
                <xs:element name="Node" type="NodeType" minOccurs="1" maxOccurs="1" />
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Apêndice II – Transformações

Transformação entre Associação e Árvore de Decisão:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:pos="http://www.pucrs.br/inf/pos">
    <xsl:output method="xml" version="1.0" encoding="iso-8859-1" indent="yes" />
    <xsl:template match="pos:Association">
        <DecisionTree xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.pucrs.br/inf/pos ../xsd/DecisionTree.xsd">
            <xsl:apply-templates select="pos:Info" />
            <xsl:apply-templates select="pos:Rules" />
        </DecisionTree>
    </xsl:template>
    <xsl:template match="pos:Info">
        <Info>
            <Name><xsl:value-of select="pos:Name" /></Name>
            <Description><xsl:value-of select="pos:Description" /></Description>
            <Algorithm><xsl:value-of select="pos:Algorithm" /></Algorithm>
            <Visualization>Decision Tree</Visualization>
        </Info>
    </xsl:template>
    <xsl:template match="pos:Rules">
        <Node>
            <Name>Default Association Root</Name>
            <Children>
                <xsl:for-each select="pos:Rule">
                    <Child><xsl:apply-templates select="current()" /></Child>
                </xsl:for-each>
            </Children>
        </Node>
    </xsl:template>
    <xsl:template match="pos:Rule">
        <Value>
            <xsl:for-each select="pos:Head/pos:Attribute/pos:Name">
                |<xsl:value-of select="." />|
            </xsl:for-each>
        </Value>
        <Node>
            <xsl:element name="Name">
                <xsl:for-each select="pos:Head/pos:Attribute/pos:Value">
                    |<xsl:value-of select="." />|
                </xsl:for-each>
            </xsl:element>
            <Children>
                <Child>
                    <xsl:element name="Value">
                        <xsl:for-each select="pos:Body/pos:Attribute/pos:Name">
                            |<xsl:value-of select="." />|
                        </xsl:for-each>
                    </xsl:element>
                    <xsl:element name="Result">
                        <xsl:for-each select="pos:Body/pos:Attribute/pos:Value">
                            |<xsl:value-of select="." />|
                        </xsl:for-each>
                    </xsl:element>
                </Child>
            </Children>
        </Node>
    </xsl:template>
</xsl:stylesheet>
```

Transformação entre Classificação e Árvore de Decisão:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:pos="http://www.pucrs.br/inf/pos">
    <xsl:output method="xml" version="1.0" encoding="iso-8859-1" indent="yes" />
    <xsl:template match="pos:Classification">
        <DecisionTree xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.pucrs.br/inf/pos ../xsd/DecisionTree.xsd">
            <xsl:apply-templates select="pos:Info" />
            <xsl:apply-templates select="pos:Attribute" />
        </DecisionTree>
    </xsl:template>
    <xsl:template match="pos:Info">
        <Info>
            <Name>
                <xsl:value-of select="pos:Name" />
            </Name>
            <Description>
                <xsl:value-of select="pos:Description" />
            </Description>
            <Algorithm>
                <xsl:value-of select="pos:Algorithm" />
            </Algorithm>
            <Visualization>Decision Tree</Visualization>
        </Info>
    </xsl:template>
    <xsl:template match="pos:Attribute">
        <Node>
            <Name>
                <xsl:value-of select="pos:Name" />
            </Name>
            <xsl:if test="count(pos:Results) > 0">
                <Children>
                    <xsl:for-each select="pos:Results/pos:Result">
                        <Child>
                            <Value>
                                <xsl:value-of select="pos:Value" />
                            </Value>
                            <xsl:if test="count(pos:Label) = 1">
                                <Result>
                                    <xsl:value-of select="pos:Label" />
                                </Result>
                            </xsl:if>
                            <xsl:if test="count(pos:Attribute) >= 1">
                                <xsl:apply-templates select="pos:Attribute" />
                            </xsl:if>
                        </Child>
                    </xsl:for-each>
                </Children>
            </xsl:if>
        </Node>
    </xsl:template>
</xsl:stylesheet>
```

Apêndice III – Estudos de Casos

Resultado da execução do algoritmo *Apriori*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Association xmlns="http://www.pucrs.br/inf/pos"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.pucrs.br/inf/pos ../xsd/Association.xsd">
    <Info>
        <Name>weather</Name>
        <Algorithm>Apriori</Algorithm>
    </Info>
    <Rules>
        <Rule>
            <Head>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>overcast</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>play</Name>
                    <Value>yes</Value>
                </Attribute>
            </Body>
            <Support>4.0</Support>
            <Confidence>1.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(80.8-inf)</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>windy</Name>
                    <Value>FALSE</Value>
                </Attribute>
            </Body>
            <Support>3.0</Support>
            <Confidence>1.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>rainy</Value>
                </Attribute>
                <Attribute>
                    <Name>play</Name>
                    <Value>yes</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>windy</Name>
                    <Value>FALSE</Value>
                </Attribute>
            </Body>
            <Support>3.0</Support>
            <Confidence>1.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>rainy</Value>
                </Attribute>
```

```
            <Attribute>
                <Name>windy</Name>
                <Value>FALSE</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>play</Name>
                <Value>yes</Value>
            </Attribute>
        </Body>
        <Support>3.0</Support>
        <Confidence>1.0</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>humidity</Name>
                <Value>(77.4-83.6]</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>outlook</Name>
                <Value>rainy</Value>
            </Attribute>
        </Body>
        <Support>2.0</Support>
        <Confidence>1.0</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>temperature</Name>
                <Value>(72.4-76.6]</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>play</Name>
                <Value>yes</Value>
            </Attribute>
        </Body>
        <Support>2.0</Support>
        <Confidence>1.0</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>humidity</Name>
                <Value>(83.6-89.8]</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>temperature</Name>
                <Value>(80.8-inf)</Value>
            </Attribute>
        </Body>
        <Support>2.0</Support>
        <Confidence>1.0</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>humidity</Name>
                <Value>(77.4-83.6]</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>windy</Name>
                <Value>FALSE</Value>
```

```
                </Attribute>
            </Body>
            <Support>2.0</Support>
            <Confidence>1.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(77.4-83.6]</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>play</Name>
                    <Value>yes</Value>
                </Attribute>
            </Body>
            <Support>2.0</Support>
            <Confidence>1.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(83.6-89.8]</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>windy</Name>
                    <Value>FALSE</Value>
                </Attribute>
            </Body>
            <Support>2.0</Support>
            <Confidence>1.0</Confidence>
        </Rule>
    </Rules>
</Association>
```

Resultado da execução do algoritmo *Tertius*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Association xmlns="http://www.pucrs.br/inf/pos"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.pucrs.br/inf/pos ../xsd/Association.xsd">
    <Info>
        <Name>weather</Name>
        <Algorithm>Tertius</Algorithm>
    </Info>
    <Rules>
        <Rule>
            <Head>
                <Attribute>
                    <Name>play</Name>
                    <Value>yes</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(72.4-76.6]</Value>
                </Attribute>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(77.4-83.6]</Value>
                </Attribute>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>overcast</Value>
                </Attribute>
            </Body>
            <Support>0.727325</Support>
            <Confidence>0.142857</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(89.8-inf)</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(68.2-72.4]</Value>
                </Attribute>
                <Attribute>
                    <Name>play</Name>
                    <Value>no</Value>
                </Attribute>
            </Body>
            <Support>0.642658</Support>
            <Confidence>0.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(89.8-inf)</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(68.2-72.4]</Value>
                </Attribute>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>sunny</Value>
                </Attribute>
            </Body>
```

```
            <Support>0.642658</Support>
            <Confidence>0.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(68.2-72.4]</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(89.8-inf)</Value>
                </Attribute>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>sunny</Value>
                </Attribute>
            </Body>
            <Support>0.642658</Support>
            <Confidence>0.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>sunny</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(-inf-71.2]</Value>
                </Attribute>
                <Attribute>
                    <Name>play</Name>
                    <Value>no</Value>
                </Attribute>
            </Body>
            <Support>0.642658</Support>
            <Confidence>0.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>windy</Name>
                    <Value>FALSE</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(80.8-inf)</Value>
                </Attribute>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(77.4-83.6]</Value>
                </Attribute>
            </Body>
            <Support>0.641109</Support>
            <Confidence>0.214286</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>play</Name>
                    <Value>yes</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
```

```
                <Value>(72.4-76.6]</Value>
            </Attribute>
            <Attribute>
                <Name>outlook</Name>
                <Value>overcast</Value>
            </Attribute>
        </Body>
        <Support>0.641109</Support>
        <Confidence>0.214286</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>play</Name>
                <Value>yes</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>humidity</Name>
                <Value>(77.4-83.6]</Value>
            </Attribute>
            <Attribute>
                <Name>outlook</Name>
                <Value>overcast</Value>
            </Attribute>
        </Body>
        <Support>0.641109</Support>
        <Confidence>0.214286</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>humidity</Name>
                <Value>(89.8-inf)</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>temperature</Name>
                <Value>(68.2-72.4]</Value>
            </Attribute>
        </Body>
        <Support>0.633754</Support>
        <Confidence>0.071429</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>temperature</Name>
                <Value>(68.2-72.4]</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>humidity</Name>
                <Value>(89.8-inf)</Value>
            </Attribute>
        </Body>
        <Support>0.633754</Support>
        <Confidence>0.071429</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>play</Name>
                <Value>yes</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>temperature</Name>
                <Value>(72.4-76.6]</Value>
```

```
                </Attribute>
                <Attribute>
                    <Name>windy</Name>
                    <Value>FALSE</Value>
                </Attribute>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>overcast</Value>
                </Attribute>
            </Body>
            <Support>0.590214</Support>
            <Confidence>0.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(80.8-inf)</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(83.6-89.8]</Value>
                </Attribute>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>overcast</Value>
                </Attribute>
            </Body>
            <Support>0.590214</Support>
            <Confidence>0.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(89.8-inf)</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(68.2-72.4]</Value>
                </Attribute>
                <Attribute>
                    <Name>windy</Name>
                    <Value>TRUE</Value>
                </Attribute>
            </Body>
            <Support>0.555556</Support>
            <Confidence>0.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>play</Name>
                    <Value>no</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>windy</Name>
                    <Value>TRUE</Value>
                </Attribute>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>sunny</Value>
                </Attribute>
            </Body>
            <Support>0.555556</Support>
            <Confidence>0.0</Confidence>
        </Rule>
```

```
<Rule>
    <Head>
        <Attribute>
            <Name>humidity</Name>
            <Value>(-inf-71.2]</Value>
        </Attribute>
    </Head>
    <Body>
        <Attribute>
            <Name>temperature</Name>
            <Value>(-inf-68.2]</Value>
        </Attribute>
        <Attribute>
            <Name>outlook</Name>
            <Value>sunny</Value>
        </Attribute>
    </Body>
    <Support>0.538289</Support>
    <Confidence>0.0</Confidence>
</Rule>
<Rule>
    <Head>
        <Attribute>
            <Name>windy</Name>
            <Value>TRUE</Value>
        </Attribute>
        <Attribute>
            <Name>play</Name>
            <Value>no</Value>
        </Attribute>
    </Head>
    <Body>
        <Attribute>
            <Name>temperature</Name>
            <Value>(76.6-80.8]</Value>
        </Attribute>
        <Attribute>
            <Name>outlook</Name>
            <Value>rainy</Value>
        </Attribute>
    </Body>
    <Support>0.538289</Support>
    <Confidence>0.0</Confidence>
</Rule>
<Rule>
    <Head>
        <Attribute>
            <Name>windy</Name>
            <Value>TRUE</Value>
        </Attribute>
    </Head>
    <Body>
        <Attribute>
            <Name>temperature</Name>
            <Value>(68.2-72.4]</Value>
        </Attribute>
        <Attribute>
            <Name>humidity</Name>
            <Value>(-inf-71.2]</Value>
        </Attribute>
        <Attribute>
            <Name>play</Name>
            <Value>no</Value>
        </Attribute>
    </Body>
    <Support>0.538289</Support>
    <Confidence>0.0</Confidence>
</Rule>
<Rule>
    <Head>
        <Attribute>
            <Name>windy</Name>
            <Value>TRUE</Value>
```

```
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>temperature</Name>
                <Value>(68.2-72.4]</Value>
            </Attribute>
            <Attribute>
                <Name>humidity</Name>
                <Value>(-inf-71.2]</Value>
            </Attribute>
            <Attribute>
                <Name>outlook</Name>
                <Value>sunny</Value>
            </Attribute>
        </Body>
        <Support>0.538289</Support>
        <Confidence>0.0</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>windy</Name>
                <Value>FALSE</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>temperature</Name>
                <Value>(80.8-inf)</Value>
            </Attribute>
            <Attribute>
                <Name>humidity</Name>
                <Value>(77.4-83.6]</Value>
            </Attribute>
            <Attribute>
                <Name>outlook</Name>
                <Value>sunny</Value>
            </Attribute>
        </Body>
        <Support>0.535599</Support>
        <Confidence>0.071429</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>windy</Name>
                <Value>FALSE</Value>
            </Attribute>
            <Attribute>
                <Name>play</Name>
                <Value>yes</Value>
            </Attribute>
        </Head>
        <Body>
            <Attribute>
                <Name>temperature</Name>
                <Value>(80.8-inf)</Value>
            </Attribute>
            <Attribute>
                <Name>humidity</Name>
                <Value>(77.4-83.6]</Value>
            </Attribute>
        </Body>
        <Support>0.53193</Support>
        <Confidence>0.142857</Confidence>
    </Rule>
    <Rule>
        <Head>
            <Attribute>
                <Name>windy</Name>
                <Value>TRUE</Value>
            </Attribute>
```

```
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(76.6-80.8]</Value>
                </Attribute>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(-inf-71.2]</Value>
                </Attribute>
            </Body>
            <Support>0.53193</Support>
            <Confidence>0.142857</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>play</Name>
                    <Value>yes</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(72.4-76.6]</Value>
                </Attribute>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(-inf-71.2]</Value>
                </Attribute>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>overcast</Value>
                </Attribute>
            </Body>
            <Support>0.53193</Support>
            <Confidence>0.142857</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>play</Name>
                    <Value>yes</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(-inf-68.2]</Value>
                </Attribute>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(-inf-71.2]</Value>
                </Attribute>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>overcast</Value>
                </Attribute>
            </Body>
            <Support>0.53193</Support>
            <Confidence>0.142857</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>humidity</Name>
                    <Value>(83.6-89.8]</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(80.8-inf)</Value>
```

```
                </Attribute>
            </Body>
            <Support>0.503827</Support>
            <Confidence>0.0</Confidence>
        </Rule>
        <Rule>
            <Head>
                <Attribute>
                    <Name>windy</Name>
                    <Value>FALSE</Value>
                </Attribute>
                <Attribute>
                    <Name>outlook</Name>
                    <Value>overcast</Value>
                </Attribute>
            </Head>
            <Body>
                <Attribute>
                    <Name>temperature</Name>
                    <Value>(80.8-inf)</Value>
                </Attribute>
            </Body>
            <Support>0.503827</Support>
            <Confidence>0.0</Confidence>
        </Rule>
    </Rules>
</Association>
```

Resultado da transformação de resultados de algoritmos de *Associação* para *Árvores de Decisão*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<DecisionTree xmlns:pos="http://www.pucrs.br/inf/pos"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.pucrs.br/inf/pos ../xsd/DecisionTree.xsd">
    <Info>
        <Name>weather</Name>
        <Description />
        <Algorithm>Apriori</Algorithm>
        <Visualization>Decision Tree</Visualization>
    </Info>
    <Node>
        <Name>Default Association Root</Name>
        <Children>
            <Child>
                <Value>|outlook|</Value>
                <Node>
                    <Name>|overcast|</Name>
                    <Children>
                        <Child>
                            <Value>|play|</Value>
                            <Result>|yes|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>|temperature|</Value>
                <Node>
                    <Name>|(80.8-inf)|</Name>
                    <Children>
                        <Child>
                            <Value>|windy|</Value>
                            <Result>|FALSE|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>|outlook||play|</Value>
                <Node>
                    <Name>|rainy||yes|</Name>
                    <Children>
                        <Child>
                            <Value>|windy|</Value>
                            <Result>|FALSE|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>|outlook||windy|</Value>
                <Node>
                    <Name>|rainy||FALSE|</Name>
                    <Children>
                        <Child>
                            <Value>|play|</Value>
                            <Result>|yes|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>|humidity|</Value>
                <Node>
                    <Name>|(77.4-83.6]|</Name>
                    <Children>
                        <Child>
                            <Value>|outlook|</Value>
```

```
                            <Result>|rainy|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>|temperature|</Value>
                <Node>
                    <Name>|(72.4-76.6]|</Name>
                    <Children>
                        <Child>
                            <Value>|play|</Value>
                            <Result>|yes|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>|humidity|</Value>
                <Node>
                    <Name>|(83.6-89.8]|</Name>
                    <Children>
                        <Child>
                            <Value>|temperature|</Value>
                            <Result>|(80.8-inf)|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>|humidity|</Value>
                <Node>
                    <Name>|(77.4-83.6]|</Name>
                    <Children>
                        <Child>
                            <Value>|windy|</Value>
                            <Result>|FALSE|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>|humidity|</Value>
                <Node>
                    <Name>|(77.4-83.6]|</Name>
                    <Children>
                        <Child>
                            <Value>|play|</Value>
                            <Result>|yes|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>|humidity|</Value>
                <Node>
                    <Name>|(83.6-89.8]|</Name>
                    <Children>
                        <Child>
                            <Value>|windy|</Value>
                            <Result>|FALSE|</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
        </Children>
    </Node>
</DecisionTree>
```

Resultado da transformação de resultados de algoritmos de *Classificação* para *Árvores de Decisão*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<DecisionTree xmlns:pos="http://www.pucrs.br/inf/pos"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.pucrs.br/inf/pos ../xsd/DecisionTree.xsd">
    <Info>
        <Name>weather</Name>
        <Description />
        <Algorithm>J48</Algorithm>
        <Visualization>Decision Tree</Visualization>
    </Info>
    <Node>
        <Name>outlook</Name>
        <Children>
            <Child>
                <Value>sunny</Value>
                <Node>
                    <Name>humidity</Name>
                    <Children>
                        <Child>
                            <Value>(-inf-71.2]</Value>
                            <Result>yes</Result>
                        </Child>
                        <Child>
                            <Value>(71.2-77.4]</Value>
                            <Result>no</Result>
                        </Child>
                        <Child>
                            <Value>(77.4-83.6]</Value>
                            <Result>no</Result>
                        </Child>
                        <Child>
                            <Value>(83.6-89.8]</Value>
                            <Result>no</Result>
                        </Child>
                        <Child>
                            <Value>(89.8-inf)</Value>
                            <Result>no</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
            <Child>
                <Value>overcast</Value>
                <Result>yes</Result>
            </Child>
            <Child>
                <Value>rainy</Value>
                <Node>
                    <Name>windy</Name>
                    <Children>
                        <Child>
                            <Value>TRUE</Value><Result>no</Result>
                        </Child>
                        <Child>
                            <Value>FALSE</Value><Result>yes</Result>
                        </Child>
                    </Children>
                </Node>
            </Child>
        </Children>
    </Node>
</DecisionTree>
```