Aquisição de conhecimento de conjuntos de
exemplos no formato atributo valor utilizando
aprendizado de máquina relacional

Mariza Ferro

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 28/07/2004

Assinatura: _____

Aquisição de conhecimento de conjuntos de exemplos no formato atributo valor utilizando aprendizado de máquina relacional¹

Mariza Ferro

Orientadora: *Prof^a Dr^a Maria Carolina Monard*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos necessários para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Julho/2004

¹Trabalho realizado com auxílio financeiro da CAPES.

Este documento foi preparado utilizando-se o formatador de textos $\text{\LaTeX}$. Sua bibliografia é gerada automaticamente pelo $\text{\BibTeX}$, utilizando o estilo Chicago. Foi utilizado um estilo desenvolvido por Ronaldo Cristiano Prati e adaptado por Marcos Aurélio Domingues.

Dedicatória

*Aos meus pais, Claudio
e Marlene, meus irmãos, Claudia
e Rafael, meu namorado, Robinson.*

Agradecimentos

À professora Maria Carolina Monard que, além de transmitir conhecimento também foi exemplo de educadora e de postura ética diante de todas as situações. Pela sua confiança em meu trabalho e imensa paciência. Por todos os momentos que trabalhou comigo nos fins de semana e noites e também por seu apoio nos momentos difíceis.

Aos meus pais, Claudio e Marlene, e meus irmãos Claudia e Rafael pelo amor, carinho e apoio para que pudesse realizar o meu trabalho. À minha mãe agradeço pela sua doçura e por aceitar tantas vezes a minha ausência. Em especial ao meu pai, por sempre me apoiar nas decisões que tomei. Quando decidi que queria fazer minha pós-graduação no ICMC ele confiou em mim, me incentivou a lutar pelos meus sonhos e disse que sempre faria tudo que estivesse ao seu alcance para que eu conseguisse. Eu consegui pai, obrigada.

Ao meu querido Robinson Massayuki pela sua interminável paciência e companheirismo, constante apoio e incentivo. Por todos os fins de semana e noites que ficou comigo enquanto eu estudava. Pelo seu abraço nos momentos bons e difíceis dessa caminhada.

À amiga Huei Diana Lee que foi minha orientadora de graduação e guiou os meus primeiros passos na pesquisa. Hoje minha colega de trabalho no LABIC, agradeço pelas preciosas dicas e sugestões acrescentadas a este trabalho. Ao seu marido Paulo por ajudar tantas vezes nos trabalhos de pesquisa e pelo constante incentivo.

Ao colega, Edson Takashi, que me ajudou tantas vezes com minhas dúvidas e muitas vezes parou o seu trabalho para auxiliar no meu.

A todos os colegas do LABIC, em especial ao Gustavo, Ronaldo, Marcos Aurélio, Edson Melanda e Verônica pelo apoio.

A Deus, meu ponto de apoio constante.

Resumo

O Aprendizado de Máquina trata da questão de como desenvolver programas de computador capazes de aprender um conceito ou hipótese a partir de um conjunto de exemplos ou casos observados. Uma das dimensões que influencia no aprendizado é o grau de supervisão. No aprendizado supervisionado é fornecido ao algoritmo de aprendizado um conjunto de exemplos de treinamento descritos por um vetor de valores de características, ou atributos, e pelo rótulo da classe associada. Baseado no conjunto de treinamento o algoritmo de aprendizado induz a classificação de uma hipótese capaz de determinar corretamente a classe de novos exemplos ainda não rotulados. Linguagens de descrição são necessárias para descrever exemplos, conhecimento do domínio bem como as hipóteses aprendidas a partir dos exemplos. Em geral, essas linguagens podem ser divididas em dois tipos: linguagem baseada em atributo-valor ou proposicional e linguagem relacional. Algoritmos de aprendizado são classificados como proposicional ou relacional dependendo da linguagem de descrição que eles utilizam. Além disso, no aprendizado simbólico o objetivo é gerar a classificação de hipóteses que possam ser facilmente interpretadas pelos humanos. Algoritmos de aprendizado proposicional utilizam a representação atributo-valor, a qual é inadequada para representar objetos estruturados e relações entre esses objetos. Por outro lado, a Programação Lógica Indutiva (PLI) é realizada com o desenvolvimento de técnicas e ferramentas para o aprendizado relacional. Sistemas de PLI são capazes de aprender levando em consideração conhecimento do domínio na forma de um programa lógico e também usar a linguagem de programas lógicos para descrever o conhecimento induzido. Neste trabalho é proposto e implementado um módulo chamado *Kaeru* para converter dados no formato atributo-valor para o formato relacional utilizado pelo sistema de PLI Aleph. Esse sistema foi escolhido por ser um protótipo para explorar idéias e por sua capacidade de emular diversos sistemas de

PLI. São descritos uma série de experimentos realizados com quatro conjuntos de dados naturais e um conjunto de dados real no formato atributo valor. Utilizando o módulo conversor *Kaeru* esses dados foram convertidos para o formato relacional utilizado pelo Aleph e hipóteses de classificação foram induzidas utilizando aprendizado proposicional bem como aprendizado relacional. É mostrado também, que o aprendizado proposicional pode ser utilizado para incrementar o conhecimento do domínio utilizado pelos sistemas de aprendizado relacional para melhorar a qualidade das hipóteses induzidas.

Abstract

Machine Learning addresses the question of how to build computer programs that learn a concept or hypotheses from a set of examples, objects or cases. One of the dimensions that influences learning is the degree of supervision. In supervised learning, the learning algorithm is typically presented with a set of training examples where frequently each example is described by a vector of feature or attribute values and a class label. Based on the set of training examples, the learning algorithm induces a classifying hypotheses which assigns each new case to one of a set of pre-defined classes on the basis of observed attribute values. Thus, the learner is given direct feedback about the appropriateness of its performance. Descriptive languages are necessary in order to describe the set of examples, domain knowledge as well as the hypothesis learned from these examples. In general, these languages can be divided into two types: languages based on attribute values, or *propositional* languages, and *relational* languages. Learning algorithms are often classified as propositional or relational taking into consideration the descriptive language they use. Furthermore, in what is called symbolic learning the aim is to generate classifying hypotheses that can be easily understood by humans. Typical propositional learning algorithms employ the attribute value representation, which is inadequate for problem-domains that require reasoning about the structure of objects in the domain and relations among such objects. On the other hand, Inductive Logic Programming (ILP) is concerned with the development of techniques and tools for relational learning. ILP systems are able to take into account domain knowledge in the form of a logic program and also use the language of logic programs for describing the induced knowledge or hypothesis. In this work we propose and implement a module, named *Kaeru*, to convert data in the attribute-value format to the relational format used by the ILP system Aleph. Aleph was chosen since it is intended to be a prototype for

exploring ideas and through the years has evolved to emulate some of the functionality of several other ILP systems. We describe a series of experiments performed on four natural data sets and one real data set in the attribute value format. Using the *Kaeru* module these data sets were converted to the relational format used by Aleph and classifying hypotheses were induced using propositional as well as relational learning. We also show that propositional knowledge can be used to increment the background knowledge used by relational learners in order to improve the induced hypotheses quality.

Sumário

1	Introdução	1
2	Aprendizado de Máquina e Linguagens de Representação	7
2.1	Classificação dos Sistemas de Aprendizado	7
2.2	Aprendizado Indutivo de Conceitos	10
2.3	Descrição de Objetos e Conceitos	11
2.3.1	Descrição Baseada em Atributos	12
2.3.2	Descrição Relacional	14
2.3.3	Representação Proposicional <i>versus</i> Relacional	18
2.4	Considerações Finais	18
3	Programação Lógica Indutiva	20
3.1	Introdução à PLI	20
3.2	O Problema da PLI	22
3.2.1	PLI Preditiva	23
3.2.2	PLI Descritiva	24
3.2.3	Exemplo de PLI Preditiva e Descritiva	24
3.2.4	Completeness e Consistência de uma Hipótese	25
3.3	<i>Bias</i>	28
3.4	Estruturando o Espaço de Cláusulas	29

3.5	Métodos de Generalização	30
3.6	Métodos de Especialização	33
3.7	Considerações Finais	35
4	Sistemas de Programação Lógica Indutiva	37
4.1	Dimensões da PLI	37
4.2	Descrição dos sistemas de PLI	39
4.3	O Sistema Aleph	41
4.3.1	Funcionamento Básico	42
4.3.2	Declarações de Modo	42
4.3.3	Tipos	44
4.3.4	Declaração dos Determinations	44
4.3.5	Exemplos Positivos e Negativos	45
4.3.6	Parâmetros	45
4.3.7	Características Importantes	46
4.3.8	Um Exemplo de Execução	46
4.4	Considerações Finais	50
5	A Ferramenta Proposta	51
5.1	O Projeto DISCOVER	52
5.1.1	A Biblioteca de Classes Discover Object Library - DOL	53
5.1.2	O Ambiente Computacional SNIFFER	54
5.1.3	A Sintaxe Padrão para Conjuntos de Dados no Formato Atributo-Valor	54
5.1.4	Tipos de Dados Implementados	56
5.2	O Módulo Conversor <i>Kaeru</i>	57
5.2.1	Descrição do Módulo Conversor <i>Kaeru</i>	58
5.2.2	Formato dos Arquivos de Entrada	59
5.2.3	Formato do Arquivo de Saída .b	62
5.2.4	Exemplo do Arquivo de Saída .b	64

5.2.5	Considerações Finais	65
6	Experimentos com Dados Naturais	67
6.1	Conjuntos de Dados	67
6.1.1	Descrição	68
6.1.2	Características	69
6.2	Descrição dos Experimentos	70
6.2.1	<i>See5</i>	70
6.2.2	Aprendizado Proposicional	71
6.2.3	Aprendizado Relacional	72
6.2.4	Análise dos Resultados	78
6.3	Considerações Finais	78
7	Estudo de Caso	81
7.1	Fases do Estudo de Caso	81
7.2	O Problema do Processamento de Sêmen	83
7.3	Descrição dos Conjuntos de Dados	86
7.4	Descrição dos Experimentos	87
7.4.1	Etapa 1	88
7.4.2	Etapa 2	89
7.5	Considerações Finais	94
8	Conclusões	96
A	Parâmetros do Aleph	99
B	Conhecimento do Domínio do Conjunto de Dados Processamento de Sêmen	105
	Referências	116

Lista de Figuras

2.1	Hierarquia do aprendizado indutivo	11
3.1	Intersecção de aprendizado de máquina e programação lógica - PLI	21
3.2	Completeness e consistência de uma hipótese	27
3.3	Uma árvore de derivação linear inversa	34
3.4	Parte do grafo de refinamento gerado no aprendizado filha/2	35
3.5	Como os conceitos para buscas no espaço de hipóteses se relacionam	36
4.1	Trens que viajam para Leste e Oeste	47
5.1	Interação entre filtros, sintaxes e bibliotecas	53
5.2	Ambiente de PLI: interação entre o módulo conversor <i>Kaeru</i> , o Aleph, a biblioteca DOL e o Ambiente SNIFFER PLI	58
7.1	Fases do estudo de caso	82
7.2	Injeção Intracitoplasmática do Espermatozóides no Oócito - ICSI	84

Lista de Tabelas

2.1	Características dos sistemas de Aprendizado de Máquina	8
2.2	Formato atributo-valor para dados	13
2.3	Cobertura de uma Regra $B \rightarrow H$	14
3.1	Exemplos de treinamento e conhecimento do domínio para aprendizado relacional	25
3.2	Exemplos de <i>lgg</i> de termos	31
3.3	Exemplos de <i>lgg</i> de literais	31
4.1	Resumo dos sistemas de PLI	41
4.2	Arquivos <code>trens.f</code> e <code>trens.n</code>	47
5.1	Subconjunto de dados <code>voyage</code>	55
5.2	Exemplo de arquivo de declaração de atributos: <code>voyage.names</code>	56
5.3	Exemplo de arquivo de declaração de dados: <code>voyage.data</code>	56
5.4	Exemplo de arquivo de declaração de parâmetros do módulo conversor <i>Kaeru</i> : <code>voyage.bk</code>	60
5.5	Exemplo de arquivo de declaração de parâmetros do Aleph: <code>voyage.param</code>	61
5.6	Exemplo do arquivo de saída do módulo conversor <i>Kaeru</i> : <code>voyage.b</code>	66
6.1	Breast-cancer – Descrição dos atributos	68

6.2	Breast cancer2 – Descrição dos atributos	69
6.3	Bupa – Descrição dos atributos	69
6.4	Pima – Descrição dos atributos	69
6.5	Resumo dos conjuntos de dados	70
6.6	Resultados dos experimentos com <i>See5</i>	71
6.7	Resultados dos experimentos com Aleph - Etapa 1	73
6.8	Conjunto de exemplos <code>teste1</code>	73
6.9	Arquivos <code>teste1.f</code> e <code>teste1.n</code>	74
6.10	Arquivos <code>teste1.f</code> com <code>class(7,pos)</code> adicionado e <code>teste1.n</code>	76
6.11	Atributos e valores que separam classes no conjunto breast-cancer	77
6.12	Resultados dos experimentos com Aleph - Etapa 2	78
7.1	Processamento de sêmen – Descrição dos atributos	87
7.2	Resumo do conjunto de dados processamento de sêmen utilizado nos ex- perimentos	87
7.3	Resultados dos experimentos com o conjunto processamento de sêmen - Etapa 1	88
7.4	Número médio de exemplos cobertos pelas regras - Etapa 1	89
7.5	Resultados dos experimentos com o conjunto processamento de sêmen - Etapa 2	90
7.6	Número médio de exemplos cobertos pelas regras - Etapa2	91
B.1	Conhecimento adicional com-ad1	105
B.2	Conhecimento adicional com-ad2	106
B.3	Conhecimento adicional com-ad3	106
B.4	Conhecimento adicional com-ad4	107
B.5	Conhecimento adicional com-ad5	108

Lista de Abreviaturas

AM Aprendizado de Máquina

FIV Fertilização In Vitro

IA Inteligência Artificial

LABIC Laboratório de Inteligência Computacional

LPO Linguagem de Primeira Ordem

PLI Programação Lógica Indutiva

ICSI Injeção Intracitoplasmática do Espermatozóide no Óvulo

IUI Inseminação Intra Uterina

RBC Raciocínio Baseado em Casos

Introdução

Significativos avanços tecnológicos vêm sendo alcançados nos últimos anos tanto na área de *software* como de *hardware*. Esses avanços trouxeram para sociedade contemporânea enormes mudanças, as quais possibilitam a aplicação dos computadores nas mais diversas áreas e sua conseqüente popularização. Aliado a isso veio o rápido desenvolvimento das tecnologias relacionadas ao armazenamento e à comunicação, o que tornou possível a coleta e o armazenamento de grandes volumes de dados. Isso implica em cada vez mais dados e informações sendo acumulados os quais precisam ser convertidos para uma forma mais concisa e inteligível para que possam ser úteis nas tomadas de decisão, ou mesmo para que seja descoberto conhecimento escondido nos dados. Atualmente há um grande interesse na extração de conhecimento de grandes volumes de dados, tanto científicos como industriais. Esse é o objetivo do processo de mineração de dados, que utiliza, entre outros, sistemas de Aprendizado de Máquina (AM) para descobrir padrões e conhecimento útil nos dados.

O AM pode ser definido como uma sub-área da Inteligência Artificial (IA) que estuda métodos computacionais para adquirir novos conhecimentos de forma automática bem como para organizar o conhecimento já existente. Um sistema de AM supervisionado é um programa de computador que induz hipóteses baseado em experiências acumuladas contidas em casos resolvidos com sucesso. Esses casos, ou exemplos, constituem o conjunto de exemplos de treinamento fornecidos ao algoritmo de aprendizado. Os diversos sistemas

de AM possuem características comuns que permitem classifica-los, entre outros, quanto ao paradigma de aprendizado utilizado. Existem diversos paradigmas de AM, exemplos desses são: paradigma simbólico, estatístico, conexionista, baseado em exemplos e evolutivo. No primeiro deles encontra-se os sistemas de aprendizado simbólico. Esse tipo de sistemas, tratados neste trabalho, buscam aprender construindo representações simbólicas de um conceito por meio da análise de exemplos e contra exemplos desse conceito. O aprendizado de um conceito nos sistemas de AM pode ser realizado por meio da indução.

A indução é a forma de inferência lógica que permite obter conclusões genéricas sobre um conjunto particular de exemplos, ou casos observados. É caracterizada como o raciocínio que parte do específico para o geral. Na indução, um conceito é aprendido efetuando-se inferência indutiva sobre os exemplos apresentados. É por aprendizado indutivo que os humanos, a partir de um conjunto de observações, chegam a descobrir algumas características particulares dessas observações, ou que chegam a realizar algumas generalizações que permitem “explicar” essas observações. O aprendizado indutivo pode ser dividido em supervisionado, não-supervisionado e semi-supervisionado. No aprendizado supervisionado, tratado neste trabalho, o objetivo do algoritmo de indução é construir um classificador que possa determinar corretamente a classe de novos exemplos não rotulados. Para rótulos de classe discretos, esse problema é conhecido como classificação e para valores contínuos como regressão.

Ao solucionar problemas com o uso do computador é importante que se defina como traduzi-los em termos computacionais. Especificamente em AM, isso significa como representar objetos (exemplos), conceitos (hipóteses) induzidos e conhecimento do domínio. Algumas linguagens de representação freqüentemente utilizadas em AM simbólico são: linguagens baseadas em atributo-valor ou proposicionais e linguagens baseadas em Linguagem de Primeira Ordem (LPO) ou relacionais.

O formato atributo-valor, ou proposicional, é a linguagem de representação de objetos mais freqüentemente utilizada em AM, embora sua baixa capacidade de expressão impeça a representação de objetos estruturados, bem como a relação entre objetos ou entre seus componentes. Assim, aspectos relevantes dos objetos, que de alguma maneira poderiam caracterizar o conceito que está sendo aprendido, não podem ser representados utilizando o formato atributo-valor. Por outro lado, na linguagem de representação relacional, objetos podem ser descritos em termos de seus componentes e relações entre esses componentes. Os sistemas de aprendizado que utilizam esse tipo de linguagem de representação são chamados de sistemas de aprendizado relacional. Esses sistemas possuem uma alta expressividade para representar conceitos e a importante habilidade de representar conhecimento do domínio, o que não acontece com os sistemas de aprendizado proposicional.

Desde 1990, quando Stephen Muggleton introduziu o nome Programação Lógica Indutiva (PLI), uma área de pesquisa começa a chamar a atenção da comunidade de AM. Essa área de pesquisa combina métodos e técnicas de AM indutivo com a representação da LPO e é chamada de PLI. Em 1991 Muggleton organizou o primeiro *workshop* internacional de PLI que passou a ocorrer todos os anos. Atualmente existem quatro categorias internacionais de eventos ocorrendo regularmente na área (conferências, *workshops*, seminários e escolas de verão), estabelecendo a PLI como uma próspera área de pesquisa. Desde 1991 dezenas de sistemas foram implementados e vem sendo aplicados com sucesso em diversas áreas, entre elas pode-se citar: síntese indutiva de programas, teoria do aprendizado, invenção de predicados e a área que vem crescendo muito nos últimos cinco anos, a mineração de dados relacional.

A PLI se diferencia da maioria dos outros modos de AM pelo uso de uma expressiva linguagem de representação e sua habilidade para utilizar conhecimento prévio do domínio. O conhecimento do domínio utilizado na construção de hipóteses é uma característica importante em PLI. Quando o conhecimento do domínio é relevante, pode-se melhorar substancialmente os resultados do aprendizado. Conhecimento do domínio irrelevante pode ter o efeito contrário. Muito da arte da PLI está na seleção e formulação apropriada do conhecimento do domínio para ser utilizado na tarefa de aprendizado.

Os sistemas de PLI se situam em duas sub-áreas, os sistemas interativos de PLI, os quais estão fortemente relacionados com os sistemas revisores de teoria, e um pequeno número de exemplos está disponível, e os sistemas empíricos, nos quais a ênfase é a extração de padrões de um grande número de exemplos. O foco deste trabalho está neste último tipo de sistemas. Entre os diversos sistemas empíricos encontra-se o sistema de PLI Aleph, muito bem considerados pela comunidade de PLI. O Aleph permite simular vários sistemas de PLI empíricos, o que motivou seu uso neste trabalho.

Um dos objetivos deste trabalho é a extração de conhecimento de bases de dados reais utilizando o sistema de PLI Aleph. Porém, muitas dessas bases de dados estão num formato originalmente proposicional (atributo-valor), que não permite explorar a expressividade dos sistemas de PLI. Para que essas bases possam ser utilizadas pelos sistemas de PLI, elas precisam passar por um processo de transformação que converte esse formato proposicional para um formato relacional equivalente. Assim, como parte dos objetivos deste trabalho, foi implementado um módulo específico para realizar essa transformação, chamado de módulo conversor *Kaeru*.

O módulo conversor *Kaeru*, além de transformar bases de dados proposicionais para o formato relacional cria os arquivos necessários para executar o sistema de PLI Aleph. O módulo conversor *Kaeru* é um dos módulos do ambiente de PLI, o qual será integrado futu-

ramente ao projeto DISCOVER. O projeto DISCOVER em desenvolvimento no Laboratório de Inteligência Computacional (LABIC) é um ambiente computacional, para descoberta de conhecimento em bases de dados, no qual estão integrados algoritmos de aprendizado proposicional implementados pela comunidade, bem como ferramentas específicas desenvolvidas por pesquisadores do LABIC, as quais oferecem funcionalidades voltadas para o aprendizado de máquina proposicional, mineração de dados e mineração de textos. Assim, o módulo *Kaeru* tem como objetivo permitir a incorporação de algoritmos de PLI no ambiente DISCOVER.

Além da extração de conhecimento relacional dessas bases de dados proposicionais, um outro objetivo deste trabalho é verificar a possibilidade de utilizar o conhecimento extraído desses conjuntos de dados utilizando algoritmos de aprendizado proposicional como forma de incrementar o conhecimento do domínio para aprendizado relacional. Neste trabalho foram realizados experimentos tanto com dados reais quanto naturais, que mostram que a utilização de conhecimento adicional do domínio, obtido a partir de modelos proposicionais, pode auxiliar no aprendizado relacional.

O sistema de PLI Aleph foi utilizado para extração de conhecimento relacional de bases de dados médicas. Como mencionado anteriormente, PLI permite a adição de conhecimento prévio do domínio. Em outras palavras, além do uso de bases de dados no formato atributo valor, transformadas com o módulo *Kaeru* para o formato relacional, e do conhecimento extraído por indutores proposicionais, é possível inserir conhecimento adicional fornecido pelo especialista. Entretanto, nessa fase do trabalho o especialista esteve impedido de participar.

Acreditamos que esse é um dos principais motivos pelo qual não foi possível induzir conhecimento relacional dessas bases de dados. Ainda assim, é interessante observar que o conhecimento proposicional induzido com Aleph é diferente do conhecimento proposicional induzido por indutores proposicionais.

Este trabalho está organizado da seguinte forma: No Capítulo 2 são apresentados alguns conceitos sobre AM, a descrição de aprendizado indutivo de conceitos, a linguagem de descrição baseada em atributo-valor, a linguagem de descrição relacional bem como a notação e terminologia utilizadas.

No Capítulo 3 são apresentados conceitos básicos sobre PLI. O problema de aprendizado da PLI é apresentado formalmente bem como os métodos básicos de PLI para estruturar o espaço de hipóteses e sistematizar a busca. Métodos de generalização e especialização são também apresentados.

No Capítulo 4, são apresentadas as dimensões em que os sistemas de PLI podem ser

divididos, suas características e alguns sistemas de PLI. O sistema de PLI Aleph é descrito com maiores detalhes.

No Capítulo 5 são apresentados o projeto **DISCOVER**, bem como a biblioteca de classes **DOL** e o ambiente **SNIFFER**, além da sintaxe padrão utilizada para representar conjuntos de dados no formato atributo valor as quais são utilizadas no desenvolvimento do módulo conversor *Kaeru*. É descrito também o módulo conversor *Kaeru*, sua arquitetura e funcionamento e como ele interage com o projeto **DISCOVER** e com o sistema de PLI Aleph.

No Capítulo 6 são descritos os experimentos realizados utilizando bases de dados naturais da área médica, cujo formato original é o formato atributo-valor, que foram transformadas para o formato relacional utilizando o módulo conversor *Kaeru* e submetidas ao sistema de PLI Aleph.

No Capítulo 7 é apresentado um estudo de caso realizado com uma base de dados real da área médica, relacionado ao processamento de sêmen, originalmente no formato atributo-valor e transformada para o formato relacional. São descritas todas as etapas desenvolvidas durante o estudo de caso bem como os resultados alcançados.

Finalmente no Capítulo 8 são apresentadas as conclusões deste trabalho e trabalhos futuros.

Aprendizado de Máquina e Linguagens de Representação

O AM é uma área da IA cujo objetivo é projetar e desenvolver sistemas capazes de adquirir conhecimento de maneira automática. Neste capítulo são descritas várias abordagens que podem ser utilizadas pelos sistemas de aprendizado computacional, entre essas abordagens encontra-se o aprendizado por indução, tratado neste trabalho. O aprendizado indutivo permite obter novos conhecimentos a partir de exemplos, previamente, observados. Entretanto, ele é um dos mais desafiadores, pois o conhecimento gerado ultrapassa o limite das premissas, e não existem garantias de que esse conhecimento seja verdadeiro.

2.1 Classificação dos Sistemas de Aprendizado

Um sistema de AM é um programa de computador que toma decisões baseado em experiências acumuladas contidas em casos resolvidos com sucesso (Weiss & Kulikowski, 1991). Esses casos constituem o conjunto de exemplos de treinamento fornecidos ao algoritmo de aprendizado. Os diversos sistemas de AM possuem características que possibilitam sua classificação quanto ao paradigma, modo, forma de aprendizado e linguagem de descrição

utilizada para descrever exemplos e conhecimento. Na Tabela 2.1 estão resumidas algumas características dos sistemas de AM discutidos nesta seção.

Modos	Paradigmas	Formas	Linguagens de Descrição
Supervisionado	Simbólico	Incremental	Exemplos ou Objetos - $\mathcal{L}_\mathcal{E}$
Não-Supervisionado	Estatístico	Não-Incremental	Hipóteses - $\mathcal{L}_h$
Semi-Supervisionado	Baseado em Exemplos		Conhecimento do domínio - $\mathcal{L}_\mathcal{K}$
	Conexionista		
	Evolutivo		

Tabela 2.1: Características dos sistemas de Aprendizado de Máquina

Os modos de aprendizado podem ser divididos em *supervisionado*, *não-supervisionado* e *semi-supervisionado*. No caso do aprendizado supervisionado, os conjuntos de exemplos (observações) fornecidos para o algoritmo de aprendizado (ou indutor) estão rotulados com suas respectivas classes. Nesse caso, o objetivo do algoritmo de indução é construir um classificador que possa determinar corretamente a classe de novos exemplos ainda não rotulados. Uma das maiores restrições do aprendizado supervisionado é a necessidade de um conjunto de exemplos com uma quantidade expressiva de exemplos rotulados para a indução de um bom classificador, o que nem sempre acontece nas bases de dados.

Já no aprendizado *não-supervisionado*, os exemplos não estão rotulados. O algoritmo analisa os exemplos fornecidos e tenta agrupá-los de alguma maneira, utilizando algum critério de similaridade, formando agrupamentos ou *clusters*.

Recentemente surgiu um terceiro tipo de aprendizado de máquina no qual são utilizados poucos exemplos rotulados ao invés de uma quantidade expressiva necessária para o aprendizado supervisionado. Essa área é denominada aprendizado *semi-supervisionado* (Blum & Mitchell, 1998; Matsubara, 2004; Sanches, 2003). O aprendizado semi-supervisionado representa a junção do aprendizado supervisionado e não-supervisionado, e tem o potencial de reduzir a necessidade de dados rotulados quando somente um pequeno conjunto de exemplos rotulados está disponível.

Dentro da área de AM foram propostos vários *paradigmas de aprendizado*, capazes de aprender a partir de um conjunto de exemplos, tais como: simbólico, estatístico, baseado em exemplos, conexonista e evolutivo (Mitchell, 1998), os quais são descritos a seguir.

Simbólico Os sistemas de aprendizado simbólico buscam aprender construindo representações simbólicas de um conceito por meio da análise de exemplos e contra-exemplos desse conceito. As representações simbólicas mais comumente utilizadas são árvores de decisão, regras de decisão e linguagens lógicas de primeira ordem;

Estatístico Diversos métodos de classificação têm sido criados pelos pesquisadores em es-

tatística. A idéia geral desses métodos consiste em utilizar modelos estatísticos para encontrar uma boa aproximação do conceito induzido. Entre os métodos estatísticos, destacam-se os de aprendizado Bayesiano, que utilizam um modelo probabilístico baseado no conhecimento prévio do problema, o qual é combinado com os exemplos de treinamento para determinar a probabilidade final de uma hipótese;

Baseado em Exemplos Uma maneira de classificar um caso é lembrar de um caso similar cuja classe é conhecida e assumir que o novo caso terá a mesma classe. Essa característica explica os sistemas baseados em exemplos, que classificam casos nunca vistos por meio de casos similares conhecidos. As técnicas mais conhecidas nesse paradigma são *Nearest Neighbours* e Raciocínio Baseado em Casos (RBC);

Conexionista Redes Neurais são construções matemáticas simplificadas inspiradas no modelo biológico do sistema nervoso. A representação de uma rede neural envolve unidades altamente interconectadas, justificando o nome conexionismo para descrever essa área de estudo;

Evolutivo Este paradigma faz uma analogia com a teoria de Darwin, na qual somente os mais adaptados sobrevivem. Um classificador evolutivo consiste de uma população de elementos de classificação que competem entre si para fazer a predição. Os elementos de performance fraca serão descartados e os mais fortes proliferarão produzindo variações sobre eles próprios.

A *forma de aprendizado* diz respeito ao modo como os exemplos são apresentados ao algoritmo de aprendizado, sendo classificadas como *não-incremental*, também conhecida como modo *batch*, e *incremental*. Algoritmos não-incrementais exigem que todos os exemplos estejam, simultaneamente, disponíveis para o algoritmo de aprendizado; esses algoritmos devem ser utilizados quando todos os exemplos estão disponíveis e não sofrem mudanças durante o processo de aquisição de conhecimento. Já os algoritmos incrementais modificam, se necessário, a definição do conhecimento (hipótese) adquirido a cada novo exemplo observado. Portanto, no modo incremental o algoritmo tenta atualizar a hipótese antiga sempre que novos exemplos são adicionados ao conjunto de treinamento.

Ao solucionar problemas com o uso do computador, é importante definir como traduzi-los em termos computacionais. Especificamente em AM, isso significa como descrever exemplos, hipóteses e conhecimento do domínio. Para essa finalidade são usadas as seguintes *linguagens de descrição*:

- Linguagens de descrição de exemplos, $\mathcal{L}_E$;

- Linguagens de descrição de hipóteses, $\mathcal{L}_h$;
- Linguagens de descrição de conhecimento do domínio, $\mathcal{L}_K$.

As linguagens de descrição mais freqüentemente utilizadas em AM simbólico, em ordem crescente de complexidade e força expressiva, são: linguagem de ordem zero, linguagem baseada em atributos, linguagem baseada em lógica de primeira ordem e linguagem de segunda ordem. Na Seção 2.3 na página oposta são descritas linguagens de descrição baseadas em atributos (ou proposicional) e baseadas em lógica de primeira ordem (ou relacional), utilizadas neste trabalho.

2.2 Aprendizado Indutivo de Conceitos

A indução é a forma de inferência lógica que permite obter conclusões genéricas sobre um conjunto particular de exemplos, ou casos observados. É caracterizada como o raciocínio que parte do específico para o geral. Na indução, um conceito é aprendido efetuando-se inferência indutiva sobre os exemplos apresentados. Portanto, as hipóteses geradas por meio da inferência indutiva podem ou não preservar a verdade. Mesmo assim, a inferência indutiva é um dos principais métodos utilizados para derivar conhecimento novo e predizer eventos futuros. Foi por meio da indução que Arquimedes descobriu a primeira lei da hidrostática e o princípio da alavanca, Kepler descobriu as leis do movimento planetário e Darwin descobriu as leis da seleção natural das espécies.

No caso do *aprendizado indutivo por exemplos*, são dados ao aprendiz alguns exemplos e a tarefa é induzir descrições gerais de conceitos utilizando exemplos específicos desse conceito (Michalski et al., 1983). O aprendizado indutivo vem sendo aplicado com sucesso em vários problemas de classificação e regressão, tais como o diagnóstico de pacientes ou doenças em plantas e a predição de propriedades de materiais com base em suas características químicas. Esses problemas podem ser formulados como tarefas de aprendizado de conceitos por exemplos, referenciado como *aprendizado indutivo de conceitos*.

Para definir o problema de *aprendizado indutivo de conceitos*, primeiramente é necessário definir o que é *conceito*. Seja $\mathcal{U}$ o conjunto universal de objetos (ou observações), um conceito $\mathcal{C}$ pode ser formalizado como um subconjunto de objetos em $\mathcal{U} : \mathcal{C} \subseteq \mathcal{U}$. Por exemplo, $\mathcal{U}$ pode ser o conjunto de todos os pacientes em uma base de dados e $\mathcal{C} \subseteq \mathcal{U}$ o conjunto de todos os pacientes que apresentam uma determinada doença. Aprender um conceito $\mathcal{C}$ significa aprender a reconhecer objetos a em $\mathcal{C}$, ou seja, ser capaz de dizer se $a \in \mathcal{C}$ para cada $a \in \mathcal{U}$ (Lavrač & Džeroski, 1994).

Figura 2.1: Hierarquia do aprendizado indutivo

O aprendizado indutivo pode ser dividido em supervisionado, não-supervisionado e semi-supervisionado. No aprendizado supervisionado, como já descrito na Seção 2.1 na página 7, o objetivo do algoritmo de indução é construir um classificador que possa determinar corretamente a classe de novos exemplos não rotulados. Para rótulos de classe discretos, esse problema é conhecido como *classificação* e para valores contínuos como *regressão*. Na Figura 2.1 é mostrada a hierarquia do aprendizado indutivo, na qual os nós mais escuros representam o aprendizado supervisionado para problemas de classificação, o qual é tratado neste trabalho.

2.3 Descrição de Objetos e Conceitos

Para a expressão de qualquer paradigma de AM são necessárias, no mínimo, linguagens que descrevam os objetos e os conceitos aprendidos. Como mencionado anteriormente, objetos são descritos em uma linguagem de descrição de objetos $\mathcal{L}_e$. Conceitos podem ser descritos em uma mesma linguagem ou em uma linguagem diferente para descrição de conceitos, $\mathcal{L}_h$ — Tabela 2.1 na página 8.

Em AM simbólico, vários formalismos lógicos têm sido utilizados em sistemas de aprendizado indutivo para a representação de exemplos e conceitos. Em geral, distingue-se dois tipos de descrição: descrição baseada em atributos ou proposicional e descrição relacional (Lavrač & Džeroski, 1994).

2.3.1 Descrição Baseada em Atributos

As linguagens baseadas em atributo-valor para representar exemplos e conceitos, são equivalentes às linguagens proposicionais. As vantagens dessa representação são a sua simplicidade e eficiência, bem como uma grande quantidade de técnicas desenvolvidas para lidar com ruído nos dados (Monard & Baranauskas, 2003a).

Alguns dos algoritmos/sistemas de aprendizado indutivo que têm sido mais utilizados são os da família TDIDT¹ (Quinlan, 1986, 1988) e o $\mathcal{CN}^2$ (Boswell, 1990a; Clark & Niblett, 1989), os quais utilizam linguagens baseadas em atributos para representar conceitos. Apesar do relativo sucesso desses sistemas proposicionais de aprendizado, tais sistemas são fortemente limitados, justamente pela linguagem de descrição de exemplos e de conceitos empregada. Uma outra limitação dos sistemas de aprendizado proposicionais, é que eles não consideram, ou consideram de uma maneira muito limitada, conhecimento do domínio², o que não acontece com os sistemas de aprendizado relacionais.

A seguir são descritas a linguagem de descrição de objetos (ou exemplos) e a linguagem de descrição de conceitos (ou hipóteses) mais freqüentemente utilizada pelos sistemas de AM proposicionais.

Notação e Terminologia

O formato atributo-valor é a linguagem de descrição de objetos ($\mathcal{L}_{\mathcal{E}}$) mais freqüentemente utilizada em AM. Objetos, ou exemplos, são descritos em termos de atributos e valores desses atributos, por meio de um vetor, contendo valores para os atributos de um determinado exemplo e um rótulo que atribui uma classe ao exemplo, quando o aprendizado for supervisionado.

A Tabela 2.2 mostra o formato geral de um conjunto de dados T com N exemplos e M atributos, classificados segundo uma classe de interesse. Nessa tabela uma linha i refere-se ao i -ésimo exemplo ($i = 1, 2, \dots, N$) e x_{ij} refere-se ao valor do j -ésimo atributo X_j ($j = 1, 2, \dots, M$) do exemplo i . Assim, exemplos são pares $T_i = (x_{i1}, x_{i2}, \dots, x_{iM}, y_i) = (\mathbf{x}_i, y_i)$ e o conjunto de exemplos é referenciado como $(\mathbf{X}, Y)$, onde o último atributo, Y , é um atributo especial, denominado classe (ou rótulo), que se deseja predizer com base nos outros X atributos, isto é, $Y = f(\mathbf{X})$. Cada $\mathbf{x}_i$ é um elemento do conjunto $X_1 \times X_2 \times \dots \times X_M$ onde X_j é o domínio do j -ésimo atributo e y_i pertence a uma das N_{cl} classes, isto é, $Y \in \{C_1, C_2, \dots, C_{N_{cl}}\}$. Diz-se que exemplos descritos, segundo os apresentados na Tabela

¹ *Top Down Induction of Decision Trees.*, nos quais o conhecimento adquirido é representado na forma de árvores de decisão, por exemplo o $\mathcal{C4.5}$

² também denominado conhecimento de fundo ou conhecimento prévio do domínio.

2.2, estão no formato *atributo-valor*.

	X_1	X_2	$\dots$	X_M	Y
T_1	x_{11}	x_{12}	$\dots$	x_{1M}	y_1
T_2	x_{21}	x_{22}	$\dots$	x_{2M}	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
T_N	x_{N1}	x_{N2}	$\dots$	x_{NM}	y_N

Tabela 2.2: Formato atributo-valor para dados

Os conceitos, ou hipóteses, aprendidos por algoritmos de AM simbólicos são geralmente representados por árvores de decisão ou conjuntos de regras. Como sempre é possível escrever uma árvore de decisão como um conjunto de regras disjuntas, neste trabalho, o termo *regra* faz referência a uma regra extraída de uma árvore de decisão (regras disjuntas) ou uma regra diretamente induzida por um algoritmo de AM supervisionado.

Uma regra é geralmente representada na forma (Monard & Baranauskas, 2003b)

$$R: \text{if } \langle \text{condição} \rangle \text{ then } \langle \text{class} = C_i \rangle$$

onde C_i é um dos possíveis valores para a classe e $\langle \text{condição} \rangle$ é uma disjunção de conjunções de testes para os atributos da forma

$$X_i \text{ op valor}$$

onde $X_i \in \mathbf{X}$ e $op \in \{\leq, \geq, =\}$.

Para facilitar a leitura, será adotada uma representação mais genérica para qualquer regra R , onde

$$R: \underbrace{\text{if } \langle \text{condição} \rangle}_{\text{Body ou } B} \text{ then } \underbrace{\langle \text{class} = C_i \rangle}_{\text{Head ou } H}$$

passando a denotar uma regra como

$$Body \rightarrow Head$$

ou resumidamente $B \rightarrow H$.

Dizemos que um exemplo T_i é coberto por uma regra R se e somente se o exemplo satisfaz todas as condições da regra (todas as condições de uma regra são avaliadas como verdadeiras, dada a descrição do exemplo). Ou seja, um exemplo T_i é coberto por uma

regra R se e somente se B é verdade. Por outro lado, um exemplo que não satisfaz a condição B da regra não é coberto pela regra.

Dizemos que um exemplo T_i é corretamente coberto por uma regra R se e somente se T_i é coberto pela regra e a classe y_i do exemplo é a mesma prevista pela regra. Ou seja, um exemplo T_i é corretamente coberto pela regra R se e somente se B é verdade e H é verdade. Entretanto, se o exemplo satisfaz a condição B da regra mas não satisfaz a condição H , o exemplo é incorretamente coberto pela regra. Um resumo das quatro possíveis situações pode ser visto na Tabela 2.3 (Prati et al., 2001a).

Exemplos satisfazendo ...	são ...
B	cobertos pela regra
$\overline{B}$	não cobertos pela regra
$B \wedge H$	cobertos corretamente pela regra
$B \wedge \overline{H}$	cobertos incorretamente pela regra

Tabela 2.3: Cobertura de uma Regra $B \rightarrow H$

2.3.2 Descrição Relacional

A descrição relacional é baseada em Linguagem de Primeira Ordem, a qual é uma linguagem de representação de propósito geral que permite a descrição de objetos e relações entre os objetos. A LPO dá ao seu usuário a liberdade para descrever os objetos da maneira mais apropriada para seu domínio (Russel & Norvig, 2003), como no exemplo a seguir:

"Rei John, o maldoso, reinou na Inglaterra em 1200."

Objetos: John, Inglaterra, 1200.

Relações: reinou.

Propriedades: maldoso, rei.

Nesse exemplo, *rei* é considerado como uma propriedade de pessoa. Porém, se fosse mais apropriado, *rei* poderia ser uma relação entre pessoas e países ou, ainda, entre países e pessoas em um mundo onde cada país tem seu *rei*.

Os sistemas de aprendizado que utilizam esse tipo de linguagem de representação, são chamados de sistemas de aprendizado relacional. Nesses sistemas, objetos podem ser descritos estruturalmente, isto é, em termos de seus componentes e relações entre esses componentes. O uso de LPO como linguagem de representação em sistemas de aprendizado, permite que relações ou predicados possam ser induzidos. Isso faz com que o espaço dos

conceitos passíveis de serem aprendidos seja aumentado. Esses sistemas possuem uma alta expressividade para representar conceitos e a habilidade de representar conhecimento do domínio. Além da alta expressividade e do uso de conhecimento do domínio, os sistemas de aprendizado relacional têm a vantagem de expressarem seu conhecimento de uma forma diretamente inteligível aos humanos, característica muito importante quando o objetivo é a extração de conhecimento (Rezende et al., 2003; Muggleton, 1999).

Sistemas de aprendizado que induzem hipóteses na forma de programas lógicos, são chamados de sistemas de Programação Lógica Indutiva (PLI). Em PLI, os conceitos, bem como o conhecimento do domínio, podem ser descritos de forma extensional ou intensional. Um conceito é descrito de forma extensional listando a descrição de todas as suas instâncias. A descrição extensional de conceitos pode ser indesejável pois o conceito pode ter um número extremamente grande de instâncias. Conseqüentemente, é preferível a descrição intensional para descrever exemplos e conceitos, a qual permite fazer uma descrição de uma maneira bem mais compacta e concisa. Segue uma descrição da terminologia de programação lógica relevante para este trabalho.

Notação e Terminologia

Esta seção está baseada em (Lavrač & Džeroski, 1994) e (Lavrač & Džeroski, 2001).

Sintaxe Um *alfabeto* de primeira ordem consiste de variáveis, símbolos de predicados e símbolos de funções (que incluem constantes). Uma *variável* é representada por uma letra maiúscula, seguida por uma *string* de letras minúsculas e/ou dígitos.

Um *símbolo de função* é uma letra minúscula, seguida por uma *string* de letras minúsculas e/ou dígitos.

Uma variável é um *termo*. Um símbolo de função imediatamente seguido por uma n-upla de termos entre parênteses é um termo; assim, $f(g(A), h)$ é um termo quando f , g e h são símbolos de função e A é uma variável.

Uma *constante* é um símbolo de função de aridade 0, isto é, seguida por uma 0-upla de termos.

Um *símbolo de predicado*, imediatamente seguido por uma n-upla de termos, é chamado de *fórmula atômica* ou *átomo*.

Um *literal* é um átomo (literal positivo) ou um átomo negado (literal negativo).

Uma *fórmula bem formada* (ou *sentença*) é uma fórmula atômica ou uma fórmula que assume alguma das seguintes configurações: F , (F) , $\neg F$, $F \vee G$, $F \wedge G$, $F \longrightarrow G$,

$F \longleftrightarrow G$, $\forall A : F$ e $\exists A : F$, onde F e G são fórmulas bem formadas e A é uma variável.

Uma *cláusula* é uma disjunção de literais precedida por um quantificador universal para cada uma das variáveis que aparecem na disjunção. Uma cláusula tem a forma:

$$\forall A_1, \dots, \forall A_s (L_1 \vee L_2 \vee \dots \vee L_n)$$

em que cada L_i é um literal e $A_1, \dots, A_s$ são todas as variáveis que ocorrem na cláusula.

Uma cláusula também pode ser representada como um conjunto finito, possivelmente vazio de literais $\{L_1, L_2, \dots, L_n\}$. O conjunto $\{A_1, A_2, \dots, A_h, \neg B_1, \neg B_2, \dots, \neg B_b\}$, onde A_i e B_i são átomos, representa a cláusula $(A_1 \vee A_2 \vee \dots \vee A_h \vee \neg B_1 \vee \neg B_2 \vee \dots \vee \neg B_b)$, que escrevendo-a sob a forma de uma implicação é

$$A_1 \vee A_2 \vee \dots \vee A_m \leftarrow B_1 \wedge B_2 \wedge \dots \wedge B_n$$

sendo, a parte da cláusula correspondente a $A_1 \vee A_2 \vee \dots \vee A_m$ é denominada de *cabeça* da cláusula e $B_1 \wedge B_2 \wedge \dots \wedge B_n$, denominado *corpo* da cláusula. As vírgulas na cabeça da cláusula denotam disjunção e as do corpo conjunção.

Um conjunto de cláusulas é chamado de *teoria clausal* e representa a conjunção de suas cláusulas.

Literal, cláusulas e teoria clausal são fórmulas bem formadas.

Dado E , uma fórmula bem formada e $vars(E)$ que denota um conjunto de variáveis em E , E é dito ser *ground* (instanciado ou fechado), se e somente se, $vars(E) = \emptyset$.

Uma *cláusula de Horn* é uma cláusula que contém, no máximo, um literal positivo; ela é uma *cláusula definida* se contém exatamente um literal positivo. Um conjunto de cláusulas definidas é chamado de *programa lógico definido*. Um *fato* é uma cláusula definida com o corpo vazio. Uma *meta* é uma cláusula de Horn que não contém nenhum literal positivo.

Uma *cláusula de programa* é uma cláusula da forma

$$P \leftarrow L_1, \dots, L_n$$

na qual P é um átomo e cada $L_1, \dots, L_n$ é da forma L ou $\neg L$. Um *programa normal* (ou programa lógico) é um conjunto de cláusulas de programa.

Uma *definição de predicado* é um conjunto de cláusulas de programa que contém como cabeça o mesmo símbolo de predicado e aridade

Uma *substituição* é uma função que troca variáveis por termos em uma expressão. Por exemplo, a substituição $\{A/3, B/Z\}$ troca a variável A pelo termo 3 e troca a variável B pelo termo Z . Dada uma substituição θ e uma literal L , escreve-se $L\theta$ para indicar o resultado da aplicação da substituição θ em L .

Uma *substituição unificadora* de dois literais L_1 e L_2 é uma substituição θ tal que $L_1\theta = L_2\theta$.

Semântica A teoria de modelos preocupa-se com a atribuição do significado (valor verdade) às sentenças em uma linguagem de primeira ordem. Informalmente, uma sentença é mapeada para um dado assunto sobre um determinado domínio por meio de um processo conhecido como interpretação. Uma *interpretação* é determinada pelo conjunto de fatos *ground* para a qual atribui-se o valor verdade.

Uma interpretação que atribui o valor verdadeiro para uma determinada sentença é dita satisfazer a sentença e é chamada de *modelo* da sentença. Uma interpretação é um modelo para um conjunto de sentenças se e somente se ela é um modelo para cada uma das sentenças que pertencem a esse conjunto. Uma sentença é *satisfatível* se ela tem pelo menos um modelo, caso contrário é *insatisfatível*.

Uma sentença F *implica logicamente* uma sentença G , denotado por $F \models G$, se e somente se cada modelo de F for também modelo de G . Alternativamente, diz-se que G é uma *consequência lógica* de F . Por extensão, tem-se a noção de implicação lógica entre conjuntos de sentenças.

Uma *interpretação de Herbrand* sobre um alfabeto de primeira ordem é um conjunto de fatos *ground* construído com os símbolos de predicado no alfabeto e com os termos *ground* do domínio de Herbrand correspondente aos símbolos de função; esse é o conjunto de átomos *ground* considerados verdadeiros pela interpretação. Uma interpretação de Herbrand I é um modelo para uma cláusula c se e somente se c for verdadeiro em I . Uma interpretação de Herbrand I é um modelo para uma teoria clausal T se e somente se é um modelo para todas as cláusulas em T . Nesse caso diz-se que I é um *Modelo de Herbrand* de c , correspondentemente T .

O Modelo de Herbrand de um programa lógico é, informalmente, um conjunto de átomos fechados que validam logicamente cada cláusula do programa.

Um programa lógico P tem um conjunto de Modelos de Herbrand que se organizam em um reticulado segundo a relação de ordem parcial $\subseteq$ entre conjuntos. O elemento

mínimo desse reticulado é denominado *modelo mínimo de Herbrand* denotado por $M(P)$.

2.3.3 Representação Proposicional *versus* Relacional

Como mencionado, entre as diferentes dimensões que distinguem os sistemas de aprendizado simbólico, uma das mais significativas está relacionada com o poder de representação das linguagens para representar objetos ou exemplos, $\mathcal{L}_E$, conceitos ou hipóteses, $\mathcal{L}_h$, bem como conhecimento do domínio, $\mathcal{L}_K$. Embora a maioria dos indutores utilize a lógica de atributos para descrever exemplos e hipóteses, sua baixa capacidade de expressão impede a representação de objetos estruturados, assim como a relação entre objetos ou entre seus componentes. Assim, aspectos relevantes dos exemplos, que de alguma maneira poderiam caracterizar o conceito sendo aprendido, podem não ser representados.

Como mencionado, a linguagem de representação proposicional de exemplos, $\mathcal{L}_E$, mais freqüentemente utilizada, é a representação atributo-valor que utiliza uma única tabela para representar o conjunto de dados; cada exemplo, ou observação, corresponde a uma única tupla de uma única relação. Por outro lado, a representação relacional utiliza uma representação estrutural de primeira ordem. Conjuntos de exemplos correspondem a conjuntos de fatos e exemplos que podem consistir de múltiplas tuplas pertencentes a várias tabelas.

Com relação à linguagem de representação de conceitos ou hipóteses, $\mathcal{L}_h$, para muitas tarefas de aprendizado de conceitos, o poder de representação de linguagens proposicionais tem sido suficiente. Porém, quando os objetos são estruturados, consistindo de várias partes, é necessário enriquecer o formalismo representacional com variáveis para referenciar essas diversas partes (Flach, 2000). Sistemas que utilizam a representação relacional possuem um alto poder de expressividade. Porém, devido ao espaço de busca desses métodos ser muito maior, o custo computacional e a complexidade de tempo, no pior caso, torna-se muito maior que em sistemas que utilizam a representação proposicional (Lavrač & Džeroski, 2001).

2.4 Considerações Finais

O AM é uma área da IA cujo objetivo é projetar e desenvolver sistemas capazes de adquirir conhecimento de forma automática. Os diversos sistemas de AM possuem características que possibilitam sua classificação quanto ao paradigma, modo de aprendizado, forma de

aprendizado e linguagem de descrição utilizada. Como visto neste capítulo, existem várias abordagens de aprendizado que podem ser utilizadas por um sistema computacional, entre essas abordagens encontra-se o aprendizado por indução. Vários formalismos vêm sendo utilizados nos sistemas de aprendizado indutivo para descrever exemplos e conceitos induzidos. Em geral, distingue-se dois tipos de descrição: descrição baseada em atributos ou proposicional e descrição relacional.

Alguns algoritmos amplamente utilizados pela comunidade, empregam como linguagem de descrição de objetos e conceitos a linguagem proposicional. Apesar do relativo sucesso desses sistemas proposicionais de aprendizado, eles são fortemente limitados devido a pouca expressividade do formalismo representacional e a capacidade limitada de utilizar conhecimento do domínio. Por outro lado, a descrição relacional é baseada em LPO, o que permite a descrição de objetos e relações entre os objetos. Sistemas de aprendizado que utilizam esse tipo de linguagem de representação são chamados de sistema de aprendizado relacional. Esses sistemas possuem uma alta expressividade para representar conceitos e a habilidade de representar conhecimento do domínio. Dentre os sistemas de aprendizado relacional, destacam-se os sistemas de PLI, foco deste trabalho, descritos com maiores detalhes no próximo capítulo.

Programação Lógica Indutiva

A Programação Lógica Indutiva é uma área de intersecção de pesquisas entre Aprendizado de Máquina e programação lógica. O objetivo da PLI é aprender programas lógicos, a partir de exemplos e conhecimento do domínio. Neste capítulo são apresentados conceitos relevantes da PLI, área na qual este trabalho se insere. São apresentados alguns conceitos básicos, como é realizado o processo de aprendizado e qual a importância do *bias* para esse processo de aprendizado.

3.1 Introdução à PLI

PLI foi definida inicialmente como a intersecção de AM e programação lógica (Muggleton, 1991). Muggleton esclarece que se a linguagem de hipóteses ($\mathcal{L}_h$) em uma tarefa de aprendizado é a linguagem de programas lógicos, então o aprendizado é, de fato, uma síntese da programação lógica e recebe o nome de Programação Lógica Indutiva. Russel & Norvig (2003) definem PLI como a combinação de métodos indutivos com o poder de representação da Linguagem de Primeira Ordem, concentrando-se em particular sobre a representação de teorias com programas lógicos. Atualmente, a definição mais utilizada considera a PLI como a intersecção de AM indutivo e programação lógica (Lavrač & Flach, 2001) — Figura 3.1.

Figura 3.1: Intersecção de aprendizado de máquina e programação lógica - PLI

Da lógica computacional, PLI herda seu formalismo representacional, várias técnicas bem estabelecidas e uma profunda base teórica. Do AM indutivo, herda uma abordagem experimental e orientação para aplicações práticas, tais como o desenvolvimento de ferramentas e técnicas para induzir hipóteses a partir de exemplos e sintetizar novos conhecimentos a partir de experiências (Muggleton & Raedt, 1994). A área de pesquisa em PLI também tem sido fortemente influenciada pela teoria de aprendizado computacional e, recentemente, pela descoberta de conhecimento em bases de dados, o que tem conduzido para o desenvolvimento de novas técnicas para mineração de dados relacional (Lavrač & Flach, 2001).

A PLI se diferencia da maioria dos outros modos de AM pelo uso de uma expressiva linguagem de representação e sua habilidade para utilizar conhecimento do domínio. O conhecimento do domínio cumpre um papel importantíssimo para o aprendizado, no qual a tarefa é encontrar, a partir de exemplos observados, uma relação desconhecida (predicado alvo), em termos de relações já conhecidas de conhecimento do domínio. Por exemplo, se um aprendiz não tem conhecimento prévio sobre o problema de aprendizado, ele vai aprender exclusivamente dos exemplos. Porém, para problemas de aprendizado que são difíceis, o aprendiz tipicamente requer uma quantidade substancial de conhecimento do domínio para aprender. Usando o conhecimento do domínio, o aprendiz pode encontrar uma generalização dos exemplos observados de uma forma mais natural e concisa.

O conhecimento do domínio utilizado na construção de hipóteses é uma característica importante em PLI. Quando o conhecimento do domínio é relevante, pode-se melhorar substancialmente os resultados do aprendizado em termos de precisão, eficiência e do potencial explanatório do conhecimento induzido. Conseqüentemente, conhecimento de fundo irrelevante terá o efeito contrário. Muito da arte da PLI está na seleção e formulação apropriada do conhecimento de fundo para ser utilizado na tarefa de aprendizado (Lavrač, 1998).

3.2 O Problema da PLI

O problema básico da PLI consiste do aprendizado de definições lógicas de relações, no qual tuplas que pertencem, ou não, à relação alvo, são dadas como exemplos. Dos exemplos dados, a PLI induz um programa lógico (definição de predicado) correspondendo a uma teoria, que define a relação alvo, em termos de outras relações que são dadas como conhecimento do domínio.

De uma maneira geral, PLI pode ser descrita a partir de uma teoria de conhecimento do domínio inicial $\mathcal{K}$ e algum conjunto de exemplos $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$, onde $\mathcal{E}^+$ representa os exemplos positivos e $\mathcal{E}^-$ os exemplos negativos do conceito a ser aprendido. O objetivo da PLI consiste em induzir uma hipótese h que junto com $\mathcal{K}$ *explica* os exemplos $\mathcal{E}$. No caso geral, h , $\mathcal{K}$ e $\mathcal{E}$ podem ser qualquer conjunto de cláusulas. Entretanto, na maioria dos problemas, o conhecimento do domínio, os exemplos e a hipótese induzida devem satisfazer um conjunto de restrições sintáticas $\mathcal{S}$, chamado de *bias* da linguagem. Esse *bias* define o espaço de fórmulas bem formadas usadas para representar hipóteses e pode ser considerado como parte do conhecimento prévio.

O aprendizado empírico de um único conceito (predicado) em PLI pode ser formulado da seguinte maneira (Muggleton, 1991):

Dados:

- um conjunto de exemplos de treinamento $\mathcal{E}$ descritos em uma linguagem $\mathcal{L}_{\mathcal{E}}$ e consistindo de
 - exemplos positivos, $\mathcal{E}^+$ e
 - exemplos negativos, $\mathcal{E}^-$;
- um predicado desconhecido p , especificando a relação a ser aprendida (relação *alvo* ou *meta*);
- uma linguagem de descrição de hipóteses, $\mathcal{L}_h$, especificando as restrições sintáticas na definição do predicado p ;
- um *bias* $\mathcal{S}$ de linguagem, que define o espaço de hipóteses;
- a teoria do domínio $\mathcal{K}$, descrita em uma linguagem $\mathcal{L}_{\mathcal{K}}$, definindo predicados q_i (outros que p) os quais podem ser usados na definição de p e que podem fornecer informação adicional sobre argumentos dos exemplos da relação alvo, e

- um operador entre $\mathcal{L}_{\mathcal{E}}$ e $\mathcal{L}_{\mathbf{h}}$ com relação a $\mathcal{L}_{\mathcal{K}}$ que determina se um exemplo é coberto por uma cláusula expressa em $\mathcal{L}_{\mathbf{h}}$.

Encontrar:

Uma definição $\mathbf{h}$ para p , expressa em $\mathcal{L}_{\mathbf{h}}$, tal que, $\mathcal{K} \wedge \mathbf{h} \models \mathcal{E}^+$ e $\mathcal{K} \wedge \mathbf{h} \not\models \mathcal{E}^-$.

É importante observar que o objetivo da PLI é a indução de hipóteses na forma simbólica explícita para que possam ser facilmente interpretadas pelo usuário/especialista. Assim, o conhecimento induzido pode melhorar o entendimento do problema.

Um dos problemas clássicos da PLI visa o aprendizado de descrições de conceitos no formato de regras de classificação, o qual é chamado de PLI preditiva¹. Outro problema básico da PLI é a PLI descritiva², cujo objetivo, geralmente, é o aprendizado de uma teoria clausal (Lavrač & Džeroski, 2001).

3.2.1 PLI Preditiva

A PLI preditiva é uma das tarefas mais comuns em PLI e tem como objetivo o aprendizado de regras de classificação e regressão. Essa tarefa de PLI tipicamente restringe $\mathcal{E}$ a fatos *ground*, bem como restringe $\mathbf{h}$ e $\mathcal{K}$ a conjuntos de cláusulas definidas. A noção de explicação nesse tipo de tarefa geralmente é denotada pela cobertura e requer completude e consistência global das hipóteses (Lavrač, 1998). A noção de completude e consistência de uma hipótese é abordada com maiores detalhes na Seção 3.2.4 na página 25.

Formalmente, o problema da PLI preditiva é definido como:

Dado o conhecimento do domínio $\mathcal{K}$, hipóteses $\mathbf{h}$ e um conjunto de exemplos $\mathcal{E}$; um exemplo $e \in \mathcal{E}$ é coberto por $\mathbf{h}$ se $\mathcal{K} \cup \mathbf{h} \models e$.

A hipótese é completa se $\forall e \in \mathcal{E}^+ : \mathcal{K} \cup \mathbf{h} \models e$.

A hipótese é consistente se $\forall e \in \mathcal{E}^- : \mathcal{K} \cup \mathbf{h} \not\models e$.

Dada a restrição para a teoria T definida como $T = \mathcal{K} \cup \mathbf{h}$, para a qual existe um único *Modelo Mínimo de Herbrand* $M(T)$ e átomos *ground* como exemplos, isso é equivalente a exigir que todos os exemplos em $\mathcal{E}^+$ sejam verdadeiros em $M(\mathcal{K} \cup \mathbf{h})$.

Para permitir teorias incompletas e inconsistentes, que satisfaçam outros critérios de aceitação, tais como precisão preditiva, significância e compressão, os problemas podem ser estendidos para incluir aprendizado de regras de classificação de dados imperfeitos,

¹também referenciada como PLI normal, PLI forte entre outros.

²também referenciada como PLI fraca, PLI aprendizado de descrições, PLI não-monotônica entre outros.

bem como o aprendizado de árvores de decisão. Neste trabalho utilizamos PLI preditiva.

3.2.2 PLI Descritiva

Na tarefa de PLI descritiva o objetivo, geralmente, é o aprendizado de uma teoria clausal. PLI descritiva, tipicamente, restringe $\mathcal{K}$ a um conjunto de cláusulas definidas³, $\mathbf{h}$ a um conjunto de cláusulas e $\mathcal{E}$ a exemplos positivos.

A rigorosa noção de explicação usada nesse conjunto de problemas, exige que, todas as cláusulas c em $\mathbf{h}$ sejam verdadeiras em algum modelo predefinido de $T = \mathcal{K} \cup \mathcal{E}$, no qual o modelo predefinido de T pode ser, por exemplo, o *Modelo Mínimo de Herbrand* $M(T)$. Relaxando a noção de explicação utilizada na descoberta clausal e permitindo teorias que satisfaçam alguns outros critérios de aceitação, por exemplo similaridade, associatividade e interessabilidade, a PLI preditiva pode ser estendida para incorporar o aprendizado de regras de associação e *clustering* de primeira ordem entre outros (Lavrač, 1998).

3.2.3 Exemplo de PLI Preditiva e Descritiva

Considere o seguinte exemplo de aprendizado de relações familiares descrito em (Lavrač & Džeroski, 2001), no qual o problema é o aprendizado de uma relação *filha*(P_1, P_2), em que uma pessoa P_1 é filha de uma pessoa P_2 , em termos de relações definidas de conhecimento do domínio. Essas relações, e o conjunto de exemplos de treinamento $\mathcal{E}$ para o predicado alvo *filha*/2, são mostrados na Tabela 3.1. Nessa tabela há dois exemplos positivos e dois negativos da relação alvo, indicados, respectivamente, com os símbolos $\oplus$ e $\ominus$. Há, ainda, algum conhecimento do domínio declarado de forma intensional⁴ e extensional⁵.

Na linguagem de hipóteses de programas lógicos, um sistema de PLI preditiva poderia induzir a seguinte cláusula da relação (conceito), *filha*/2 desejada, dados $\mathcal{E}^+$, $\mathcal{E}^-$ e $\mathcal{K}$:

$$filha(P_1, P_2) \leftarrow mulher(P_1), progenitor(P_2, P_1).$$

Dependendo do conhecimento do domínio, da linguagem $\mathcal{L}_{\mathbf{h}}$ e da complexidade do conceito procurado, a definição da relação alvo pode consistir de um conjunto de cláusulas, tais como:

³definite clause.

⁴pode conter tanto fatos *ground* quanto cláusulas com variáveis.

⁵pode conter apenas fatos *ground*.

$$filha(P_1, P_2) \leftarrow mulher(P_1), mae(P_2, P_1).$$

$$filha(P_1, P_2) \leftarrow mulher(P_1), pai(P_2, P_1).$$

Exemplos de Treinamento	Conhecimento do Domínio		
$\oplus filha(maria, ana).$	$mae(ana, maria).$	$progenitor(P_1, P_2) \leftarrow$	$mulher(ana).$
$\oplus filha(eva, tomas).$	$mae(ana, tomas).$	$mae(P_1, P_2).$	$mulher(maria).$
$\ominus filha(tomas, ana).$	$pai(tomas, eva).$	$progenitor(P_1, P_2) \leftarrow$	$mulher(eva).$
$\ominus filha(eva, ana).$	$pai(tomas, carol).$	$pai(P_1, P_2).$	$mulher(carol).$
			$homem(tomas).$

Tabela 3.1: Exemplos de treinamento e conhecimento do domínio para aprendizado relacional

Em um problema de PLI descritiva, são dados somente $\mathcal{E}^+$ e $\mathcal{K}$. Assim, uma teoria induzida poderia conter as seguintes cláusulas:

$$\leftarrow filha(P_1, P_2), mae(P_1, P_2).$$

$$mulher(P_1) \leftarrow filha(P_1, P_2).$$

$$mae(P_1, P_2); pai(P_1, P_2) \leftarrow progenitor(P_1, P_2).$$

Ou seja, no problema de descoberta de conhecimento preditivo, regras de classificação são geradas, enquanto que no problema descritivo apenas propriedades verdadeiras do conjunto de exemplos são obtidas. Além disso, na PLI preditiva, a hipótese induzida pode ser utilizada para substituir os exemplos, uma vez que o conhecimento do domínio e a hipótese implicam os exemplos observados (Muggleton & Raedt, 1994).

3.2.4 Completude e Consistência de uma Hipótese

Após ter selecionado as linguagens de descrição de objetos (exemplos) e conceitos, é necessário que se estabeleça se um dado objeto pertence a um determinado conceito, ou seja, se a descrição do objeto satisfaz a descrição do conceito. Quando essa condição é satisfeita, diz-se que, a descrição do conceito *cobre* a descrição do objeto, ou que a descrição do objeto é *coberta* pela descrição do conceito. O problema do aprendizado de um único conceito $\mathcal{C}$, por meio de exemplos, poderia ser definido como:

Dado um conjunto $\mathcal{E}$, de exemplos positivos e negativos de um conceito $\mathcal{C}$, encontrar uma hipótese $\mathbf{h}$, descrita em uma dada linguagem de descrição de conceitos $\mathcal{L}_{\mathbf{h}}$, tal que:

- todo exemplo positivo $e \in \mathcal{E}^+$ é coberto por $\mathbf{h}$, e

- nenhum exemplo negativo $e \in \mathcal{E}^-$ é coberto por $\mathbf{h}$;

Para o teste de cobertura, uma função $cobre(\mathbf{h}, e)$ pode ser definida. Essa função retorna o valor *verdade* se e é coberto por $\mathbf{h}$, e *falso* caso contrário. Essa função, simplesmente testa se e satisfaz qualquer das disjunções em $\mathbf{h}$. A função pode ser redefinida para conjuntos de exemplos da seguinte forma:

$$cobre(\mathbf{h}, \mathcal{E}) = \{e \in \mathcal{E} \mid cobre(\mathbf{h}, e) = verdade\}$$

Uma hipótese $\mathbf{h}$ é completa em relação aos exemplos $\mathcal{E}$ se ela cobre todos os exemplos positivos, isto é, $cobre(\mathbf{h}, \mathcal{E}^+) = \mathcal{E}^+$.

Uma hipótese $\mathbf{h}$ é consistente em relação aos exemplos $\mathcal{E}$ se ela não cobre nenhum exemplo negativo, isto é, $cobre(\mathbf{h}, \mathcal{E}^-) = \emptyset$.

Quatro situações podem ocorrer, dependendo de como a hipótese $\mathbf{h}$ cobre os exemplos positivos e negativos, como mostra a Figura 3.2 (Lavrač & Džeroski, 1994), na qual:

- (a) $\mathbf{h}$ completa e consistente, cobre todos os exemplos positivos e nenhum exemplo negativo;
- (b) $\mathbf{h}$ incompleta e consistente, não cobre todos os exemplos positivos e não cobre exemplos negativos;
- (c) $\mathbf{h}$ completa e inconsistente, cobre todos os exemplos positivos e cobre alguns exemplos negativos e
- (d) $\mathbf{h}$ incompleta e inconsistente, não cobre todos os exemplos positivos e cobre alguns exemplos negativos.

A função $cobre$ pode ser redefinida para considerar também o conhecimento do domínio $\mathcal{K}$, da seguinte maneira:

$$cobre(\mathbf{h}, \mathcal{K}, \mathcal{E}) = cobre(\mathbf{h} \cup \mathcal{K}, \mathcal{E}).$$

Quando é considerado o conhecimento do domínio, a completude e consistência também precisam ser redefinidas, como mostrado a seguir.

Uma hipótese $\mathbf{h}$ é completa, em relação ao conhecimento do domínio $\mathcal{K}$ e aos exemplos $\mathcal{E}$, se todos os exemplos positivos são cobertos, isto é, se $cobre(\mathbf{h}, \mathcal{K}, \mathcal{E}^+) = \mathcal{E}^+$.

Uma hipótese $\mathbf{h}$ é consistente, em relação ao conhecimento do domínio $\mathcal{K}$ e aos exemplos $\mathcal{E}$, se nenhum exemplo negativo é coberto, isto é, se $cobre(\mathbf{h}, \mathcal{K}, \mathcal{E}^-) = \emptyset$.

(a) h : completa, consistente

(b) h : incompleta, consistente

(c) h : completa, inconsistente

(d) h : incompleta, inconsistente

Figura 3.2: Completude e consistência de uma hipótese (Lavrač & Džeroski, 1994)

3.3 *Bias*

No aprendizado relacional, a linguagem utilizada para descrever exemplos, conceitos e conhecimento do domínio é, tipicamente, um subconjunto da LPO composto por cláusulas de Horn (Cohen, 1993). Ainda que o uso de cláusulas de Horn restringe o espaço de hipóteses da LPO, essa linguagem é bastante expressiva, e é necessário utilizar alguma forma de *bias* para restringir ainda mais o espaço de hipóteses expressas com cláusulas de Horn.

Muggleton & Raedt (1994) distinguem dois tipos de *bias*: *bias* sintático (*bias* de linguagem) e *bias* semântico. O *bias* sintático impõe restrições na forma (sintaxe) das cláusulas permitidas nas hipóteses. O *bias* semântico impõe restrições no significado ou no comportamento das hipóteses.

Segundo Nédellec, Rouveirol, Adé, Bergadano, & Tausend (1996), exceto para os exemplos e contra exemplos apresentados do conceito sendo aprendido, todos os fatores que influenciam na seleção de hipóteses constituem o *bias*. Esses fatores incluem:

- a linguagem com a qual são descritas as hipóteses;
- o espaço de hipóteses que o programa pode considerar;
- o procedimento que define em qual ordem as hipóteses serão consideradas;
- o critério de aceitação que define se um procedimento de busca pode parar com uma dada hipótese ou se deveria continuar a busca por uma escolha melhor.

Segundo Nédellec et al. (1996); Nienhuys-Cheng & Wolf (1997), essa noção de *bias* pode ser dividida em três categorias distintas: *bias* de linguagem, *bias* de busca e *bias* de validação. O *bias* de linguagem se refere a restrições nas cláusulas contidas no espaço de busca. Quanto mais restrições forem impostas às cláusulas, menor será o espaço de busca e, conseqüentemente, mais rápido o sistema finaliza a busca pela hipótese *h*. Por outro lado, restrições podem fazer com que boas hipóteses sejam descartadas. O *bias* deve, então, ser fraco o suficiente para permitir programas completos e consistentes, e forte o suficiente para prevenir a total degradação da performance. O *bias* de busca se refere a maneira como um sistema realiza a busca pelo espaço de cláusulas permitidas. Em um extremo está a busca exaustiva, a qual percorre o espaço de busca completamente, no outro as buscas heurísticas, que indicam quais partes do espaço de busca serão percorridas e quais serão ignoradas. Finalmente, o *bias* de validação se refere ao critério de parada do sistema. Um critério pode ser parar a busca quando uma hipótese correta é encontrada.

Por outro lado, é preciso considerar a existência de ruído nos dados, nesse caso a busca pode parar quando a restrição sobre a corretude da hipótese é relaxada.

3.4 Estruturando o Espaço de Cláusulas

A maioria dos métodos disponíveis de aprendizado de conceitos podem ser vistos como métodos de busca (Mitchell, 1982). Para que uma busca seja efetiva, é fundamental que ela seja conduzida de maneira sistemática. No caso particular de PLI, para sistematizar a busca no espaço de hipóteses é importante que esse espaço seja estruturado por meio do estabelecimento de uma ordenação de seus elementos. A relação subsunção- θ , aqui descrita, permite fazer isso.

Lembrando que uma *substituição* $\theta = V_1/t_1, \dots, V_n/t_n$ é uma função de variáveis V_i para termos t_i , a aplicação $W\theta$ de uma substituição θ a uma fórmula W é obtida trocando todas as ocorrências de uma variável V_i em W pelo termo t_i correspondente. A seguir tem-se um exemplo de substituição na qual, dada a cláusula c :

$$c = filha(P_1, P_2) \leftarrow progenitor(P_2, P_1)$$

a *substituição* $\theta = P_1/maria, P_2/ana$ aplicada à cláusula c é obtida pela aplicação de θ para cada um dos literais:

$$c\theta = filha(maria, ana) \leftarrow progenitor(ana, maria)$$

A *subsunção- θ* é definida da seguinte forma: uma cláusula $c_1\theta$ *subsume* uma cláusula c_2 , se e somente se, existe uma substituição θ tal que $c_1\theta \subseteq c_2$. Ou seja, c_1 é uma generalização de c_2 , e c_2 uma especialização de c_1 (Muggleton & Raedt, 1994).

Por exemplo:

$$\begin{aligned} pai(P_1, P_2) \leftarrow progenitor(P_1, P_2), homem(P_1), \\ \theta\text{-subsume} \\ pai(joao, paulo) \leftarrow progenitor(joao, paulo), progenitor(joao, ana), homem(joao), \\ mulher(ana) \\ \text{com } \theta = \{P_1 = joao, P_2 = ana\}. \end{aligned}$$

A subsunção- θ introduz uma noção sintática de generalidade que estrutura o espaço de hipóteses e ajuda a eliminar buscas em grandes partes do espaço de busca pois:

- generalizações de uma cláusula que inclui um exemplo negativo também o incluem, logo não precisam ser testadas, e
- especializações de uma cláusula que não cobre um exemplo positivo também não irão incluí-lo e, assim, também não precisam ser testadas.

Subsunção- θ possui duas propriedades importantes:

1. Se $c_1 \theta$ *subsume* c_2 então c_1 também implica c_2 , ou seja, $c_1 \models c_2$. O inverso nem sempre é verdade.
2. Introduz um *reticulado* no conjunto de todas as cláusulas. Isso quer dizer que duas cláusulas nesse conjunto têm sempre uma cláusula que serve como limite superior mínimo delas e outra que funciona como limite inferior máximo.

Com o espaço de hipóteses estruturado pela introdução de uma ordem parcial, buscas nesse espaço tornam-se factíveis e podem ser realizadas sistematicamente, de maneira *bottom-up* (métodos de generalização) ou *top-down* (métodos de especialização), descritos a seguir.

3.5 Métodos de Generalização

Métodos que fazem uma busca no espaço de hipóteses de maneira *bottom-up* são conhecidos como métodos de generalização. Os métodos de generalização realizam a busca no espaço de hipóteses, iniciando a busca a partir dos exemplos de treinamento (hipóteses mais específicas) e generalizando os exemplos por meio de operadores de generalização. Os métodos de generalização mais utilizados em PLI são: a *generalização menos geral*, a *generalização menos geral relativa* e a *resolução inversa*, apresentados a seguir.

Generalização Menos Geral (*lgg*)

A noção de *generalização menos geral* – *lgg*⁶ – apresentada primeiro em (Plotkin, 1970), é importante para PLI pois forma a base da generalização “cautelosa”. Esse tipo de generalização assume que se duas cláusulas c_1 e c_2 são verdadeiras, é bem provável que sua generalização mais específica também seja verdadeira. Define-se a *lgg* de duas cláusulas

⁶do inglês, *least general generalization*; será mantida essa notação por ser mais comumente utilizada pela comunidade de PLI.

c_1 e c_2 como sendo o mínimo limite superior de c_1 e c_2 no reticulado introduzido por subsunção- θ .

É possível implementar a *lgg* de termos, literais e cláusulas. Exemplos de *lgg* de termos são mostrados na Tabela 3.2.

T_1	T_2	$lgg(T_1, T_2)$
$f(r, s, t, u)$	$f(v, x, y, z)$	$f(A, B, C, D)$
$f(r, s, t, s)$	$f(v, x, x, x)$	$f(A, B, C, B)$
$f(r, s, t, u)$	$g(r, s, t, u)$	A
$[1, 2, 3]$	$[1, 4, 5]$	$[1, A, B]$
$[r, s, t, u]$	$[r, t, s]$	$[r, A, B C]$

Tabela 3.2: Exemplos de *lgg* de termos

A *lgg* de dois literais é calculada comparando os termos que estão na mesma posição em cada literal. Se forem iguais, o valor do termo é mantido na *lgg*; se não, são substituídos por uma variável. Exemplos de *lggs* de alguns literais são mostrados na Tabela 3.3.

L_1	L_2	$lgg(L_1, L_2)$
$p(r, s)$	$p(v, x)$	$p(A, B)$
$\neg p(r, s)$	$p(r, s)$	indefinido
$\text{append}([1,2], [3,4], [1,2,3,4])$	$\text{append}([r], [s], [r, s])$	$\text{append}([A B], [C D], [A, E F])$

Tabela 3.3: Exemplos de *lgg* de literais

Pode ser observado que a *lgg* não leva em consideração o conhecimento do domínio. Plotkin estendeu o conceito de *lgg* para a *generalização menos geral relativa*, na qual o conhecimento do domínio é considerado (Plotkin, 1971).

Generalização Menos Geral Relativa (*rlgg*)

A *generalização menos geral relativa*—*rlgg*⁷—de duas cláusulas c_1 e c_2 é a generalização menos geral, $lgg(c_1, c_2)$, em relação ao conhecimento do domínio $\mathcal{K}$. Em outras palavras, se o conhecimento do domínio consiste de átomos *ground* e $\mathcal{K}$ denota a conjunção de todos esses fatos, a *rlgg* de dois exemplos de treinamento positivos e_1 e e_2 , relativo a $\mathcal{K}$ é definido como:

$$rlgg(e_1, e_2) = lgg((e_1 \leftarrow K), (e_2 \leftarrow K))$$

Um sistema de PLI que utiliza essa técnica como base para realizar o aprendizado é o GOLEM (Muggleton & Feng, 1990).

⁷do inglês, *relative least general generalization*; novamente essa notação será mantida por ser mais comumente utilizada pela comunidade de PLI.

Resolução Inversa

Outro método de generalização é a *Resolução Inversa* — *ires*⁸. A idéia básica da resolução inversa, introduzida como uma técnica de generalização para PLI por Muggleton & Buntine (1988), é inverter a regra de *resolução* da inferência dedutiva (Robinson, 1965). Um passo básico de resolução aplicado em lógica proposicional deriva a proposição $p \vee r$ dadas as premissas $p \vee \neg q$ e $q \vee r$. Em lógica de primeira ordem, resolução é mais complicada, pois envolve substituições. A conclusão obtida de duas cláusulas c_1 e c_2 por meio de um passo de inferência por resolução é denotada por $res(c_1, c_2)$ e é chamado de *resolvente* de c_1 e c_2 .

A resolução Inversa, como foi implementada em Muggleton & Buntine (1988), utiliza um operador de generalização baseado no inverso de substituição (Buntine, 1988). Dada uma cláusula W , uma *substituição inversa* θ^{-1} de uma substituição θ é uma função que mapeia termos em $W\theta$ para variáveis, de modo que $W\theta\theta^{-1} = W$.

Por exemplo, dada a cláusula

$$c = filha(P_1, P_2) \leftarrow mulher(P_1), progenitor(P_2, P_1).$$

e a substituição $\theta = \{P_1/sara, P_2/eva\}$:

$$c' = c\theta = filha(sara, eva) \leftarrow mulher(eva), progenitor(eva, sara).$$

aplicando a substituição inversa $\theta^{-1} = \{sara/P_1, eva/P_2\}$, é possível obter a cláusula original:

$$c = c'\theta^{-1} = filha(P_1, P_2) \leftarrow mulher(P_1), progenitor(P_2, P_1).$$

No caso geral, a substituição inversa se torna mais complexa, pois ela envolve as posições dos termos para assegurar que as variáveis da cláusula inicial W sejam restauradas a seus lugares apropriados em $W\theta\theta^{-1}$.

Por exemplo, considere o conhecimento do domínio $\mathcal{K}$ dado pelos dois fatos (cláusulas)

$$b_1 = mulher(sara) \text{ e } b_2 = progenitor(eva, sara).$$

Seja a hipótese $\mathbf{h} = \emptyset$ e suponha que o sistema de aprendizado encontre o exemplo positivo $e_1 = filha(sara, eva)$. O processo de resolução inversa poderia ser:

⁸do inglês, *inverse resolution*.

- Primeiro, procura-se uma cláusula c_1 que, junto com b_2 , implique e_1 e possa ser adicionada à hipótese atual $\mathbf{h}$ no lugar de e_1 . Por meio da substituição inversa $\theta_2^{-1} = \{eva/P_2\}$, um passo de resolução inversa gera a cláusula

$$c_1 = ires(b_2, e_1) = filha(sara, P_2) \leftarrow progenitor(P_2, sara).$$

Com isso, a cláusula c_1 se torna a hipótese atual $\mathbf{h}$, tal que $\{b_2\} \cup \mathbf{h} \models e_1$.

- Num passo seguinte, a resolução inversa poderia considerar $b_1 = mulher(sara)$ e a hipótese atual

$$\mathbf{h} = \{c_1\} = \{filha(sara, P_2) \leftarrow progenitor(P_2, sara)\}.$$

Calculando c_2 , com a substituição inversa $\theta_1^{-1} = \{sara/P_1\}$, a cláusula pode ser generalizada com relação ao conhecimento do domínio $\mathcal{K}$, resultando na cláusula

$$c_2 = filha(P_1, P_2) \leftarrow mulher(P_1), progenitor(P_2, P_1).$$

Assim, c_1 pode ser substituída na hipótese atual $\mathbf{h}$ pela cláusula mais geral c_2 que junto com $\mathcal{K}$ implica o exemplo e_1 . A hipótese induzida é

$$\mathbf{h} = \{filha(P_1, P_2) \leftarrow mulher(P_1), progenitor(P_2, P_1)\}$$

O operador de generalização mostrado no exemplo da Figura 3.3 na página seguinte (Lavrač & Džeroski, 2001) é chamado de operador de *absorção* (ou operador **V**). O exemplo mostra a árvore de derivação linear inversa correspondente a esse processo.

3.6 Métodos de Especialização

Métodos de especialização executam a busca no espaço de hipóteses de uma maneira *top-down*, das hipóteses gerais para as específicas, usando um operador de especialização baseado na subsunção- θ . O método básico de especialização em PLI é a *busca top-down em grafos de refinamento*. Esse método foi introduzida no sistema de PLI interativo MIS — Model Inference System (Shapiro, 1983). Também é utilizada no sistema empírico FOIL (Quinlan, 1990), e seus derivados, como mFOIL (Džeroski & Bratko, 1992) e FOCL (Pazzani & Kibler, 1991).

Figura 3.3: Uma árvore de derivação linear inversa

Sistemas de aprendizado *top-down* começam com a cláusula mais geral e a refinam repetidamente até que ela não cubra mais exemplos negativos. Durante o processo de aprendizado, os sistemas asseguram que a cláusula considerada cobre pelo menos um exemplo positivo.

Dada uma linguagem de hipóteses $\mathcal{L}_h$ e conhecimento do domínio $\mathcal{K}$, o espaço de hipóteses de cláusulas de programa forma um reticulado, estruturado por meio da ordenação de generalidade da relação θ – *subsume*. Nesse reticulado, um *grafo de refinamento* pode ser definido como um grafo acíclico e dirigido no qual *nós* são cláusulas de programa e *arcos* correspondem às operações básicas de refinamento: substituir uma variável por um termo e adicionar um literal ao corpo de uma cláusula.

Para a construção do grafo de refinamento para o problema de relações familiares, apresentado na Seção 3.2 na página 22, a busca pela cláusula é iniciada com a cláusula $filha(X, Y) \leftarrow$, a qual cobre dois exemplos negativos. Os refinamentos da cláusula são considerados em seguida, dos quais $filha(X, Y) \leftarrow mulher(X)$ e $filha(X, Y) \leftarrow pai(Y, X)$ cobrem apenas um exemplo negativo. As duas cláusulas tem um refinamento em comum

$$filha(X, Y) \leftarrow mulher(X), pai(Y, X)$$

que não cobre os exemplos negativos, terminando assim a busca. Parte do grafo de

refinamento referente ao problema das relações familiares é apresentado na Figura 3.4.

Figura 3.4: Parte do grafo de refinamento gerado no aprendizado filha/2 (Lavrač & Džeroski, 1994)

Um grafo de refinamento é tipicamente percorrido heurísticamente, usando heurísticas baseadas em números de exemplos positivos e negativos cobertos pela cláusula. Algumas das técnicas de busca que podem ser empregadas são *hill-climbing*, *A* best-first* e *breadth-first* (Lavrač & Džeroski, 2001).

3.7 Considerações Finais

A PLI é uma área de pesquisa de intersecção de AM indutivo e programação lógica que investiga a construção indutiva de teorias clausal de primeira ordem, a partir de exemplos e conhecimento do domínio. Um dos problemas clássicos da PLI visa o aprendizado de descrições de conceitos no formato de regras de classificação, a qual é chamada de PLI preditiva, usada no desenvolvimento deste trabalho. Outro problema básico em PLI é o aprendizado de teoria clausal, tarefa da PLI descritiva. Após o processo de aprendizado, tanto em PLI preditiva como na descritiva, é induzida uma hipótese (teoria) h .

Entretanto, o espaço das possíveis hipóteses h neste tipo de aprendizado relacional é muito grande. Assim, para que uma busca nesse espaço seja efetiva é fundamental que

Figura 3.5: Como os conceitos para buscas no espaço de hipóteses se relacionam

ela seja conduzida de maneira sistemática. No caso particular da PLI, para sistematizar a busca no espaço de hipóteses é importante que esse espaço seja estruturado por meio do estabelecimento de uma ordenação parcial dos seus elementos. A relação subsunção- θ , descrita permite fazer isso. Com o espaço de hipóteses estruturado pela introdução de uma ordem parcial, buscas nesse espaço tornam-se factíveis e podem ser realizadas de maneira *bottom-up* (métodos de generalização) ou *top-down* (métodos de especialização). Nos métodos de generalização as buscas são realizadas utilizando os métodos *lgg*, *rlgg* e *resolução inversa*. Nos métodos de especialização a busca pode ser realizada utilizando o método da *busca top-down em grafos de refinamento*. Na Figura 3.5 é apresentado um esquema de como esses conceitos estão relacionados.

Sistemas de Programação Lógica Indutiva

Os sistemas de PLI situam-se em duas grandes sub-áreas. Em uma delas estão os *sistemas interativos* de PLI, os quais tem a característica de aprenderem múltiplos predicados e estão fortemente relacionados aos sistemas revisores de teoria. Esses sistemas aprendem definições de múltiplos predicados a partir de um conjunto pequeno de exemplos e consultas ao usuário. Na segunda sub-área de PLI estão os chamados *sistemas empíricos* de PLI; são sistemas não-interativos de aprendizado não-incremental, que aprendem um único predicado a partir do zero. Nesses sistemas a ênfase é a extração de padrões de um grande número de exemplos, os quais podem conter ruído. Neste capítulo são apresentadas as características que distinguem os sistemas de PLI, uma breve descrição de alguns sistemas além de uma descrição detalhada do sistema empírico de PLI Aleph, utilizado no desenvolvimento deste trabalho.

4.1 Dimensões da PLI

Os sistemas de PLI podem ser divididos, segundo algumas características básicas, em várias dimensões (Muggleton, 1991), como descrito a seguir:

- podem aprender um *único* conceito ou *múltiplos* conceitos (predicados). No caso de aprender um único conceito, as observações são simplesmente exemplos desse conceito. A meta de um aprendizado de múltiplos conceitos é aprender um conjunto de definições de predicados, possivelmente interrelacionados;
- podem necessitar que todos os exemplos sejam dados antes de iniciar o processo de aprendizado (aprendizado em *batch* ou *não-incremental*) ou podem utilizar exemplos de treinamento individuais, fornecidos um a um, durante o processo de aprendizado (aprendizado *incremental*);
- podem precisar de um oráculo (um especialista ou o próprio usuário), durante o processo de aprendizado, para verificar a validade das generalizações e/ou classificar novos exemplos. Nesse caso, o sistema é chamado de *interativo*, caso contrário é chamado de *não-interativo*. Interatividade implica em incrementabilidade, mas pode ser encontrada também em alguns sistemas de aprendizado empírico que fazem uso de revisão de teoria;
- podem inventar termos (predicados) novos. Isso amplia o vocabulário disponível do conhecimento do domínio, e pode facilitar a tarefa de aprender um conceito. Sistemas que apresentam essa qualidade são chamados de sistemas de *indução construtiva* (Russel & Norvig, 2003);
- podem utilizar conhecimento do domínio intensional ou extensional. Uma teoria extensional é representada apenas por fatos *ground*¹, enquanto que uma teoria intensional contém tanto fatos *ground* quanto cláusulas com variáveis, permitindo reduzir a descrição do conceito. A grande maioria dos primeiros sistemas de PLI restringiam a teoria a exemplos *ground*, mas existem atualmente vários sistemas que permitem descrever a teoria de forma intensional, tais como Progol (Muggleton, 1995) e FOCL (Pazzani & Silverstein, 1991) entre outros e
- podem aceitar uma hipótese vazia, aprendendo um conceito do início, ou podem aceitar uma hipótese (teoria) inicial, a qual é revisada durante o processo de aprendizado. Sistemas que fazem esse tipo de revisão são chamados de *revisores de teoria*.

Embora essas dimensões sejam, a princípio, independentes, os sistemas atuais de PLI se situam em duas extremidades de um espectro. Em uma extremidade estão os sistemas não-interativos de aprendizado não-incremental, que aprendem um único predicado a partir do zero, denominados *sistemas empíricos* de PLI, e na outra ficam os sistemas interativos

¹ não contém variáveis, apenas literais.

e revisores de teoria, que aprendem múltiplos predicados e são denominados de *sistemas interativos* de PLI (Lavrač & Džeroski, 1994). MIS (Shapiro, 1983), MARVIN (Sammut & Banerji, 1986) e CLINT (De Raedt, 1992) são exemplos de sistemas de PLI interativos; eles aprendem definições de múltiplos predicados a partir de um conjunto pequeno de exemplos e consultas ao usuário.

Os sistemas empíricos de PLI tendem a aprender um único predicado utilizando uma grande coleção de exemplos. Essa classe de sistemas de PLI inclui FOIL (Quinlan, 1990), GOLEM (Muggleton & Feng, 1990), LINUS (Džeroski et al., 1995) e Progol (Muggleton, 1995). Os sistemas empíricos têm-se mostrado mais adequados para aplicações práticas (Lavrač & Džeroski, 1994). Neste trabalho é utilizado o sistema empírico de PLI Aleph, que permite, entre outros, simular diversos sistemas de PLI, o qual é descrito com detalhes na Seção 4.3 na página 41.

4.2 Descrição dos sistemas de PLI

Nesta seção são descritos brevemente alguns sistemas de PLI, apresentando algumas das suas características básicas. É apresentada também uma tabela — Tabela 4.1 na página 41 — na qual são apontadas características, descritas ao longo deste trabalho, para vários sistemas de PLI.

FOIL O sistema empírico FOIL (Quinlan, 1990) lê especificações extensionais de um conjunto de relações e produz definições de cláusulas de Horn para uma ou mais delas. Aprende múltiplos predicados de modo não-interativo e não-incremental, realizando uma busca *top-down* no espaço de hipóteses;

Progol O sistema empírico Progol (Muggleton, 1995) pode aprender múltiplos predicados, de modo não-interativo e não-incremental. Realiza uma busca do geral ao específico por um grafo de refinamento das cláusulas, ou seja, de forma *top-down*;

GOLEM O sistema empírico GOLEM (Muggleton & Feng, 1990) aprende um único predicado de cada vez de modo não-interativo e não-incremental a partir de uma busca *bottom-up*. Nesse sistema só pode ser utilizado conhecimento do domínio na forma de fatos *ground*;

FORS O sistema empírico de regressão relacional, FORS (Karalič & Bratko, 1997) realiza predição a partir de exemplos e conhecimento do domínio em problemas em que as classes possuem valores reais. Aprende um único predicado de cada vez de modo não-interativo e não-incremental realizando uma busca *top-down* no espaço de hipóteses;

MIS O sistema interativo e revisor de teoria MIS Shapiro (1983) aprende a definição de múltiplos predicados de forma incremental. Realiza busca *top-down* por um grafo de refinamento. Este foi o primeiro sistema de PLI a aceitar conhecimento do domínio de forma intensional e extensional;

Tilde O Tilde² (Blockeel, 1997) é um sistema para o aprendizado de árvores de decisão lógicas. Cada teste no caminho da raiz até as folhas da árvore é uma conjunção ou disjunção de literais que são parte de uma cláusula. A árvore resultante pode ser usada diretamente para classificação de novos exemplos, ou ser facilmente transformada em um programa lógico;

LINUS O LINUS (Džeroski et al., 1995) é um sistema empírico de aprendizado não-interativo e não-incremental que aprende um único predicado de cada vez. Ele transforma problemas de PLI para uma representação atributo-valor. O problema transformado é então solucionado por um sistema de aprendizado proposicional e a solução induzida é transformada em um conjunto de cláusulas.

Uma descrição de vários sistemas de PLI pode ser encontrada em (Lavrač & Flach, 2001); (Nienhuys-Cheng & Wolf, 1997); (Lavrač & Džeroski, 1994); (Muggleton & Raedt, 1994).

Na Tabela 4.1 são apontadas características de vários sistemas de PLI e quais características estão presentes em cada sistema. As características dessa tabela foram apresentadas no decorrer deste trabalho, tais como, qual o tipo de busca realizada — Seção 3.5 e 3.6, se realiza tarefa preditiva — Seção 3.2.1 — ou descritiva — Seção 3.2.2 —, entre outros. Essa tabela é composta pelas seguintes colunas:

1. Sistema: nome do sistema considerado;
2. TD: se o sistema realiza busca do tipo *top-down*, ou seja, utiliza técnicas de especialização;
3. BU: se o sistema realiza busca do tipo *bottom-up*, ou seja, utiliza técnicas de generalização;
4. Preditivo: se a tarefa de descoberta de conhecimento é preditiva, ou seja, regras de classificação são geradas;
5. Descritivo: se a tarefa de descoberta de conhecimento é descritiva, ou seja, apenas propriedades verdadeiras do conjunto de exemplos são observadas;

² *Top-down Induction of Logical Decision Trees*

6. Inc e N-Inc: se o sistema realiza aprendizado incremental ou não-incremental, respectivamente;
7. Int e N-Int: se o sistema é do tipo interativo ou não-interativo, respectivamente;
8. Mult-Pred: se o sistema pode aprender múltiplos predicados (conceitos).

Sistema	TD	BU	Preditivo	Descritivo	Inc	N-Inc	Int	N-Int	Multi-Pred
Aleph		✓	✓			✓	✓		
Cigol		✓			✓		✓	✓	
Claudian	✓			✓		✓		✓	
Clint					✓		✓		✓
FOIL	✓		✓			✓		✓	
FORS	✓		✓			✓		✓	
GOLEM		✓	✓			✓		✓	
LINUS						✓	✓		
MARVIN		✓					✓	✓	
MIS	✓				✓		✓		✓
MOBAL	✓		✓			✓		✓	✓
Progol	✓		✓			✓		✓	✓
Tilde				✓					
WARMR				✓					

Tabela 4.1: Resumo dos sistemas de PLI

Deve ser observado que o Aleph é apresentado nessa tabela como um sistema de busca *bottom-up* de aprendizado não-incremental e não-interativo, porém essas características podem ser modificadas. Por exemplo, modificando seus parâmetros ele pode realizar busca do tipo *top-down* ou mesmo realizar aprendizado de forma interativa. Assim, o sistema Aleph permite simular vários sistemas diferentes de PLI. O Aleph é apresentado com maiores detalhes a seguir.

4.3 O Sistema Aleph

O sistema Aleph (**A** *L*earning **E**ngine for **P**roposing **H**ypotheses) (Srinivasan, 2000) foi desenvolvido com o intuito de ser um protótipo para explorar idéias de PLI. Atualmente, Aleph emula funcionalidades de vários outros sistemas de PLI, tais como: Progol (Muggleton, 1995), FOIL (Quinlan, 1990), FORS (Karalič & Bratko, 1997), Indlog (Camacho, 1994), MIDOS (Wrobel, 1997), Tilde (Blockeel, 1997) e WARMR (Dehaspe & De Raedt, 1997).

O Aleph possui uma poderosa linguagem de representação que permite representar expressões complexas e incorporar novo conhecimento do domínio com facilidade. Além

dessas características comuns aos sistemas de PLI, o Aleph possui outras características muito interessantes, tais como: permitir escolher qual a ordem de geração das regras (geral para o específico ou específico para o geral); mudar a função de avaliação (entropia, precisão, Laplace entre outras) e a ordem de busca (*Hill-Climbing*, *Branch-and-Bound* entre outras). Essas características do sistema Aleph, além de ser um *software* de uso livre, motivaram a escolha desse sistema para o desenvolvimento deste trabalho.

4.3.1 Funcionamento Básico

O Aleph segue um procedimento muito simples, que pode ser descrito pelos seguintes quatro passos:

1. Seleciona um exemplo para ser generalizado. Se não existirem mais exemplos pára;
2. Constrói a cláusula mais específica, que implique no exemplo selecionado no passo anterior e que esteja de acordo com as restrições da linguagem. Normalmente é uma cláusula com muitos literais chamada de cláusula *bottom*, construída com resolução inversa. Esse passo é chamado de saturação;
3. Busca a cláusula mais geral que a cláusula *bottom*. O padrão é do tipo *Branch-and-Bound*. Esse passo é chamado de redução;
4. Adiciona a melhor cláusula à teoria e todos os exemplos redundantes são removidos e retorna ao passo 1.

O Aleph utiliza a informação contida nos seguintes três arquivos para construir uma teoria:

- **file.b**: contém a informação sobre o conhecimento do domínio (intensional e extensional), as restrições de busca e linguagem (declarações de modo e determinations descrito a seguir), restrições de tipos e parâmetros;
- **file.f**: contém os exemplos positivos (somente fatos *ground*);
- **file.n**: contém os exemplos negativos (somente fatos *ground*). Pode não existir, já que Aleph pode realizar aprendizado utilizando somente exemplos positivos.

4.3.2 Declarações de Modo

As declarações de modo, no arquivo **file.b**, descrevem as relações (predicados) entre objetos de dados e tipos desses dados. Essas declarações permitem também informar

ao Aleph se a relação pode ser utilizada na cabeça (declarações **modeh**) ou no corpo (declarações **modeb**) das regras geradas. Na declaração de modo também são descritos o tipo dos argumentos de cada predicado e as possíveis instâncias desses argumentos. As declarações tem o seguinte formato:

`mode(Numero_Chamadas,Modo).`

O `Numero_Chamadas`, ou como é mais conhecido, o *recall*, determina o limite do número de *instanciações* alternativas de um predicado. Uma instanciação do predicado é uma substituição de tipos para cada variável ou constante. O *recall* pode ser qualquer número positivo $n \geq 1$ ou *'**'.

Se é conhecido que há somente um certo número de soluções para uma particular instanciação é possível informar isso pelo *recall*. Por exemplo, para uma declaração do predicado `progenitor_de(P,F)` poderia ser dado um *recall* igual a 2, pois todas as pessoas terão no máximo dois progenitores, enquanto que o *recall* seria 4 para uma declaração do predicado `avo(A,N)`. O *recall* *'**' é utilizado quando não há limites para o número de soluções para uma instanciação. Por exemplo, em princípio, não há limites para o número de ancestrais que uma pessoa pode ter. A segunda parte da declaração do **mode**, denominada de **Modo**, indica o formato do predicado que será utilizado e tem o seguinte formato:

`predicado(Tipo_Argumento1,Tipo_Argumento2,...,Tipo_Argumenton).`

Por exemplo, para o aprendizado da relação `tio_de(T,S)` com conhecimento do domínio descrito pelas relações `progenitor_de(P,F)` e `irmao_de(I1,I2)`, as declarações de modo podem ser:

```
:- modeh(1,tio_de(+pessoa,+pessoa)).
:- modeb(*,progenitor_de(-pessoa,+pessoa)).
:- modeb(*,progenitor_de(+pessoa,-pessoa)).
:- modeb(*,irmao_de(+pessoa,-pessoa)).
```

A declaração **modeh** indica o predicado que irá compor a cabeça das regras. No exemplo, **modeh** informa que a cabeça das regras deve ser `tio_de(T,S)`, onde `T` e `S` são do tipo `pessoa`. O símbolo *'+'* que aparece antes do tipo indica que esse argumento do predicado é uma *variável*. Assim, a cabeça pode ter a forma `tio_de(T,S)`, mas não `tio_de(ana,maria)`. O símbolo *'-'* indica uma variável de saída e o símbolo *#* indica que esse argumento poderia ser uma constante.

A segunda declaração, `modeb`, indica que as regras geradas podem ter no corpo o predicado `progenitor_de(P,F)`, no qual `P` e `F` são do tipo `pessoa`. A primeira declaração `modeb` pode ser usada para adicionar `progenitor_de/2` no corpo das regras e introduzir um ou mais `progenitor/2` de um filho (observar que `Numero Chamadas` está instanciado com `'*`'). Similarmente, a segunda declaração `modeb` permite que `progenitor_de/2` seja usado no corpo para encontrar um ou mais filhos de um progenitor, e a última declaração pode ser usada no corpo para encontrar um ou mais irmãos de um indivíduo.

4.3.3 Tipos

Os tipos devem ser especificados para cada argumento dos predicados utilizados na construção de uma hipótese. Para o Aleph os tipos são só nomes e a declaração de tipos nada mais é que um conjunto de fatos. Por exemplo, a descrição de objetos do tipo `pessoa` poderia ser

```
pessoa(jane).
pessoa(henrique).
pessoa(sueli).
pessoa(junior).
...
```

4.3.4 Declaração dos Determinations

O Aleph utiliza a declaração `determination` para declarar os predicados que podem ser usados para construir uma hipótese. Essa declaração tem o formato:

```
determination(Pred_Alvo/Aridadea,Pred_Corpo/Aridadec).
```

O primeiro argumento é o nome do predicado alvo e sua aridade, isto é, o predicado que irá aparecer na cabeça da regra induzida. O segundo argumento consiste do nome e aridade de um predicado que pode aparecer no corpo da cláusula. Por exemplo, para o aprendizado da relação `tio_de(T,S)`, uma declaração seria:

```
determination(tio_de/2,progenitor_de/2).
```

Tipicamente, são feitas muitas declarações para um predicado alvo, as quais correspondem aos predicados relevantes para construção de uma cláusula (regra). Caso nenhuma declaração para `determination` seja apresentada, o Aleph não constrói nenhuma cláusula.

4.3.5 Exemplos Positivos e Negativos

Exemplos positivos e negativos do conceito a ser aprendido devem ser apresentados na forma extensional (fatos) nos arquivos com extensão `.f` e `.n` respectivamente. Por exemplo, para o aprendizado do conceito `tio_de(T,S)` poderíamos ter a declaração dos seguintes exemplos positivos no arquivo com extensão `.f`:

```
tio_de(jane,henrique).
tio_de(sueli,henrique).
...
```

E os seguintes exemplos negativos no arquivo com extensão `.n`:

```
tio_de(julia,junior).
tio_de(julia,henrique).
...
```

4.3.6 Parâmetros

O Aleph permite ao usuário determinar uma variedade de restrições no espaço das possíveis hipóteses a serem aprendidas, bem como na busca realizada nesse espaço de possíveis hipóteses, pela definição de novos valores dos parâmetros disponíveis. O predicado `set/2` permite ao usuário definir o valor do parâmetro `Parametro`

```
set(Parametro,Valor)
```

O valor corrente de um determinado parâmetro pode ser obtido usando

```
setting(Parametro,Valor)
```

enquanto que

```
noset(Parametro)
```

retorna o valor desse parâmetro para seu valor padrão. No Apêndice A são descritos alguns dos parâmetros disponíveis no Aleph .

4.3.7 Características Importantes

O Aleph possui algumas características importantes que permitem a modificação de cada um dos passos apresentados na Seção 4.3.1 na página 42, tais como:

- Seleção de exemplos: ao invés de selecionar apenas um exemplo inicial para ser generalizado, é possível escolher mais que um. Caso sejam selecionados n exemplos iniciais; para cada um deles é criada uma cláusula *bottom*; depois do passo de redução a melhor entre as n reduções é adicionada à teoria;
- Construção da cláusula mais específica: especifica o estágio em que a cláusula *bottom* é construída;
- Busca: a busca por cláusulas pode ser alterada para diferentes estratégias de busca, funções de avaliação e operadores de refinamento;
- Remoção de redundantes: exemplos cobertos podem ser retidos para dar uma melhor estimativa das cláusulas obtidas.

4.3.8 Um Exemplo de Execução

Com o objetivo de ilustrar os passos de execução do Aleph, a seguir é mostrado o exemplo dos trens de Michalski. Dada uma sequência de trens — Figura 4.1 — cada trem tem um número de vagões e cada vagão pode ter diferentes propriedades, indicadas pelo desenho de uma figura com um determinado formato. Também sabe-se se o trem está viajando para o Leste (East) ou para o Oeste (West). O problema é encontrar uma regra capaz de prever, dada as propriedades dos seus vagões, se o trem está viajando para leste; esse trem é chamado de *eastbound*.

A Tabela 4.2 mostra a declaração dos exemplos positivos e negativos nos arquivos correspondentes `trens.f` e `trens.n`

O próximo passo é definir o arquivo `trens.b` com o conhecimento de fundo e as diretivas para a execução de Aleph. A primeira diretiva `set/2` indica que as regras que constituem a hipótese devem cobrir, no mínimo, dois exemplos positivos. Exceto essa definição de `minpos`, cujo valor padrão é 1, Aleph foi executado com os parâmetros padrão – Apêndice A. O conceito a ser aprendido é denominado `eastbound/1`, portanto deve ser declarado com `modeh/2`, e os predicados que podem aparecer na regra devem ser

Figura 4.1: Trens que viajam para Leste e Oeste

Ex.(+)	trens.f	Ex.(-)	trens.n
1	eastbound(east1).	6	eastbound(west6).
2	eastbound(east2).	7	eastbound(west7).
3	eastbound(east3).	8	eastbound(west8).
4	eastbound(east4).	9	eastbound(west9).
5	eastbound(east5).	10	eastbound(west10).

Tabela 4.2: Arquivos `trens.f` e `trens.n`

declarados com `modeb/2`, utilizando os tipos de dados dos argumentos desses predicados, definidos mais a frente, bem como as instanciações desses argumentos, como mostrado a seguir.

```
:- set(minpos,2).
:- modeb(1, eastbound(+train)).
:- modeb(1, short(+car)).
:- modeb(1, closed(+car)).
:- modeb(1, long(+car)).
:- modeb(1, open_car(+car)).
:- modeb(1, double(+car)).
:- modeb(1, jagged(+car)).
:- modeb(1, shape(+car, #shape)).
:- modeb(1, load(+car, #shape, #int)).
:- modeb(1, wheels(+car, #int)).
:- modeb(*, has_car(+train, -car)).
```

Também, devem ser declarados os `determination/2`, especificando os predicados que podem aparecer na cabeça e no corpo das regras com essa cabeça, como mostrado a seguir.

```
:- determination(eastbound/1, short/1).
:- determination(eastbound/1, closed/1).
:- determination(eastbound/1, long/1).
```

```

:- determination(eastbound/1,open_car/1).
:- determination(eastbound/1,double/1).
:- determination(eastbound/1,jagged/1).
:- determination(eastbound/1,shape/2).
:- determination(eastbound/1,wheels/2).
:- determination(eastbound/1,has_car/2).
:- determination(eastbound/1,load/3).

```

Para definir o tipo de dados `car/1` enumerando os possíveis valores que pode assumir, esses valores foram definidos considerando primeiro o trem e segundo o vagão. Por exemplo `car(car_14)` refere-se ao valor do tipo `car` para o trem 1 vagão 4. Também são definidos `shape/1` que define os possíveis formatos das figuras relacionadas às propriedades dos vagões, bem como `train/1` que define os trens.

```

car(car_11).  car(car_12).  car(car_13).  car(car_14).
car(car_21).  car(car_22).  car(car_23).
car(car_31).  car(car_32).  car(car_33).
car(car_41).  car(car_42).  car(car_43).  car(car_44).
car(car_51).  car(car_52).  car(car_53).
car(car_61).  car(car_62).
car(car_71).  car(car_72).  car(car_73).
car(car_81).  car(car_82).
car(car_91).  car(car_92).  car(car_93).  car(car_94).
car(car_101). car(car_102).

```

```

shape(ellipse).  shape(hexagon).  shape(rectangle).  shape(u_shaped).
shape(triangle). shape(circle).  shape(nil).

```

```

train(east1).  train(east2).  train(east3).  train(east4).  train(east5).
train(west6).  train(west7).  train(west8).  train(west9).  train(west10).

```

O próximo passo consiste em definir cada um dos 10 trens. Por exemplo, o trem 1 é definido como

```

short(car_12).           % vagão 2 é curto
closed(car_12).          % vagão 2 é fechado
long(car_11).            % vagão 1 é longo
long(car_13).            % vagão 3 é longo
short(car_14).           % vagão 4 é longo
open_car(car_11).        % vagão 1 é aberto
open_car(car_13).        % vagão 3 é aberto
open_car(car_14).        % vagão 4 é aberto
shape(car_11,rectangle). % vagão 1 tem forma rectangular
shape(car_12,rectangle). % vagão 2 tem forma rectangular
shape(car_13,rectangle). % vagão 3 tem forma rectangular

```



```

shape(car_14,rectangle).    % vagão 4 tem forma rectangular
load(car_11,rectangle,3).   % vagão 1 carrega 3 retângulos
load(car_12,triangle,1).    % vagão 1 carrega 1 triângulo
load(car_13,hexagon,1).     % vagão 1 carrega 1 hexágono
load(car_14,circle,1).      % vagão 1 carrega 1 círculo
wheels(car_11,2).           % vagão 1 tem 2 rodas
wheels(car_12,2).           % vagão 1 tem 2 rodas
wheels(car_13,3).           % vagão 1 tem 3 rodas
wheels(car_14,2).           % vagão 1 tem 2 rodas
has_car(east1,car_11).      % o vagão 1 é do trem east1
has_car(east1,car_12).      % o vagão 2 é do trem east1
has_car(east1,car_13).      % o vagão 3 é do trem east1
has_car(east1,car_14).      % o vagão 4 é do trem east1

```

Os outros 9 (nove) trens devem ser descritos de maneira semelhante no arquivo `trens.b`. Considerando que todos os trens estão descritos nesse arquivo, os três arquivos são lidos executando `read_all/1` e, logo após, a indução pode ser ativada executando `induce/0`, como mostrado a seguir.

```

?- read_all(train).
?- induce.

```

Aleph executa então os quatro passos descritos na Seção 4.3.1 na página 42. No passo 1 seleciona um exemplo para ser generalizado, neste caso o exemplo `eastbound(east1)`. No passo 2, saturação, é construída a cláusula *bottom* do exemplo, mostrada a seguir

```

[bottom clause]
eastbound(A) :-
    has_car(A,B), has_car(A,C), has_car(A,D), has_car(A,E),
    short(B), short(D), closed(D), long(C),
    long(E), open_car(B), open_car(C), open_car(E),
    shape(B,rectangle), shape(C,rectangle), shape(D,rectangle), shape(E,rectangle),
    wheels(B,2), wheels(C,3), wheels(D,2), wheels(E,2),
    load(B,circle,1), load(C,hexagon,1), load(D,triangle,1), load(E,rectangle,3).

```

No passo 3, redução, procura a cláusula mais geral que, neste caso, é a melhor cláusula (cobre os 5 exemplos positivos e nenhum negativo). Assim, ela é adicionada à teoria no passo 4, e os exemplos cobertos são removidos. Como não há mais exemplos positivos, Aleph pára.

```

[theory]

[Rule 1] [Pos cover = 5 Neg cover = 0]
eastbound(A) :-
    has_car(A,B), short(B), closed(B).

```

Após mostrar a teoria, é também informada a performance da teoria induzida no conjunto de treinamento e outras estatísticas, como mostrado a seguir.

[Training set performance]

Actual		
	+	-
Pred		
+	5	0
-	0	5
	5	5
	5	5
	5	10

Accuracy = 1.0

[Training set summary] [[5,0,0,5]]

[time taken] [0.07]

[total clauses constructed] [100]

4.4 Considerações Finais

Os sistemas atuais de PLI se situam em duas sub-áreas, os sistemas interativos de PLI, os quais estão fortemente relacionados com os sistemas revisores de teoria, e um pequeno número de exemplos está disponível, e os sistemas empíricos, nos quais a ênfase é a extração de padrões de um grande número de exemplos. O foco deste trabalho está neste último tipo de sistemas.

Entre os exemplos de sistemas empíricos, tais como o FOIL, GOLEM, LINUS e Progol, encontra-se o sistema de PLI Aleph, muito bem conceituados pela comunidade de PLI. O Aleph, descrito neste capítulo, permite simular vários sistemas de PLI empíricos, o que motivou seu uso no desenvolvimento deste trabalho, sendo integrado à ferramenta proposta neste trabalho. Essa ferramenta é apresentada no capítulo a seguir.

A Ferramenta Proposta

*P*ara aproveitar a expressividade dos sistemas relacionais quando os exemplos estão descritos no formato atributo-valor, é necessário transformar essa descrição proposicional para uma descrição relacional equivalente. Surge assim a necessidade de automatizar, tanto quanto possível, esse processo de transformação. Esse é o objetivo do módulo conversor *Kaeru*¹, que transforma bases de dados proposicionais para o formato relacional para que possam ser utilizadas por sistemas de PLI. Além disso, o módulo *Kaeru* cria os arquivos necessários para executar o sistema de PLI Aleph.

O módulo conversor *Kaeru* é um dos módulos do ambiente de PLI, o qual será integrado ao projeto DISCOVER. O DISCOVER é um ambiente computacional, para descoberta de conhecimento de bases de dados, que oferece funcionalidades voltadas para o aprendizado de máquina proposicional. Assim, o ambiente de PLI tem como objetivo permitir a incorporação de algoritmos de PLI ao ambiente DISCOVER. O módulo conversor *Kaeru* e o ambiente DISCOVER são apresentados neste capítulo.

¹Kaeru é uma palavra de origem japonesa que significa converter ou transformar.

5.1 O Projeto DISCOVER

Trabalhos na área de aquisição automática de conhecimento vêm sendo desenvolvidos há vários anos no LABIC ². A maioria desses trabalhos são realizados de forma cooperativa. Com o decorrer dos anos, foram observadas algumas sobreposições de implementações. Esses, e outros fatores, levaram alguns pesquisadores do LABIC a propor a implementação do DISCOVER (Baranauskas & Batista, 2000). O objetivo principal do projeto DISCOVER é reduzir problemas de implementação e integração de ferramentas e algoritmos para apoiar todas as etapas do processo de descoberta de conhecimento, oferecendo funcionalidades voltadas para aprendizado de máquina, mineração de dados e mineração de textos. O projeto DISCOVER pode ser entendido como um conjunto de métodos para manipular dados e conhecimento por meio de sintaxes padrão (Batista & Monard, 2003b; Batista, 2001; Prati et al., 2001b) para a representação de dados e do conhecimento induzido por diferentes algoritmos de aprendizado, bem como por bibliotecas que oferecem um conjunto de funcionalidades básicas para tais manipulações. Assim, foi criado um ambiente integrado, no qual bibliotecas de classes desenvolvidas em *Perl* (Wall et al., 1996), são integradas por uma interface gráfica (Geromini, 2002). Um *framework* para a integração dos componentes do ambiente DISCOVER foi proposta por Prati (2003) utilizando *software patterns*, no qual os componentes são integrados por meio de uma linguagem baseada em XML.

O DISCOVER já conta com vários ambientes, tais como o ambiente Discover Learning Environment - DLE, composto pela biblioteca de classes Discover Object Library - DOL, e o ambiente para gerenciamento de experimentos SNIFFER, que foram desenvolvidos para as diferentes tarefas de pré-processamento de dados no formato atributo-valor e análise dos classificadores induzidos por algoritmos de aprendizado utilizados no processo de Mineração de Dados (Batista, 2003). Novas funcionalidades estão sendo especificadas, principalmente para representação de regras de regressão (Dosualdo, 2003), regras de associação (Melanda, 2002) e o módulo para PLI, apresentado neste trabalho. Além disso, outros trabalhos foram desenvolvidos na área de Aprendizado de Máquina (Caulkins, 2000; Pila, 2001; Pila & Monard, 2002; Milaré, 2003), Mineração de Dados (Lee, 2003; Pugliesi, 2001; Baranauskas & Monard, 2003) e Mineração de Textos (Martins, 2003; Matsubara et al., 2003).

Na Figura 5.1 é mostrado, de uma forma simplificada, como os filtros, sintaxes e bibliotecas interagem uns com os outros para o caso de algoritmos simbólicos proposicionais de aprendizado supervisionado. Os dados, no formato atributo-valor da sintaxe padrão do

²<http://labic.icmc.sc.usp.br>

DISCOVER, podem ser submetidos a diversos processos de pré-processamento já implementados (Batista, 2003; Batista & Monard, 2003a; Batista et al., 2004; Prati et al., 2004). Após, esses dados são convertidos para a sintaxe padrão do algoritmo de aprendizado que o usuário deseja executar. O conhecimento (classificador) simbólico induzido por esse algoritmo de aprendizado, é posteriormente convertido para a sintaxe padrão da linguagem de representação de conceitos proposicionais do DISCOVER (Prati et al., 2001b). Esse conhecimento pode ser posteriormente submetido a diversos processos de pós-processamento de conhecimento implementados e sendo implementados no DISCOVER (Melanda, 2004; Pugliesi, 2004; Bernardini, 2003). As linhas tracejadas na Figura 5.1 mostram os processos que utilizam a biblioteca de classes DOL e rotinas implementadas no ambiente DLE do DISCOVER.

Figura 5.1: Interação entre filtros, sintaxes e bibliotecas, (Chiara & Monard, 2003)

5.1.1 A Biblioteca de Classes Discover Object Library - DOL

A biblioteca DOL é uma biblioteca orientada a objeto baseada em padrões de projeto (Gamma et al., 1995). As classes dessa biblioteca implementam as tarefas de manipulação e gerenciamento de dados mais comuns em pré-processamento, tais como gerenciamento de diferentes sintaxes de arquivos de dados e atributos, amostragens, métodos de *resampling*, estatísticas descritivas, normalizações de dados, entre outros (Batista & Monard, 2003b). O objetivo da DOL é dar suporte à criação de novos métodos de pré-processamento de dados e ser uma biblioteca de métodos de pré-processamento.

O conjunto de métodos da DOL permite ao desenvolvedor uma forma simples de ter acesso aos dados. Os dados podem estar tanto disponíveis em arquivos texto como em tabelas de banco de dados relacionais. Os dados são carregados em uma estrutura e uma grande quantidade de métodos fornece acesso aos dados, possibilitando sua manipulação. Em seguida, eles podem ser novamente armazenados em arquivos texto, em diferentes sintaxes, ou carregados em uma tabela de banco de dados relacional. Entre as principais funcionalidades da biblioteca DOL pode-se citar: manipulação de atributos e dados, integração com diversos sistemas de aprendizado de máquina, integração com sistemas gerenciadores de banco de dados, filtros de exemplos, estatísticas descritivas e correlações, além de diversos métodos de *resampling*. A biblioteca DOL foi desenvolvida em uma arquitetura modular na qual cada módulo é constituído de uma ou mais classes que realizam um conjunto bem definido de tarefas. A biblioteca possui um módulo central chamado Core, o qual carrega um conjunto de dados em uma estrutura e disponibiliza, atualmente, mais de 60 métodos capazes de consultar e manipular essa estrutura. O módulo Core é o único que precisa obrigatoriamente ser carregado por uma aplicação que utiliza a biblioteca. Já os demais módulos são carregados apenas quando for necessário.

5.1.2 O Ambiente Computacional SNIFFER

O ambiente computacional SNIFFER é um ambiente de gerenciamento de avaliações e comparações experimentais de algoritmos de AM. O ambiente computacional SNIFFER automatiza a avaliação experimental, e está atualmente integrado com diversos sistemas de AM simbólico tais como *C4.5* (Quinlan, 1988), *C4.5 Rules* (Quinlan, 1987), *ID3* (Quinlan, 1986), *CN2* (Clark & Boswell, 1991) e *Newld* (Boswell, 1990b). Dessa forma, é possível comparar o desempenho de novos métodos de pré-processamento de dados aplicados a diversos sistemas de aprendizado.

O ambiente SNIFFER complementa a biblioteca DOL pois permite que diferentes indutores proposicionais bem como métodos de pré-processamento de dados desenvolvidos sejam avaliados e comparados experimentalmente de uma forma rápida e segura (Batista & Monard, 2003b).

5.1.3 A Sintaxe Padrão para Conjuntos de Dados no Formato Atributo-Valor

No desenvolvimento de pesquisas na área de extração de conhecimento, realizar uma investigação que envolve extrair conhecimento de vários conjuntos de dados utilizando

diversos sistemas de aprendizado é, normalmente, muito trabalhoso. Isso porque, sistemas de aprendizado usam sintaxes diferentes para representar os dados e atributos nos respectivos arquivos de entrada. Por esse motivo, no **DISCOVER**, foi decidido adotar uma sintaxe padrão para representação de dados no formato atributo-valor e representação do conhecimento extraído por algoritmos de AM simbólicos proposicionais (Prati et al., 2001b; Batista, 2001), bem como implementar bibliotecas que oferecem funcionalidades sobre essas sintaxes padrão. A partir dessa sintaxe padrão, a qual foi dada o nome de DSX (*Discover Dataset Syntax*), é possível utilizar a biblioteca de classes **DOL**, para converter um arquivo de dados para a sintaxe utilizada por diversos sistemas de aprendizado.

A sintaxe padrão de dados no formato atributo-valor utiliza arquivos do tipo texto para declarar os atributos, seus respectivos tipos e os valores que esses atributos assumem para um determinado conjunto de exemplos. Dado um conjunto de exemplos, um arquivo com extensão **.names** é utilizado para declarar os atributos, enquanto os valores que esses atributos assumem para esse conjunto de exemplos são declarados em um arquivo com a extensão **.data**. Os dois arquivos devem possuir o mesmo nome, diferenciando-se apenas pela extensão. Opcionalmente, também podem ser usados arquivos de dados com as extensões **.test**, com casos rotulados de teste para medir o erro de classificação, **.validation** com casos rotulados para validação de modelos e **.cases** com casos não rotulados para serem rotulados pelo classificador.

A fim de ilustrar, considere o subconjunto de seis exemplos do conjunto de dados **voyage** (Quinlan, 1988), mostrados na Tabela 5.1

Exemplo	outlook	temperature	humidity	windy	class
1	sunny	25	72	yes	go
2	sunny	28	91	yes	dont_go
3	overcast	23	90	yes	go
4	overcast	29	78	no	go
5	rain	22	95	no	go
6	rain	19	70	yes	dont_go

Tabela 5.1: Subconjunto de dados **voyage**

Cada exemplo encontra-se rotulado com a classe **go** ou **dont_go**, e é descrito pelos seguintes atributos:

- **outlook**: que pode assumir os valores **sunny**, **overcast** ou **rain**;
- **temperature**: que pode assumir um valor numérico;
- **humidity**: que pode assumir um valor numérico;

- **windy**: que pode assumir os valores **yes** ou **no**.

Na Tabela 5.2 é apresentado o arquivo de declaração de atributos **voyage.names**, na sintaxe DSX, para o conjunto de dados artificial **voyage**.

```
class.          | Class Attribute

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

Tabela 5.2: Exemplo de arquivo de declaração de atributos: **voyage.names**

Na Tabela 5.3 é apresentado o arquivo de declaração de dados correspondente aos exemplos descritos na Tabela 5.1. Nesse arquivo são declarados, na mesma ordem dos atributos no arquivo **.names**, os valores que esses atributos assumem para um determinado exemplo.

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

Tabela 5.3: Exemplo de arquivo de declaração de dados: **voyage.data**

5.1.4 Tipos de Dados Implementados

Na DOL foram definidos alguns tipos de dados que podem ser associados aos identificadores de atributos. Segue uma breve descrição e exemplo de alguns dos tipos de dados implementados.

Nominal O tipo nominal é utilizado para declarar atributos que podem assumir um grupo restrito de valores.

```
Atrib1: nominal(circulo,quadrado).
```


Enumerated O tipo enumerated é bastante semelhante ao tipo nominal. A principal diferença é que no tipo enumerated é possível definir uma ordem que os valores podem assumir.

Atrib2: `enumerated(pequeno,médio,grande)`.

Integer O tipo integer é utilizado para declarar um atributo que assume valores inteiros.

Atrib3: `integer`.

Real O tipo real é semelhante ao integer exceto pelo real armazenar números com ou sem parte fracionária.

Atrib4: `real`.

String Um atributo do tipo string pode assumir como valores seqüências de caracteres de tamanho indefinido e com quaisquer caracteres, inclusive (`\n`).

Atrib5: `string`.

Date Este tipo de dado é capaz de armazenar datas no formato `aaaa/mm/dd` (ano, mês e dia).

Atrib6: `date`.

Time Este tipo de dado permite declarar atributos que podem conter um horário e devem estar no formato `hh:mm:ss` (horas, minutos e segundos).

Atrib7: `time`.

Uma descrição completa dos tipos de dados e facilidades implementadas na DOL podem ser encontrados em (Batista & Monard, 2003b).

5.2 O Módulo Conversor *Kaeru*

No processo de extração de conhecimento de base de dados, caso esses dados estejam no formato atributo-valor, é possível utilizar, entre outros, sistemas de aprendizado simbólico proposicionais para extrair conhecimento dos dados. Entretanto, como já mencionado, a linguagem de descrição de hipóteses desses sistemas não permite descobrir relações implícitas nos dados, ou seja, conhecimento relacional.

O primeiro passo para tentar descobrir conhecimento relacional utilizando dados que estão no formato atributo-valor, é transformar esses dados para o formato relacional. Após essa transformação, esses dados podem ser utilizados por algoritmos de PLI para induzir conhecimento relacional. O objetivo do módulo conversor *Kaeru* (Ferro & Monard, 2004)

é automatizar a transformação de dados no formato atributo-valor na sintaxe padrão do DISCOVER, descrita na Seção 5.1.3 na página 54, para o formato relacional. Além disso, foi também decidido construir as diretivas necessárias para utilizar o sistema Aleph. Esse sistema, descrito em maiores detalhes na Seção 4.3, permite simular diversos algoritmos de PLI, facilitando assim o trabalho de pesquisa na área.

5.2.1 Descrição do Módulo Conversor *Kaeru*

O módulo conversor *Kaeru* é um dos módulos do ambiente de PLI a ser integrado no DISCOVER. Assim, a sua implementação segue os mesmos padrões de implementação. O módulo utiliza as facilidades da biblioteca de classes DOL e o ambiente para gerenciamento de experimentos SNIFFER PLI. O SNIFFER PLI, atualmente em desenvolvimento, é semelhante ao ambiente SNIFFER — Seção 5.1.2 na página 54 —, mas para gerenciamento de experimentos relacionados, exclusivamente, a algoritmos de PLI. Ele está integrado ao sistema Aleph e acrescido de funcionalidades específicas para manipular dados relacionais. A Figura 5.2 mostra como esses ambientes, bibliotecas e módulos estão integrados no ambiente de PLI.

Figura 5.2: Ambiente de PLI: interação entre o módulo conversor *Kaeru*, o Aleph, a biblioteca DOL e o Ambiente SNIFFER PLI

A entrada do módulo conversor *Kaeru* consiste dos arquivos `.names` e `.data` no formato DSX — Seção 5.1.3 —, os quais são inicialmente manipulados e verificados pelos métodos da biblioteca DOL. Além desses dois arquivos, também são necessários mais dois

arquivos de entrada com extensão `.param` e `.bk`. O arquivo com extensão `.param` contém a declaração dos parâmetros a serem utilizados pelo Aleph, enquanto que o arquivo com extensão `.bk` contém declarações, em um formato padrão descrito na Seção 5.2.2, a serem utilizadas pelo módulo conversor *Kaeru*. Os quatro arquivos de entrada devem ter o mesmo nome, se diferenciando apenas pela sua extensão.

Deve ser observado que antes de fazer a conversão, é verificada a existência, no arquivo `.names`, de um atributo especial denominado `att_id`, cujo valor identifica cada um dos exemplos no arquivo `.data`. Caso esse atributo não exista, ele é inserido internamente pelo conversor, tanto no arquivo `.names` quanto no arquivo `.data`³.

Utilizando a informação desses quatro arquivos de entrada, são construídos os três arquivos necessários para a execução do sistema Aleph. Esses três arquivos, com extensão `.b`, `.f` e `.n`, contém as seguintes informações:

- `.b` conhecimento do domínio;
- `.f` exemplos positivos;
- `.n` exemplos negativos.

Após a conversão automática dos arquivos, o usuário pode fazer uma intervenção manual nesses arquivos, pois eles são gerados com uma configuração padrão para o sistema Aleph. Caso o usuário precise realizar execuções utilizando outras configurações do Aleph, ele pode modificar tais parâmetros, ou mesmo adicionar mais conhecimento do domínio específico para o domínio da sua aplicação.

O usuário pode simplesmente utilizar o sistema Aleph para induzir conhecimento (hipótese) dos dados ou pode, também, utilizar o Ambiente **SNIFFER** PLI para medir o erro da hipótese induzida usando *k-fold cross-validation*, ou criando conjuntos específicos de treinamento e teste para executar o Aleph.

5.2.2 Formato dos Arquivos de Entrada

Como mencionado anteriormente, o módulo conversor *Kaeru* utiliza quatro arquivos de entrada com extensão `.names`, `.data`, `.param` e `.bk` para realizar a conversão de um conjunto de exemplos no formato atributo-valor para a representação relacional utilizada pelo Aleph. Os arquivos de entrada `.names` e `.data`, que descrevem os exemplos no formato

³Nesse caso o valor do atributo `att_id` do primeiro exemplo é 0, do segundo exemplo é 1 e assim por diante.

atributo-valor, devem estar no formato DSX do DISCOVER, descrito na Seção 5.1. O formato utilizado para definir a informação contida nos outros dois arquivos é apresentado brevemente nesta seção. Maiores detalhes podem ser encontrados em (Ferro & Monard, 2004).

Arquivo .bk

Este arquivo contém declarações utilizadas pelo módulo conversor *Kaeru*, as quais são necessárias para a criação do arquivo .b. Na Tabela 5.4 é apresentado um exemplo desse arquivo para o conjunto de dados *voyage* (Quinlan, 1988) descrito na Seção 5.1.3 na página 54.

```
head{}  
body{}  
positive{go}  
negative{dont_go}
```

Tabela 5.4: Exemplo de arquivo de declaração de parâmetros do módulo conversor *Kaeru*: *voyage.bk*

O primeiro parâmetro que deve ser declarado no arquivo .bk é o **head**. Este parâmetro define a cabeça do predicado a ser induzido pelo sistema, ou seja, qual é o predicado alvo. No sistema de PLI Aleph, assim como na maioria dos sistemas de PLI, o predicado alvo é definido pelo parâmetro **modeh**. A declaração **head** indica ao conversor qual o predicado alvo do **modeh** para o conjunto de dados apresentado nos arquivos de entrada .names e .data. A declaração é feita utilizando-se a palavra **head**, seguida por parâmetros entre colchetes ({}). Caso não seja fornecido nenhum parâmetro, como no exemplo apresentado na Tabela 5.4, o conversor cria um predicado padrão que utiliza como predicado alvo o próprio nome da classe (declarado no arquivo .names) e que tem sempre aridade 2.

No caso do usuário declarar parâmetros para **head**, ele pode definir qual é o predicado alvo desejado. Porém, deve ser observado que o nome desse predicado deve ser o nome de qualquer dos atributos declarados no arquivo .names. Neste caso, o usuário também define qual é a aridade do predicado, que pode ter um ou vários argumentos. Os tipos de cada argumento devem ter o mesmo nome de qualquer dos atributos declarados no arquivo .names.

O segundo parâmetro que deve ser declarado no arquivo bk é o **body**. Ele define os predicados que podem aparecer no corpo das regras induzidas por Aleph. Nos sistemas

de PLI, esses predicados são definidos pelo parâmetro **modeb**. A declaração é feita por meio da palavra **body**, seguida por abre e fecha colchetes (**{}**) e admite zero, um ou vários parâmetros entre os colchetes. Caso não seja fornecido nenhum parâmetro para **body**, como no exemplo apresentado na Tabela 5.4 na página oposta, o conversor cria, por *default*, tantos **modeb** quanto atributos (exceto o atributo **classe** e **att_id**) tenha o conjunto de exemplos considerado (arquivo **.names**). O nome de cada um desses atributos é usado para nomear um predicado de aridade 2.

No caso do usuário definir os parâmetros de **body** ele pode definir qualquer número de argumentos (aridade do predicado). Deve ser observado que os nomes dos argumentos deve ser idêntico ao nome de um dos atributos declarados no arquivo **.names**.

O terceiro e quarto parâmetro declarados são, respectivamente, os parâmetros **positive** e **negative**. O primeiro é para declarar a classe positiva dos exemplos e o segundo a classe negativa. Essa informação é usada pelo conversor para construir, respectivamente, o arquivo de exemplos positivos **.f** e negativos **.n**. A declaração é feita utilizando-se, respectivamente, a palavra **positive** ou **negative** seguida por parâmetros entre colchetes. As declarações permitem zero ou mais parâmetros.

Arquivo **.param**

Neste arquivo podem ser declarados qualquer um dos parâmetros do conjunto de parâmetros do Aleph — Apêndice A na página 99 —, na mesma sintaxe usada pelo Aleph, como ilustrado na Tabela 5.5.

```
:- set(minpos,3).  
:- set(clauselength,5).
```

Tabela 5.5: Exemplo de arquivo de declaração de parâmetros do Aleph: **voyage.param**

Cada declaração se inicia com o sinal **:-**, seguido pelo comando **set**, o parâmetro que se deseja modificar e um ponto no fim da declaração. Cada declaração deve ser feita em uma linha do arquivo. Caso nenhuma declaração seja feita, o Aleph será executado com todos os valores padrão dos seus parâmetros. Neste arquivo podem ser declarados qualquer número de parâmetros do Aleph, os quais serão integrados ao arquivo **.b**, descrito na Seção 5.2.3 na próxima página.

5.2.3 Formato do Arquivo de Saída .b

Entre os três arquivos de saída gerados pelo módulo conversor *Kaeru*, encontram-se os arquivos com os exemplos positivos (.f) e negativos (.n), cujo formato foi descrito na Seção 4.3.5 na página 45. O terceiro arquivo .b contém os parâmetros que direcionam a execução do Aleph. Os dados que estão no arquivo de conhecimento do domínio (.b), também são gerados automaticamente pelo módulo *Kaeru*, utilizando as informações que estão nos arquivos de entrada apresentados na Seção 5.2.2 na página 59.

A sintaxe dos dados segue a sintaxe do sistema de PLI Aleph (Srinivasan, 2000), descrito na Seção 4.3 na página 41, sendo que todas as informações necessárias para que ele possa construir os predicados encontram-se nesse arquivo. Essa informação pode ser dividida nas seguintes partes:

Parâmetros do Aleph São as declarações dos parâmetros do Aleph feitas pelo usuário no arquivo de entrada .param;

Declaração dos Modos A segunda parte que é criada pelo módulo *Kaeru* são as declarações dos modos que descrevem os predicados que serão utilizados pelo Aleph. Primeiro é feita a criação dos `modeh`, e para isso o sistema utiliza as informações declaradas no parâmetro `head` do arquivo .bk; em seguida o sistema cria todos os `modeb`, utilizando a informação do parâmetro `body` também do arquivo .bk;

Determinations A terceira parte é a criação das declarações dos determinations. Esse parâmetro descreve para o Aleph todos os predicados que ele poderá utilizar no processo de indução. Os determinations são criados utilizando a informação dos parâmetros `head` e `body` do arquivo .bk. Como descrito na Seção 4.3 na página 41, o determination tem a seguinte forma:

```
:- determination(Head/Arityh,Body/Arityb).
```

Para criar a primeira parte, que descreve no determination qual será a cabeça dos predicados e sua aridade, o sistema utiliza a informação do parâmetro `head`, enquanto que a segunda parte, que descreve qual o corpo que o predicado poderá ter e sua aridade, é retirada do parâmetro `body`;

Declaração dos Tipos A declaração dos tipos é feita pelo módulo *Kaeru* utilizando-se as informações disponíveis no arquivo .names. O sistema converte a declaração dos tipos que estão na sintaxe DSX para o formato aceito pela sintaxe do sistema

Aleph. São declarados automaticamente pelo sistema todos os atributos que foram utilizados nos predicados e seus respectivos tipos. Por exemplo, dada a seguinte declaração no arquivo `.names`:

```
outlook: nominal (sunny, overcast, rain).
temperature: integer.
```

O sistema declara no arquivo `.b`:

```
outlook(sunny).
outlook(overcast).
outlook(rain).
temperature(integer).
```

É importante lembrar que a declaração dos tipos, tal como do tipo `integer` no exemplo, não faz com que o Aleph realize nenhuma checagem de tipos. Nos sistemas relacionais, as declarações de tipos são necessárias apenas para limitar a busca realizada pelo sistema;

Conhecimento do Domínio A última parte criada pelo módulo *Kaeru* é a inclusão do conhecimento do domínio, que é necessário para que o sistema Aleph possa induzir conhecimento utilizando bases de dados originalmente proposicionais. Para isso o módulo *Kaeru* cria, para cada atributo fornecido no arquivo `.names` (exceto para o atributo classe e `att_id`) uma cláusula Prolog. Por exemplo, considerando o arquivo `voyage.names` — Tabela 5.2 na página 56 — o módulo *Kaeru* sempre cria as seguintes cláusulas:

```
outlook(Att_id,Outlook):-
    example(Att_id,Outlook,_,_,_), Outlook \= '?.
temperature(Att_id,Temperature):-
    example(Att_id,_,Temperature,_,_), Temperature \= '?.
humidity(Att_id,Humidity):-
    example(Att_id,_,_,Humidity,_,_), Humidity \= '?.
windy(Att_id,Windy):-
    example(Att_id,_,_,_,Windy,_), Windy \= '?.
```

O símbolo `?` é utilizado na sintaxe DSX para representar um atributo com valor desconhecido. Os predicados `Example/Aridade` tem sua aridade definida pelo número

de atributos (incluindo a classe) fornecidos no arquivo `.names`. O módulo *Kaeru* cria, também, para cada exemplo fornecido no arquivo `.data` um fato `Example`. Considerando o arquivo `.data` na Tabela 5.3 na página 56, o módulo *Kaeru* sempre cria os seguintes seis fatos:

```
example(0,sunny, 25, 72, yes, go).
example(1,sunny, 28, 91, yes, dont_go).
example(2,overcast, 23, 90, yes, go).
example(3,overcast, 29, 78, no, go).
example(4,rain, 22, 95, no, go).
example(5,rain, 19, 70, yes, dont_go).
```

Considerando as seguintes declarações de `body`, em que o usuário declarou diferentes predicados

```
body{outlook_temperature,outlook_humidity,outlook_humidity_temperature}
```

então o módulo conversor *Kaeru* cria, adicionalmente, as seguintes cláusulas:

```
outlook_temperature(Att_id,Outlook,Temperature):-
    example(Att_id,Outlook,Temperature,_,_,_),
    Outlook \= '?', Temperature \= '?'.
outlook_humidity(Att_id,Outlook,Humidity):-
    example(Att_id,Outlook,_,Humidity,_,_),
    Outlook \= '?', Humidity \= '?'.
outlook_humidity_temperature(Att_id,Outlook,Humidity,Temperature):-
    example(Att_id,Outlook,Temperature,Humidity,_,_),
    Outlook \= '?', Temperature \= '?', Humidity \= '?'.
```

5.2.4 Exemplo do Arquivo de Saída `.b`

Na Tabela 5.6 é mostrado o conteúdo do arquivo `.b` para o conjunto de exemplos `voyage` utilizando os arquivos `.names`, `.data`, `.param` e `.bk`, descritos neste capítulo, como arquivos de entrada.

5.2.5 Considerações Finais

Neste capítulo foi dada uma visão geral do módulo conversor *Kaeru* que faz parte do ambiente de PLI que está sendo desenvolvido para ser integrado ao ambiente computacional DISCOVER, o que permitirá incluir no DISCOVER algoritmos de aprendizado relacional. O módulo conversor *Kaeru*, do ambiente de PLI, tem como objetivo auxiliar no processo de transformação de bases de dados no formato atributo-valor, descritas na sintaxe DSX do DISCOVER, para o formato relacional utilizado por sistemas de PLI. Além disso, o módulo conversor *Kaeru* cria as declarações necessárias para executar o sistema de PLI Aleph. Experimentos realizados utilizando o módulo conversor *Kaeru* são apresentados nos próximos capítulos.

```

% Parâmetros do Aleph %
:- set(minpos,3).
:- set(clauselength,5).

% Declaração dos Modos %
:- modeh(1,class(+att_id,#class)).
:- modeb(*,outlook(+att_id,#outlook)).
:- modeb(*,temperature(+att_id,#temperature)).
:- modeb(*,humidity(+att_id,#humidity)).
:- modeb(*,windy(+att_id,#windy)).

% Determinations %
:- determination(class/2,outlook/2).
:- determination(class/2,temperature/2).
:- determination(class/2,humidity/2).
:- determination(class/2,windy/2).

% Declaração dos Tipos %
outlook(sunny).
outlook(overcast).
outlook(rain).
temperature(integer).
humidity(integer).
windy(yes).
windy(no).
class(go).
class(dont_go).

% Conhecimento do Domínio %
outlook(Att_id,Outlook):-
    example(Att_id,Outlook,_,_,_,_), Outlook \= '?.
temperature(Att_id,Temperature):-
    example(Att_id,_,Temperature,_,_,_), Temperature \= '?.
humidity(Att_id,Humidity):-
    example(Att_id,_,_,Humidity,_,_), Humidity \= '?.
windy(Att_id,Windy):-
    example(Att_id,_,_,_,Windy,_), Windy \= '?.

example(0,sunny, 25, 72, yes, go).
example(1,sunny, 28, 91, yes, dont_go).
example(2,overcast, 23, 90, yes, go).
example(3,overcast, 29, 78, no, go).
example(4,rain, 22, 95, no, go).
example(5,rain, 19, 70, yes, dont_go).

```

Tabela 5.6: Exemplo do arquivo de saída do módulo conversor *Kaeru*: `voyage.b`

Experimentos com Dados Naturais

Neste capítulo são apresentados os experimentos realizados utilizando bases de dados naturais, cujo formato original é o formato atributo-valor (proposicional). Esses conjuntos de dados são transformados para o formato relacional equivalente utilizando o módulo conversor *Kaeru*. Após a transformação, os conjuntos de dados são utilizados para realização de experimentos com Aleph, *i.é.*, utilizando aprendizado relacional. O objetivo desses experimentos é verificar a existência de conhecimento relacional nesses conjuntos de dados bem como verificar a possibilidade de utilizar o conhecimento induzido pelo algoritmo de aprendizado proposicional *See5* para incrementar o conhecimento do domínio utilizado no aprendizado relacional.

6.1 Conjuntos de Dados

Os experimentos foram realizados utilizando quatro conjuntos de dados naturais, no formato atributo-valor, obtidos no Repositório de dados UCI ¹, descritos a seguir.

¹<http://www.ics.uci.edu/mlearn/MLRepository.html>

6.1.1 Descrição

Além da descrição geral, para cada conjunto de dados é também mostrada uma tabela que descreve os atributos, nome e tipo, bem como o número de valores diferentes de cada atributo no conjunto de dados. Para atributos discretos, o número de possíveis valores do atributo é também especificado.

Breast-cancer Esse conjunto de dados foi disponibilizado pelo Dr. Willian H. Wolberg, Hospital Universidade Wisconsin, Madison. O problema é predizer se uma amostra de tecido da mama de pacientes é maligno (classe=maligno) ou benigno (classe=benigno). Essas amostras são avaliadas visualmente. Cada amostra foi associada a um vetor 9-dimensional em que cada componente está associado a um valor de 1 a 10. O valor 1 se refere ao estado normal e o valor 10 ao mais anormal. Diagnósticos malignos são confirmados pela retirada de uma amostra de tecido da mama da paciente realizando-se uma biópsia, enquanto os benignos são confirmados por biópsia ou por exames periódicos, dependendo da paciente — Tabela 6.1.

Número do Atributo	Nome do Atributo	#Valores Distintos		
		possíveis	atuais	tipo
#0	Clump Thickness	-	10	contínuo
#1	Uniformity of Cell Size	-	10	contínuo
#2	Uniformity of Cell Shape	-	10	contínuo
#3	Marginal Adhesion	-	10	contínuo
#4	Single Epithelial Cell Size	-	10	contínuo
#5	Bare Nuclei	-	10	contínuo
#6	Bland Chromatin	-	10	contínuo
#7	Normal Nucleoli	-	10	contínuo
#8	Mitoses	-	9	contínuo

Tabela 6.1: Breast-cancer – Descrição dos atributos

Breast-cancer2 Esse é um conjunto de dados de câncer de mama da UCI, disponibilizado pelo Instituto de Oncologia de Ljubljana. O problema é predizer a recorrência (classe=rec) ou não (classe=norec) de câncer de mama — Tabela 6.2.

Bupa Esse conjunto de dados foi disponibilizado por R. S. Forsyth para o Repositório da UCI. O problema é predizer se pacientes do sexo masculino têm (classe=1) ou não (classe=2) doenças no fígado, baseado em vários testes sanguíneos e quantidade de álcool consumido — Tabela 6.3.

Pima Esse conjunto de dados foi disponibilizado por V. Sigillito, Laboratório de Física Aplicada da Universidade Johns Hopkins para o Repositório da UCI. Todos os pacientes são mulheres com idade mínima de 21 anos de descendência indígena Pima

Número do Atributo	Nome do Atributo	#Valores Distintos		
		possíveis	atuais	tipo
#0	Age	-	44	contínuo
#1	Age-at-meno	-	3	nominal
#2	Tumor-size	-	23	contínuo
#3	Involved-nodes	-	18	contínuo
#4	Node-capsule	3	3	nominal
#5	Degree-of-malig	-	3	contínuo
#6	Breast	-	2	nominal
#7	Breast-Quadrant	6	6	nominal
#8	Irradiation	-	2	nominal

Tabela 6.2: Breast cancer2 – Descrição dos atributos

Número do Atributo	Nome do Atributo	#Valores Distintos		
		possíveis	atuais	tipo
#0	mcv	-	26	contínuo
#1	alkphos	-	78	contínuo
#2	sgpt	-	67	contínuo
#3	sgot	-	47	contínuo
#4	gammagt	-	94	contínuo
#5	drinks	-	16	contínuo

Tabela 6.3: Bupa – Descrição dos atributos

que moram próximas a Phoenix, Arizona, EUA. O problema é prever se a paciente seria classificada como diabética (classe=0), ou não (classe=1), segundo o critério estabelecido pela Organização Mundial de Saúde, fornecidos dados clínicos e laboratoriais — Tabela 6.4.

Número do Atributo	Nome do Atributo	#Valores Distintos		
		possíveis	atuais	tipo
#0	Number	-	17	contínuo
#1	Plasma	-	136	contínuo
#2	Diastolic	-	47	contínuo
#3	Triceps	-	51	contínuo
#4	Two	-	186	contínuo
#5	Body	-	248	contínuo
#6	Diabetes	-	517	contínuo
#7	Age	-	52	contínuo

Tabela 6.4: Pima – Descrição dos atributos

6.1.2 Características

A Tabela 6.5 mostra um resumo das características desses quatro conjuntos de dados. Para cada conjunto de dados são apresentados:

- # Exemplos: número de exemplos;

- #Duplicados ou Conflitantes: número e percentagem de exemplos duplicados (aparecem mais de uma vez) ou conflitantes (mesmos valores para todos os atributos, porém pertencentes a classes diferentes);
- # Atributos: número total de atributos bem como o número de atributos contínuos ou nominais (discretos);
- Classe e Classe%: valores da classe e percentagem de exemplos em cada classe;
- Erro da CM: erro cometido no caso dos exemplos serem classificados como sendo pertencentes à classe majoritária;
- Valores Desconhecidos: existência ou não de valores desconhecidos.

Conjunto de Dados	# Exemplos	#Duplicados ou Conflitantes (%)	# Atributos (cont.,nom.)	Classes	Classes %	Erro CM	Valores Desconhecidos
breast-cancer	699	8 (1,2%)	9 (9,0)	2	65,5%	34,5% sobre 2	Sim
				4	34,5%		
breast-cancer2	285	2 (0,7%)	9 (4,5)	rec	29,5%	29,5%	Sim
bupa	345	4 (1,2%)	6 (6,0)	no-rec	70,5%	sobre norec	Não
				1	42,0%	42,0%	
pima	769	1 (0,1%)	8 (8,0)	2	58,0%	sobre 2	Não
				0	65,0%	35,0%	
				1	35,0%	sobre 0	

Tabela 6.5: Resumo dos conjuntos de dados

6.2 Descrição dos Experimentos

Os experimentos foram realizados em duas etapas. Na primeira etapa foram gerados os modelos proposicionais utilizando-se o indutor *See5* e na segunda etapa foram gerados os modelos relacionais utilizando o sistema de PLI Aleph.

6.2.1 *See5*

O *See5* (Rulequest-Research, 1999) é uma ferramenta de mineração de dados que induz regras e árvores de decisão. Neste trabalho o *See5* foi utilizado para induzir hipóteses descritas por regras de decisão. O *See5* é uma versão melhorada do algoritmo *C4.5*, a qual incorporou-se, além da interface gráfica, melhorias tais como o aumento da velocidade de geração de regras, o aumento da precisão das regras geradas e a utilização de uma quantidade muito menor de memória. Resumidamente, pode-se subdividir os recursos disponíveis no *See5* em três funções:

- Construir o classificador: possibilita selecionar o tipo de classificador a ser utilizado e a definição de parâmetros adicionais. Quando o *See5* é executado com os valores *default* dos parâmetros, é construída uma árvore de decisão. Porém, é oferecida a opção para a geração de regras de decisão (*Ruleset*). O *See5* permite ainda a opção *Cross-validation* que possibilita a utilização de *k-fold cross-validation*, além de várias outras opções;
- Usar o classificador: uma vez construído o classificador, um interpretador interativo pode ser empregado para a classificação de novos exemplos;
- Realizar referência cruzada entre o classificador e os exemplos: esse recurso permite que possam ser identificadas quais partes do classificador são relevantes para um determinado exemplo. Também, selecionando-se uma folha da árvore ou uma regra é possível visualizar quais exemplos são mapeados pela folha ou regra selecionada.

6.2.2 Aprendizado Proposicional

Os mesmos experimentos foram realizados para os quatro conjuntos de dados descritos na Tabela 6.5, para os quais foram induzidos com *See5* modelos proposicionais na forma de regras de decisão. Os resultados obtidos encontram-se na Tabela 6.6. Para cada experimento foram medidos o erro aparente e o erro verdadeiro (média e desvio padrão) usando *10-fold cross-validation*². A última coluna, No Regras, refere-se ao número de regras induzidas pelo *See5* utilizando todos os exemplos.

Conjunto de Dados	Erro CM	Erro Aparente	Erro 10-cv	No Regras
breast-cancer	34,5% sobre 2	3,1	5,1±1,0	06
breast-cancer2	29,5% sobre norec	15,9	22,5±2,4	7
bupa	42,0% sobre 2	12,5	32,2±2,4	12
pima	35,0% sobre 0	18,6	25,5±1,5	15

Tabela 6.6: Resultados dos experimentos com *See5*

²O *10-fold cross-validation* é realizado dividindo-se os dados em 10 subconjuntos mutuamente exclusivos de casos aproximadamente do mesmo tamanho. O indutor é treinado e testado 10 vezes, cada vez testado com um subconjunto (10% dos exemplos) e treinado com o restante do conjunto de dados (90% dos exemplos). A estimativa da precisão é a média das precisões estimadas usando-se os 10 subconjuntos.

6.2.3 Aprendizado Relacional

Os quatro conjuntos de dados no formato atributo-valor foram transformados para o formato relacional equivalente utilizando o módulo conversor *Kaeru* e submetidos para o Aleph. Os experimentos foram realizados para todos os conjuntos de dados utilizando os parâmetros padrão do Aleph, exceto para o parâmetro `minpos`, o qual foi definido como

```
:-set(minpos,2).
```

para que o Aleph induza cláusulas que cubram no mínimo dois exemplos positivos. Como mencionado, um dos objetivos desses experimentos é verificar a existência de conhecimento relacional nesses conjuntos de dados. Além disso, pretende-se verificar a possibilidade de utilizar, parcialmente, o conhecimento induzido pelo *See5* para aumentar o conhecimento do domínio para realizar aprendizado relacional.

Os resultados obtidos utilizando Aleph são mostrados na Tabela 6.7. A coluna Experimento, mostra os resultados utilizando somente os dados no formato atributo-valor transformados pelo módulo *Kaeru* para o formato relacional (sem-ad), e utilizando conhecimento do domínio adicional (com-ad), ou seja:

- sem-ad: sem nenhum conhecimento do domínio adicional;
- com-ad: adicionando como conhecimento os predicados `menor_igual/2` e `maior/2`, definidos da seguinte forma:

```
menor_igual(X,Y):-  
    var(Y), !,  
    X = Y.  
menor_igual(X,Y):-  
    number(X), number(Y),  
    X <= Y.  
  
maior(X,Y):-  
    var(Y), !,  
    X = Y.  
maior(X,Y):-  
    number(X), number(Y),  
    X > Y.
```

A coluna Classe Positiva mostra a classe utilizada como classe positiva no aprendizado do conceito. Em todos os experimentos, cada classe foi utilizada como positiva e a outra como negativa.

Conjunto de Dados	Experimento	Classe Positiva (%)	No Regras	Erro Aparente	Erro 10-cv
breast-cancer	sem-ad	2 (65,5%)	18	0,6	5,9±2,1
		4 (34,5%)	24	0,7	6,6±1,9
	com-ad	2 (65,5%)	00	65,5	64,7±2,0
		4 (34,5%)	06	10,7	11,5±2,8
breast-cancer2	sem-ad	norec (70,5%)	37	3,9	27,6±5,7
		rec (29,5%)	25	6,7	28,1±4,3
	com-ad	norec (70,5%)	14	50,2	52,8±7,3
		rec (29,5%)	04	24,6	37,9±22,0
bupa	sem-ad	dois (58,0%)	64	11,6	52,8±5,2
		um (42,0%)	47	11,6	38,1±7,5
	com-ad	dois (58,0%)	03	54,8	54,7±2,6
		um (42,0%)	04	38,3	39,6±2,6
pima	sem-ad	um (35,0%)	93	8,1	36,4±3,2
		zero (65,0%)	06	10,7	47,0±4,9
	com-ad	um (35,0%)	02	34,2	34,9±0,3
		zero (65,0%)	01	64,5	65,1±0,3

Tabela 6.7: Resultados dos experimentos com Aleph - Etapa 1

Como pode ser observado na Tabela 6.7, os resultados dos experimentos realizados utilizando `menor_igual/2` e `maior/2` como conhecimento adicional (com-ad), não melhora a precisão do modelo. Este foi um resultado inesperado pois, aparentemente, esse conhecimento não deveria influir negativamente no modelo induzido. Assim, resolveu-se investigar melhor como o Aleph utiliza esse conhecimento adicional.

Para ilustrar como Aleph trabalha com atributos contínuos usando como conhecimento de fundo a definição `menor_igual/2`, será utilizado um exemplo muito simples. Considere o conjunto de exemplos na Tabela 6.8.

Ex.	At-1	Classe
1	2	+
2	3	+
3	4	+
4	6	-
5	7	-
6	8	-

Tabela 6.8: Conjunto de exemplos `teste1`

Claramente, uma possível hipótese correspondente a esses exemplos seria: Se $At-1 \leq 4$ então classe = +.

Considere a declaração dos exemplos positivos e negativos nos arquivos correspondentes `teste1.f` e `teste1.n` na Tabela 6.9.

E o seguinte arquivo `teste1.b` com o conhecimento de fundo e as diretivas para a execução de Aleph com os parâmetros padrões, exceto a definição de `minpos` para uma cobertura

Ex.(+)	trens.f	Ex.(-)	trens.n
1	class(1,pos).	4	class(4,neg).
2	class(2,pos).	5	class(5,neg).
3	class(3,pos).	6	class(6,neg).

Tabela 6.9: Arquivos teste1.f e teste1.n

de no mínimo dois exemplos positivos:

```
:- set(minpos,2).
:-modeh(1,class(+att_id,+class)).
:-modeb(*,peso(+att_id,-peso)).
:-modeb(*,menor_igual(+peso,#peso)).

:-determination(class/2,peso/2).
:-determination(class/2,menor_igual/2).

peso(num2).    class(pos).    class(neg).

menor_igual(X,Y):-
    var(Y), !,
    X = Y.
menor_igual(X,Y):-
    number(X), number(Y),
    X <= Y.
```

```
peso(1,2).  peso(2,3).  peso(3,4).  peso(4,6).  peso(5,7).  peso(6,8).  peso(7,4).
```

Neste caso, a execução de Aleph:

```
?- read_all(teste1).
?- induce.
```

induz uma teoria vazia, com

```
[theory]

[Training set performance]
      Actual
      +      -
+ 0      0      0
Pred
- 3      3      6
      3      3      6
```

```
Accuracy = 0.5 [Training set summary] [[0,0,3,3]] [time taken]
[0.03000000000000011] [total clauses constructed] [6] yes
```

Analisando a execução, é possível observar que Aleph constrói, para o primeiro exemplo positivo, a *bottom clause*

```
class(A,B) :-
    peso(A,C), menor_igual(C,2).
```

que não cobre outro exemplo positivo, pois todos eles têm valor maior que 2. Assim, é construída a segunda *bottom clause*

```
class(A,B) :-
    peso(A,C), menor_igual(C,3).
```

Entretanto, Aleph não consegue “ver” o exemplo positivo `class(1,pos)` com `peso(1,2)` que já foi utilizado antes. O mesmo acontece com a construção da próxima *bottom clause* utilizando `class(3,pos)`, e nenhuma teoria é induzida. Entretanto, caso o exemplo positivo `class(3,pos)` seja declarado como primeiro exemplo no arquivo `teste1.f`, Aleph encontra a seguinte teoria, pois ele “enxerga” os exemplos positivos 1 e 2.

```
[theory]
```

```
[Rule 1] [Pos cover = 3 Neg cover = 0] class(A,B) :-
    peso(A,C), menor_igual(C,4).
```

```
[Training set performance]
```

Actual			
	+	-	
Pred			
+	3	0	3
-	0	3	3
	3	3	6

```
Accuracy = 1.0 [Training set summary] [[3,0,0,3]] [time taken]
[0.009999999999999801] [total clauses constructed] [3] yes
```

Essa teoria também é induzida incrementando os exemplos positivos na Tabela 6.9, com mais um exemplo `class(7,pos)`, como mostrado na Tabela 6.10, e incluindo no arquivo `teste1.b` a informação correspondente `peso(7,4)`.

Nesse caso, quando Aleph constrói a *bottom clause* para o exemplo positivo 3, ele enxerga o exemplo positivo 7. Assim, a regra cobre dois exemplos positivos, que é o número mínimo de exemplos que uma regra deve cobrir, como definido por `minpos`. Aleph verifica se há mais exemplos positivos cobertos por essa regra, que são todos, e finaliza com

Ex.(+)	trens.f	Ex.(-)	trens.n
1	class(1,pos).	4	class(4,neg).
2	class(2,pos).	5	class(5,neg).
3	class(3,pos).	6	class(6,neg).
7	class(7,pos).		

Tabela 6.10: Arquivos `teste1.f` com `class(7,pos)` adicionado e `teste1.n`

[theory]

[Rule 3] [Pos cover = 4 Neg cover = 0] class(A,B) :-
 peso(A,C), menor_igual(C,4).

[Training set performance]

	Actual		
	+	-	
Pred			
+	4	0	4
-	0	3	3
	4	3	7

Accuracy = 1.0 [Training set summary] [[4,0,0,3]] [time taken]
 [0.03999999999999991] [total clauses constructed] [9] yes

Esse comportamento de Aleph explica porque os resultados dos experimentos descritos na Tabela 6.7 são piores utilizando esse conhecimento adicional. Portanto, para atributos contínuos, é melhor o usuário definir os limites dessas relações. No exemplo considerado, definindo a relação `menor_igual/2` como

```
menor_igual(X,4):-
  number(X),
  X <= 4.
```

permite a Aleph induzir a teoria em todos os casos. Essa idéia de definir os limites para relações do tipo `menor_igual`, e `maior` é utilizada nos próximos experimentos realizados neste trabalho. Para verificar a viabilidade do uso do conhecimento induzido pelo *See5* como conhecimento do domínio adicional para o aprendizado relacional, foi decidido realizar esses experimentos com o conjunto de dados **breast-cancer** utilizando 2 como classe positiva, pois esse conjunto apresenta os melhores resultados sem uso adicional de conhecimento do domínio — Tabela 6.7 — e também apresenta os melhores resultados com os experimentos realizados com *See5* — Tabela 6.6.

Observando as regras induzidas pelo *See5* com o conjunto **breast-cancer**, foram verificadas faixas de valores que conseguem separar as classes 2 e 4. Esses atributos e valores

```

Bare Nuclei <= 2 para classe=2 e
Bare Nuclei > 2 para classe=4.

Clump Thickness <= 4 para classe=2 e
Clump Thickness > 4 para classe=4.

Uniformity of Cell Shape <= 2 para classe=2 e
Uniformity of Cell Shape > 2 para classe=4.

```

Tabela 6.11: Atributos e valores que separam classes no conjunto breast-cancer

são descritos na Tabela 6.11. Com esse conhecimento foram definidos predicados para serem incluídos no conhecimento do domínio. Esses predicados foram incluídos um por vez, induzindo o modelo correspondente. A denominação dada a esses novos experimentos e o conhecimento incluído em cada um deles é descrito a seguir

- com barnucl: adicionando somente o conhecimento definido por `barenucl/3`:

```

barenucl(Att_id,alta,2):-
    bare_nuclei(Att_id,BARE_NUCLEI),
    maior(BARE_NUCLEI,2).
barenucl(Att_id,baixa,2):-
    bare_nuclei(Att_id,BARE_NUCLEI),
    menor_igual(BARE_NUCLEI,2).

```

- com barnucl e clump: além de `barenucl/3` adicionando o conhecimento definido por `clumpthick`:

```

clumpthick(Att_id,alta,4):-
    clump_thickness(Att_id,CLUMP_THICKNESS),
    maior(CLUMP_THICKNESS,4).
clumpthick(Att_id,baixa,4):-
    clump_thickness(Att_id,CLUMP_THICKNESS),
    menor_igual(CLUMP_THICKNESS,4).

```

- com barnucl clump e cell: além de `barenucl/3` e `clumpthick/3`, adicionado o conhecimento definido por `cellshape/3`:

```

cellshape(Att_id,alta,2):-
    uniformity_of_cell_shape(Att_id,UNIFORMITY_OF_CELL_SHAPE),
    maior(UNIFORMITY_OF_CELL_SHAPE,2).
cellshape(Att_id,baixa,2):-
    uniformity_of_cell_shape(Att_id,UNIFORMITY_OF_CELL_SHAPE),
    menor_igual(UNIFORMITY_OF_CELL_SHAPE,2).

```

Os resultados obtidos estão resumidos na Tabela 6.12, onde pode ser observado que o melhor resultado foi obtido utilizando a classe 2 como positiva e o conhecimento do domínio incluído com os três predicados.

Conjunto de Dados	Experimento	Classe Positiva (%)	No Regras	Erro Aparente	Erro 10-cv
breast-cancer	com <code>barnucl/3</code>	2 (65,5%)	05	9,9	4,6±1,7
	com <code>barnucl/3</code>	2 (65,5%)	05	4,7	
	e <code>clump/3</code>				
	com <code>barnucl/3</code>	2 (65,5%)	07	3,6	
	<code>clump/3</code> e <code>cell/3</code>				
	com <code>barnucl/3</code>	4 (34,5%)	11	7,0	
	<code>clump/3</code> e <code>cell/3</code>				

Tabela 6.12: Resultados dos experimentos com Aleph - Etapa 2

Nos resultados experimentais apresentados nesta seção, bem como no Capítulo 7, a avaliação do erro verdadeiro foi calculado apenas para os experimentos que apresentam o menor erro verdadeiro, pois o *10-fold cross-validation* é extremamente custoso de ser calculado manualmente.

6.2.4 Análise dos Resultados

Como pode ser observado na Tabela 6.12 os resultados dos experimentos incluindo conhecimento do domínio adicional fornecido pela hipótese induzida com *See5*, apresentam bons resultados. A cada novo experimento, ou seja, a cada novo conhecimento adicionado, esse novo conhecimento foi incorporado na hipótese induzida, o erro aparente diminuiu e o número de regras induzidas manteve-se bem limitado. Para o experimento com o menor erro aparente, classe positiva 2 com `barnucl/3`, `clump/3` e `cell/3` adicionados ao conhecimento do domínio, foi também calculado o erro verdadeiro utilizando *10-fold cross validation*, o qual é menor que o obtido sem conhecimento do domínio adicional — Tabela 6.7 na página 73. Além disso, o número de regras induzidas (7 regras) é bem menor que o número de regras induzidas sem-ad (18 regras). O número de regras induzidas utilizando como classe positiva a classe 4, também diminuiu de 24 regras sem o uso de conhecimento do domínio adicional para 11 regras.

6.3 Considerações Finais

Neste capítulo foi apresentada uma série de experimentos utilizando conjuntos de dados naturais da área médica, originalmente no formato atributo-valor. Esses conjuntos de

dados foram transformados para o formato relacional equivalente utilizando o módulo conversor *Kaeru*. Após a transformação os conjuntos de dados foram submetidos ao sistema de PLI Aleph para verificar a existência de conhecimento relacional. Além disso, foi utilizado o modelo proposicional induzido pelo *See5* como conhecimento adicional do domínio. Utilizando esse conhecimento adicional foi possível melhorar os modelos induzidos por Aleph, mostrando que conhecimento proposicional pode auxiliar na definição de conhecimento adicional do domínio para o aprendizado relacional. Isso é particularmente importante quando não é possível ter um especialista do domínio para auxiliar nessa tarefa, já que um bom conhecimento do domínio é fundamental para a descoberta de conhecimento utilizando sistemas de PLI.

Estudo de Caso

Neste capítulo é apresentado um estudo de caso realizado com uma base de dados real, originalmente no formato atributo-valor, com o objetivo de extrair conhecimento relacional utilizando o sistema de PLI Aleph. A base de dados deste estudo de caso foi anteriormente utilizada para extrair conhecimento, mas utilizando AM simbólico proposicional. Assim, o objetivo é verificar a existência de conhecimento relacional nessa base, além da utilização do conhecimento proposicional adquirido nos trabalhos anteriormente realizados como conhecimento adicional do domínio para os experimentos relacionais.

7.1 Fases do Estudo de Caso

A realização de um estudo de caso pode ser dividido nas seguintes fases, como apresentado na Figura 7.1

1. Entrevista com especialista e coleta dos dados;
2. Preparação dos dados;
3. Escolha do conhecimento do domínio;

Figura 7.1: Fases do estudo de caso

4. Modelagem dos dados;
5. Análise dos Resultados.

As fases de estudo de caso com conjuntos de dados reais envolvem, normalmente, na primeira fase uma entrevista com o especialista. Durante a entrevista com o especialista são interpretadas as informações disponíveis na base de dados, são discutidas as melhores formas para abordagem do problema, quais dados são relevantes e devem ser considerados, como agrupá-los, como as classes são definidas, entre outras questões. Após a entrevista os dados armazenados são coletados. Os dados referente a este estudo de caso foram cedidos pelos Dr. Sandro C. Esteves e coletados da Androfert¹ e encontravam-se em formato eletrônico *Word for Windows*, no qual cada documento é o exame de um paciente com todos os dados referentes ao espermograma e o processamento de sêmen. Como já citado, este conjunto de dados foi estudado anteriormente, permitindo que neste trabalho fosse dispensada essa primeira fase.

A fase de preparação dos dados envolveu a limpeza e transformação dos dados do formato atributo-valor para o formato relacional do sistema Aleph. Para essa transformação foi utilizado o módulo conversor *Kaeru*, descrito na Seção 5.2 na página 57. A terceira fase é uma das mais importantes. Nessa fase são escolhidos os predicados que irão compor o conhecimento do domínio. O auxílio do especialista nessa tarefa é muito importante. A fase de modelagem dos dados consiste na aplicação das técnicas de AM relacional e foi realizada utilizando o sistema Aleph. A quarta e última fase é a análise dos resultados

¹Centro de Referência em Reprodução Humana, Campinas - SP — <http://www.androfert.com.br>

obtidos na modelagem. Nessa fase a cooperação do especialista do domínio é fundamental. Como pode ser observado na Figura 7.1, a fase de escolha do conhecimento do domínio pode ser retomada a medida que os dados são modelados ou os resultados analisados. Muitas vezes é necessário realizar modificações no conhecimento do domínio e voltar ao passo de modelagem até obter bons resultados.

A base de dados real utilizada neste estudo de caso, está relacionada com o processamento de sêmen para a reprodução assistida, descrita a seguir.

7.2 O Problema do Processamento de Sêmen

Milhares de crianças nascem a cada ano. Um número cada vez maior desses nascimentos são originados a partir de casais que antes eram considerados inférteis, graças aos grandes avanços conseguidos no campo do diagnóstico e tratamento da infertilidade conjugal. A dificuldade para ter filhos é muito mais comum do que se imagina; de cada seis casais, um terá dificuldade para tê-los.

Para que ocorra a fecundação do óvulo, é necessário que exista um número suficiente de espermatozóides, e que os mesmos tenham movimentação progressiva no interior do aparelho reprodutor feminino, a fim de que esses espermatozóides rápidos possam atingir e penetrar no interior do óvulo, dando origem à gravidez. Além disso, é fundamental que exista um número adequado de espermatozóides com tamanho e formato normais. Espermatozóides que apresentam estrutura anormal (morfologia), como duas cabeças, cauda enrolada, cabeça muito grande ou muito pequena, entre outros, não serão capazes de fecundar o óvulo. Porém, graças aos avanços da medicina reprodutiva nos últimos tempos, mesmo os casos mais graves de infertilidade podem ser resolvidos pelas técnicas de reprodução assistida.

Em todas as situações onde houver indicação de reprodução assistida uma ou mais análises deverão ser realizadas no sêmen. O espermograma é um exame laboratorial que determina o número, a movimentação (motilidade) e o formato dos espermatozóides. Além disso, o espermograma verifica se a quantidade de líquido seminal (volume) é normal, se há sinais de infecção e várias outras medidas. Porém, existe outro exame que avalia com maior precisão a qualidade do sêmen; esse exame é chamado de processamento de sêmen diagnóstico. Entretanto, contrariamente ao espermograma, o processamento de sêmen é bastante caro e pode elevar o custo do exame em aproximadamente 80% em relação à um espermograma. Essa elevação de custos deve-se, principalmente, à necessidade de equipamentos especiais, mão de obra qualificada e tempo gasto para realização dos exames.

O processamento de sêmen pode separar os melhores espermatozóides, remover toxinas e contaminantes, melhorar o potencial fértil dos espermatozóides e ainda determinar com maior precisão o número de espermatozóides recuperados de boa qualidade. O número de espermatozóides recuperados é um dos critérios que auxilia na escolha da técnica de reprodução assistida mais apropriada para o casal.

São utilizadas três técnicas no tratamento para reprodução assistida:

1. Inseminação Intra Uterina (IUI) - nessa técnica o esperma purificado é injetado no interior do útero;
2. Fertilização In Vitro (FIV) - técnica também conhecida como bebê de proveta, envolve a fecundação do óvulo pelo espermatozóide no laboratório e após a formação do embrião esse é colocado no útero;
3. Injeção Intracitoplasmática do Espermatozóide no Óvulo (ICSI) - nessa técnica basta um único espermatozóide vivo para ser injetado em cada óvulo utilizando um microscópio de alta potência. A Figura 7.2 mostra o momento da realização do ICSI; nessa figura é possível ver as duas micropipetas do microscópio. Do lado esquerdo está a micropipeta que segura o óvulo e do lado direito a micropipeta de injeção injetando o espermatozóide no interior do óvulo.

Figura 7.2: Injeção Intracitoplasmática do Espermatozóides no Oócito - ICSI

Os espermatozóides são classificados em graus, dependendo da sua motilidade (movimentação):

- Grau A - espermatozóides que se movem rapidamente (maior grau de motilidade);

- Grau B - espermatozóides que se movem mais lentamente (apresentam um menor grau de motilidade);
- Grau C - espermatozóides que se movem em círculos;
- Grau D - espermatozóides imóveis.

Os graus de motilidade são atributos medidos tanto no espermograma como no processamento de sêmen. Porém, como já citado, o processamento de sêmen, apesar de mais preciso, é bem mais caro que o espermograma. Assim, além do interesse em extrair e avaliar o conhecimento adquirido, um dos interesses do estudo desse tema é tentar prever qual será a quantidade de espermatozóides com boa motilidade recuperados pelo processamento de sêmen antes mesmo da realização desse exame, a partir de exames menos custosos como o espermograma. Assim, utilizamos as medidas que são realizadas durante o processamento de sêmen para a determinação das classes e as medidas realizadas no espermograma são fornecidas como atributos ao algoritmo de indução. Em outras palavras, todos os pacientes nessa base de dados realizaram ambos exames laboratoriais: espermograma (baixo custo) e processamento de sêmen (alto custo). Para cada paciente, a classe é determinada pelo resultado obtido no exame de processamento de sêmen, enquanto que os valores dos atributos correspondem às diversas medidas realizadas.

Durante a fase de entrevistas com o especialista realizada nos estudos anteriores (Lee, 2000; Ferro, 2002; Ferro et al., 2002) foram definidas três classes, baseadas nas técnicas utilizadas para a reprodução assistida:

- Classe ICSI: $x < 1$;
- Classe FIV: $1 \leq x < 5$;
- Classe IUI: $x \geq 5$.

A variável x representa milhões de espermatozóides por mililitro (ml) recuperados pelo processamento de sêmen. O especialista sugeriu ainda que os valores usados na classificação dos exemplos fossem considerados da seguinte forma:

- A - considerar apenas a percentagem de espermatozóides classificados como de motilidade grau A no processamento de sêmen;
- AB - considerar a percentagem de espermatozóides classificados como de motilidade grau A e grau B.

Assim, para cada exemplo foram calculados os valores A e AB, gerando dois conjuntos de dados denominados **proc-ad** e **proc-abd**, respectivamente. Nos estudos anteriores, os experimentos com essas duas bases de dados foram conduzidos separadamente e o conjunto de dados que apresentou melhores resultados foi **proc-ad**. Baseados nesses resultados, neste estudo estamos utilizando apenas o conjunto de dados **proc-ad**. A sigla **proc-ad** é a forma resumida para processamento de sêmen diagnóstico que considera espermatozóides com motilidade grau A.

7.3 Descrição dos Conjuntos de Dados

A seguir é descrito o conjunto completo dos atributos do espermograma pertencentes ao conjunto de dados do processamento de semên. A Tabela 7.1 mostra o nome do atributo, sua descrição e tipo (contínuo ou discreto). Para atributos discretos são mostrados o número de valores possíveis e o número atual de valores encontrados no conjunto de dados. Vale observar que se o número atual encontrado no conjunto de dados é maior que o número de valores possíveis significa que há exemplos com valores desconhecidos para esse atributo.

Nome do Atributo	Descrição	#Valores Distintos		
		possíveis	atuais	tipo
idade	Idade do paciente	*	31	contínuo
medico	Médico que solicitou o exame	9	9	discreto
local	Local de coleta da amostra de sêmen:	2	2	discreto
	sala de coleta do laboratório ou em casa			
metodo	Método utilizado para coleta:	2	2	discreto
	masturbação ou outros			
ejaculado	Ejaculação total ou parcial	2	3	discreto
hora	Período da coleta da amostra de sêmen:	2	3	discreto
	manhã ou tarde			
processamento	Processamento do sêmen realizado após esta quantidade de minutos após a coleta	*	19	contínuo
tempo-abs	Tempo de abstinência	*	12	contínuo
volume	Volume do sêmen coletado	*	65	contínuo
cor	Cor do sêmen coletado: translúcido,normal (branco-opalescente) ou amarelo-opalescente	3	4	discreto
odor	Odor do sêmen coletado:	3	3	discreto
	característico, forte ou urina			
ph	Ph do sêmen coletado	*	14	contínuo
viscosidade	Viscosidade do sêmen coletado:	2	2	discreto
	normal ou amentada			
liquefação	Liquefação do sêmen coletado:	2	2	discreto
	completa ou incompleta			
concentracao	Concentração de espermatozóide por ml	*	313	contínuo
class.a	Classificação da motilidade grau A	*	68	contínuo
class.b	Classificação da motilidade grau B	*	78	contínuo
class.c	Classificação da motilidade grau C	*	45	contínuo

continuação na próxima página

<i>continuação da página anterior</i>				
Nome do Atributo	Descrição	#Valores Distintos		
		possíveis	atuais	tipo
class_d	Classificação da motilidade grau D	*	79	contínuo
vitalidade	% de espermatozóides vivos	*	68	contínuo
deteccao_leu	Detecção de leucócitos no sêmen: positivo ou negativo	2	3	contínuo
num_leu	Número de leucócitos	*	96	contínuo
num_cel	Número de células germinativas imaturas	*	195	contínuo
krueger	Morfologia estrita de krueger - % de formas normais	*	31	contínuo
hp	Teste hipo-osmótico - % de inchados	*	61	contínuo
frutose	Determinação de frutose no plasma seminal: presente, ausente ou não solicitada	3	4	contínuo

Tabela 7.1: Processamento de sêmen – Descrição dos atributos

Do conjunto de 26 atributos — Tabela 7.1 —, o especialista considerou que sete deles não são relevantes (**medico**, **local**, **metodo**, **ejaculado**, **deteccao_leu**, **num_cel** e **frutose**). Assim, eles foram retirados do conjunto de dados, o qual ficou descrito por 19 atributos. A Tabela 7.2 mostra o resumo desse conjunto, que foi utilizado para realizar os experimentos.

Conjunto de Dados	# Exemplos	#Duplicados ou Conflitantes (%)	# Atributos (cont.,nom.)	Classes	Classes %	Erro CM	Valores Desconhecidos
proc_ad	278	0 (0%)	19 (14,5)	ICSI FIV IUI	40,5% 22,5% 37,0%	59% sobre ICSI	Sim

Tabela 7.2: Resumo do conjunto de dados processamento de sêmen utilizado nos experimentos

7.4 Descrição dos Experimentos

O conjunto original de dados utilizado neste estudo de caso encontrava-se no formato atributo-valor e foi transformado para o formato relacional equivalente utilizando o módulo conversor *Kaeru*. Após a conversão, o conjunto de dados foi submetido ao sistema de PLI Aleph. Todos os experimentos utilizam os parâmetros *default* do Aleph, exceto para o parâmetro **minpos** que foi definido como

```
:-set(minpos,2).
```

pois o parâmetro padrão `:-set(minpos,1)` do Aleph somente induz, para essa base de dados, uma teoria contendo apenas fatos.

Foram realizados diversos experimentos, os quais podem ser divididos em duas etapas. Na primeira etapa, os experimentos foram realizados utilizando como conhecimento adi-

cional conhecimento fornecido pelo especialista. Na segunda etapa, foi também utilizado como conhecimento adicional, conhecimento extraído desse conjunto de dados utilizando o algoritmo de aprendizado proposicional *See5*.

7.4.1 Etapa 1

Os resultados obtidos nesta etapa de testes são mostrados na Tabela 7.3. A coluna Experimento mostra os resultados utilizando somente os dados correspondentes aos 278 exemplos (sem-ad) e utilizando conhecimento adicional do domínio (com-const) do especialista. Esse conhecimento é definido por três predicados, sugeridos pelo especialista, para construir novos atributos, definidos para cada exemplo por

```

motilidade = class_a + class_b + class_c
mot_prog   = class_a + class_b
concentracao_total = concentracao x volume

```

Na tabela é mostrado também a classe utilizada como classe positiva no aprendizado do conceito, o número de regras induzidas e o erro aparente do modelo.

Experimento	Classe Positiva (%)	No Regras	Erro Aparente
1.1	FIV (22,5%)	21	3,6
sem-ad	ICSI(40,5%)	34	4,0
	IUI (37,0%)	32	2,2
	FIV (22,5%)	22	2,5
1.2	FIV (22,5%)	22	2,5
com-const	ICSI(40,5%)	35	3,5
	IUI (37,0%)	31	1,8
	FIV (22,5%)	22	2,5
1.3	ICSI + FIV (63,0%)	42	2,5
1.4	ICSI + FIV (63,0%)	42	2,5
com-const	FIV (22,5%)	22	2,5

Tabela 7.3: Resultados dos experimentos com o conjunto processamento de sêmen - Etapa 1

Os dois primeiros experimentos na Tabela 7.3 (Experimento 1.1 e 1.2) foram realizados considerando cada uma das três classes possíveis como classe positiva, e os exemplos pertencentes as outras duas classes como exemplos negativos. O Experimento 1.2 é semelhante ao 1.1, mas introduzindo o conhecimento do especialista para a construção de novos atributos. Os dois últimos experimentos (Experimento 1.3 e 1.4 na Tabela 7.3) são, respectivamente, semelhantes aos anteriores, mas considerando como classe positiva os exemplos rotulados ICSI e FIV, seguindo a sugestão do especialista.

Na Tabela 7.4 é mostrado, para cada um desses experimentos, o número de regras que cobrem i exemplos, o número total de regras na hipótese induzida ($\#$ Regras), o número total de exemplos positivos cobertos pela hipótese ($\#$ Ex. Cobertos), o número médio de exemplos coberto pelas regras que constituem a hipótese (Média) e o desvio padrão. Por exemplo, a hipótese induzida no Experimento 1.1 utilizando a classe FIV como classe positiva consiste de 21 regras, tal que 12 dessas regras cobrem 2 exemplos, 4 regras cobrem 3 exemplos e assim por diante; o número total de exemplos positivos cobertos pela hipótese é 57 e o número médio de exemplos cobertos pelas regras é 2,7 com desvio padrão 1,0.

Cobre	Experimento							
	1.1 (FIV)	1.1 (ICSI)	1.1 (IUI)	1.2 (FIV)	1.2 (ICSI)	1.2 (IUI)	1.3 (ICSI+FIV)	1.4 (ICSI+FIV)
2	12	10	11	13	9	9	5	6
3	4	9	6	4	12	8	9	7
4	4	9	3	4	8	3	4	4
5	1	3	4	1	3	5	6	7
6		2	5		2	3	3	3
7		1	2		1	1	7	7
8							4	4
9							4	4
#Regras	21	34	31	22	35	31	42	42
#Ex. Cobertos	57	117	116	59	120	104	176	219
Média	2,7	3,4	3,7	2,7	3,4	3,3	4,2	5,2
Desvio Padrão	1,0	1,3	1,8	0,9	0,7	1,7	2,3	2,7

Tabela 7.4: Número médio de exemplos cobertos pelas regras - Etapa 1

Como pode ser observado, o número de regras induzidas é um pouco alto, exceto para os Experimentos 1.1 e 1.2 utilizando FIV como classe positiva. Entretanto, além dessa classe ser a classe minoritária do conjunto de exemplos (22,5%), a maioria das regras cobrem somente 2 exemplos — Tabela 7.4.

7.4.2 Etapa 2

Nesta segunda etapa, o objetivo é introduzir mais conhecimento do domínio utilizando parte do conhecimento induzido pelo *See5*. A Tabela 7.5 mostra os resultados obtidos.

O algoritmo *See5* consegue manipular bem atributos contínuos, achando bons pontos de corte, os quais podem ser utilizados para incrementar o conhecimento do domínio. Isto é muito útil no caso do Aleph, como mostrado na Seção 6.2.3 na página 72. A fim de observar o efeito desse conhecimento adicional, foi decidido incorporá-lo em várias fases, descritas a seguir.

- No Experimento 2.1 foi adicionado o conhecimento do *See5*, denominado com-ad1,

Experimento	Classe Positiva (%)	No Regras	Erro Aparente	Erro 10-cv
2.1 com-ad1	IUI (37,0%)	31	1,8	
2.2 com-ad2	IUI (37,0%)	27	1,8	
2.3 com-ad3	IUI (37,0%)	26	2,5	
2.4 com-ad2	ICSI(40,5%)	20	2,2	27,3±7,1
2.5 com-ad1 e com-const	IUI (37,0%)	30	2,2	
2.6 com-ad4 e com-const	IUI (37,0%)	24	1,8	
2.7 com-ad4 e com-const	ICSI(40,5%)	21	1,1	23,0±7,8
2.8 com-ad5 e com-const	ICSI + FIV (63,0%)	28	1,4	32,0±8,6

Tabela 7.5: Resultados dos experimentos com o conjunto processamento de sêmen - Etapa 2

que divide os valores do atributo `class_a` — Tabela 7.1 — em quatro faixas, `baixa`, `media_baixa`, `media_alta` e `alta`, e o atributo `num_leu` em `alta` e `baixa`. A definição desse conhecimento adicional inserido no arquivo `.b` do Aleph encontra-se na Tabela B.1 no Apêndice B na página 105.

- No Experimento 2.2, além do conhecimento com-ad1 foi inserido o conhecimento adicional com-ad2, que manipula faixas de valores dos atributos `concentracao`, `hp` e `vitalidade` — Tabela B.2 Apêndice B.
- No Experimento 2.3, além do conhecimento com-ad1 e com-ad2, foi inserido o conhecimento adicional com-ad3, que manipula faixas de valores dos atributos `idade`, `ph`, `kruger` e `class_c` — Tabela B.3 Apêndice B.
- No Experimento 2.4 foi usado o mesmo conhecimento do domínio que no experimento 2.2, mas a classe ICSI como positiva.
- No Experimento 2.5 foi usado o mesmo conhecimento que no Experimento 2.1, e foi adicionado o conhecimento do especialista para construção de novos atributos, com-const, descrito na Seção 7.4.1 na página 88.
- Nos Experimentos 2.6 e 2.7 além do conhecimento com-ad1 e com-const, foi inserido

o conhecimento com-ad4 — Tabela B.4 Apêndice B — semelhante ao conhecimento com-ad2, mas utilizando outra faixa de valores.

- No Experimento 2.8 foi utilizado o conhecimento com-const e o conhecimento com-ad5 — Tabela B.5 Apêndice B. Esse conhecimento com-ad5 inclui alguns predicados definidos em experimentos anteriores, porém com faixas de valores diferentes para os atributos `e_classea`, `concentracao_t`, `motil`, `krug` e `aha_p`.

Com relação aos oito modelos induzidos, é interessante observar que todos eles utilizaram o conhecimento do domínio adicional, exceto no Experimento 2.4, que somente não utilizou a relação `no-leucocitos/3` do conhecimento adicional com-ad1 — Tabela B.1 Apêndice B.

A informação na Tabela 7.6 é semelhante a da Tabela 7.4, para os resultados obtidos nos experimentos realizados nesta segunda etapa.

Cobre	Experimento							
	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8
2	11	5	4	2	8	3	6	
3	6	7	6	5	9	5	1	4
4	3	3	3	2	1	4	3	2
5	4	5	5		5	5	1	7
6	5	1	2	2	5	1	1	2
7	2	3	2	2	2	3	2	3
8		2	3	2		1	2	
9		1	1			1		1
10				1			1	1
11				1			1	
12						1		
13								1
14							1	
15							1	1
16						1		
18				1				1
19								2
22				1				
25								1
30				1				
34								2
40							1	
#Regras	31	27	26	20	30	25	21	28
#Ex. Cobertos	116	120	122	138	116	134	158	269
Média	3,7	4,4	4,7	6,9	3,9	5,6	7,5	9,6
Desvio Padrão	1,7	2,1	2,1	7,4	1,7	3,3	8,4	8,9

Tabela 7.6: Número médio de exemplos cobertos pelas regras - Etapa2

Deve ser lembrado que nesta base de dados os valores dos atributos correspondem a um exame de laboratório de baixo custo (espermograma) enquanto que a classe é dada por um exame de alto custo (processamento de sêmen). Um dos objetivos é verificar em que

casos é possível determinar essa classe, para novos pacientes, considerando os resultados do exame de menos custo.

É evidente que neste tipo de problemas, freqüente na área médica, é difícil conseguir um modelo para classificar bem todos os novos casos não vistos, já que isso significaria que um exame pode ser substituído pelo outro. Se esse for o caso, o médico sempre recomendaria o exame menos custoso. Entretanto, é interessante para o especialista a procura de algumas regras desse modelo, tal que, para alguns casos, seja possível prever a classe sem realizar o exame mais custoso. Esse conhecimento pode ser extraído considerando as regras do modelo que cobrem um número considerável de exemplos.

O especialista do domínio esteve impedido na época de participar da análise dos resultados. Assim, a análise dos resultados foi feita utilizando o conhecimento obtido em outros trabalhos, nos quais contamos com o auxílio do especialista.

Considerando os modelos induzidos, o número de regras e a cobertura dessas regras, os experimentos 2.4, 2.7 e 2.8 apresentam alguns resultados interessantes. Por exemplo, tanto o modelo 2.4 quanto o 2.7 que utilizam ICSI como classe positiva (técnica utilizada quando há poucos espermatozoides por mililitro), apresentam no corpo de várias regras a relação `hora(tarde)`, que está relacionada com a hora de coleta de sêmen. Isso pode ser interpretado de várias maneiras, uma delas é que o ciclo fisiológico influencia no resultado da coleta. A seguir são apresentadas as regras dos modelos 2.4 e 2.7 que cobrem mais exemplos.

No Experimento 2.4 as regras que cobrem mais exemplos são

```
[Rule 1] [Pos cover = 22 Neg cover = 0] class(A,icsi) :-  
    hora(A,tarde), e_classea(A,media_alta,17), concentracao_esp(A,baixa,29.7).
```

```
[Rule 4] [Pos cover = 30 Neg cover = 0] class(A,icsi) :-  
    class_a(A,0), aha_p(A,baixa,58).
```

```
[Rule 7] [Pos cover = 18 Neg cover = 0] class(A,icsi) :-  
    processamento(A,20), concentracao_esp(A,baixa,29.7), aha_p(A,baixa,58).
```

No Experimento 2.7 as regras que cobrem mais exemplos são

```
[Rule 1] [Pos cover = 14 Neg cover = 0] class(A,icsi) :-  
    cor(A,translucido), e_classea(A,media_alta,17), motilidade_prog(A,baixa,69).
```

```
[Rule 2] [Pos cover = 15 Neg cover = 0] class(A,icsi) :-  
    processamento(A,20), class_a(A,0), concentracao_t(A,baixa,95.2).
```

```
[Rule 3] [Pos cover = 40 Neg cover = 0] class(A,icsi) :-
    e_classea(A,media_alta,17), aha_p(A,baixa,56), motilidade_prog(A,baixa,69).
```

Deve ser lembrado que na indução do modelo 2.4 não foi utilizado o conhecimento do especialista com-const, com a construção de três novos atributos. Outra diferença é que o conhecimento adicional com-ad4 utilizado para induzir o modelo 2.7 é semelhante ao com-ad2 utilizado pelo modelo 2.4, mas com outra faixa de valores.

Nas três regras que cobrem mais exemplos do modelo 2.4, pode ser observado que as relações `concentracao_esp(A,baixa,29.7)` e `aha_p(A,baixa,58)`, aparecem em duas dessas regras. A primeira relação é verdadeira quando a concentração de espermatozóides por mililitro é menor ou igual que 29.7 (em milhões de espermatozóides); a segunda relação é verdadeira quando o hp (teste hipo-osmótico) é menor ou igual a 58%.

No modelo 2.7 isso acontece com a relação `e_classea(A,media_alta,17)`, a qual é verdadeira quando a motilidade de grau A é menor ou igual que 17%; ainda neste modelo, a relação `motilidade_prog(A,baixa,69)` é verdadeira quando a motilidade progressiva, ou seja, a soma das motilidades de grau A e B, é menor ou igual que 69%. É interessante observar que a relação `motilidade_prog/3` foi definida utilizando um dos atributos construídos pelo especialista.

Pode também ser observado que as relações `e_classea(A,media_alta,17)`, `aha_p(A,baixa,56)`, `processamento(A,20)` e `class_a(A,0)` aparecem nas regras que cobrem mais exemplos em ambos modelos 2.4 e 2.7. `processamento(A,20)` indica que o tempo, em minutos, entre a coleta e o processamento da amostra foi de 20 minutos; `class_a(A,0)` indica que a motilidade de grau A é 0%;

É interessante notar que as regras que indicam, por exemplo, que pacientes sem espermatozóides com alta motilidade precisam ser tratados utilizando a técnica ICSI.

No Experimento 2.8, no qual a classe positiva consiste dos exemplos com classe ICSI e FIV, as regras que cobrem mais exemplos são

```
[Rule 1] [Pos cover = 19 Neg cover = 0] class(A,icsi+fiv) :-
    hora(A,tarde), e_classea(A,baixa,8), concentracao_t(A,baixa,99.4).
```

```
[Rule 2] [Pos cover = 34 Neg cover = 0] class(A,icsi+fiv) :-
    e_classea(A,baixa,8), concentracao_t(A,baixa,99.4), krug(A,baixa,4.5).
```

```
[Rule 3] [Pos cover = 19 Neg cover = 0] class(A,icsi+fiv) :-
    processamento(A,25), cor(A,branco), concentracao_t(A,media_alta,304.3).
```

```
[Rule 4] [Pos cover = 18 Neg cover = 0] class(A,icsi+fiv) :-
```

```
processamento(A,30), e_classea(A,baixa,8), concentracao_t(A,media_alta,304.3).
```

```
[Rule 5] [Pos cover = 34 Neg cover = 0] class(A,icsi+fiv) :-  
    e_classea(A,baixa,8), krug(A,baixa,4.5), volum(A,baixa,3.2).
```

```
[Rule 10] [Pos cover = 25 Neg cover = 0] class(A,icsi+fiv) :-  
    processamento(A,20), liquefacao(A,completa), concentracao_t(A,baixa,99.4).
```

Como pode ser observado, também neste caso, o período de coleta da amostra tem influência. Também, como a classe positiva inclui a classe intermediária FIV, a quantidade de espermatozoides com motilidade de grau A incrementou para 8%, continuando porém sendo baixa. A relação `krug/3`, que aparece nas duas regras que cobrem mais exemplos nesse modelo, é verdadeira quando a percentagem de espermatozoides com formato normal é menor ou igual a 4.5. Assim, essas regras indicam que pacientes com poucos espermatozoides de alta motilidade e poucos espermatozoides com formato normal precisam se submeter a técnicas mais sofisticadas (ICSI ou FIV). É interessante observar que a relação `concentracao_t/3` foi definida utilizando um dos atributos construídos pelo especialista.

7.5 Considerações Finais

Neste capítulo foi descrito um estudo de caso utilizando uma base de dados real. Devido a maneira como essa base de dados foi construída, *i.e.* os valores dos atributos provêm de um exame de laboratório de baixo custo, enquanto que as classes provêm de um outro exame de maior custo, a indução de um modelo que classifique com precisão qualquer novo caso é utópica. O objetivo nesse tipo de problema, muito freqüente na área médica, é encontrar um conjunto de regras que classifique bem alguns casos. Assim, novos casos cobertos por essas regras, indicam ao médico que não há necessidade de realizar o exame de alto custo pois é possível classificar esses casos utilizando os resultados do exame de baixo custo realizado por esses pacientes. Os resultados obtidos mostram que a utilização de conhecimento adicional do domínio, obtido a partir de modelos proposicionais, pode auxiliar no aprendizado relacional.

Apesar do conhecimento utilizado ter auxiliado nos resultados o conhecimento extraído ficou no nível proposicional. Um dos motivos é a possibilidade dessa base de dados não ter conhecimento relacional e por isso não foi possível encontrá-lo. Porém, acredita-se que o principal motivo para isso foi a falta de um conhecimento do domínio adequado. O melhor teria sido também utilizar um conhecimento do domínio formulado com o auxílio de um especialista. Com isso, o único conhecimento adicional do domínio utilizado foi

extraído do indutor proposicional.

Conclusões

*A*lgorítmos de aprendizado amplamente utilizados pela comunidade empregam como linguagem de descrição de objetos e conceitos a linguagem proposicional. Apesar do relativo sucesso desses sistemas proposicionais de aprendizado, eles são fortemente limitados devido a pouca expressividade do formalismo representacional e a capacidade limitada de utilizar conhecimento do domínio. Por outro lado, a descrição relacional é baseada em LPO, o que permite a descrição de objetos e relações entre os objetos. Sistemas de aprendizado que utilizam esse tipo de linguagem de representação são chamados de sistema de aprendizado relacional. Esses sistemas possuem uma alta expressividade para representar conceitos e a habilidade de representar conhecimento do domínio. Dentre os sistemas de aprendizado relacional, destacam-se os sistemas de Programação Lógica Indutiva. Vários desses sistemas têm sido propostos, entre eles o sistema Aleph, que permite, entre outros, simular diversos sistemas de PLI, o que motivou seu uso no desenvolvimento deste trabalho.

Para aproveitar a expressividade de sistemas relacionais quando os exemplos estão descritos no formato atributo-valor, é necessário transformar essa descrição proposicional para uma descrição relacional equivalente. Surge assim a necessidade de automatizar tanto quanto possível esse processo de transformação. Esse é o objetivo do módulo conversor *Kaeru* proposto neste trabalho. Em outras palavras, o módulo conversor *Kaeru* tem como objetivo auxiliar no processo de transformação de bases de dados proposicionais para o

formato relacional, automatizando e padronizando esse processo de transformação. Após realizada a transformação para o formato relacional, a base de dados pode ser utilizada por sistemas de PLI. Além de transformar bases de dados proposicionais para o formato relacional, o módulo conversor *Kaeru* cria os arquivos necessários para executar o sistema de PLI Aleph.

Assim, a principal vantagem do módulo *Kaeru* consiste da automatização desse processo o qual seria muito custoso de ser realizado manualmente, além da alta possibilidade de inserir erros nessa transformação manual. Após a execução do *Kaeru*, o usuário pode executar diretamente o Aleph e, em função dos resultados obtidos, somente necessita ajustar convenientemente os parâmetros para melhorar o modelo induzido, se for necessário.

O módulo conversor *Kaeru* faz parte do ambiente de PLI, em desenvolvimento, que será integrado ao projeto DISCOVER. O projeto DISCOVER é um ambiente computacional para descoberta de conhecimento em bases de dados, no qual estão integrados algoritmos de aprendizado proposicional oferecendo diversas funcionalidades voltadas para o aprendizado de máquina proposicional, mineração de dados e mineração de textos. O ambiente PLI permitirá a incorporação de aprendizado relacional no ambiente DISCOVER.

Dois conjuntos de experimentos são apresentados neste trabalho. O primeiro conjunto de experimento foi realizado utilizando-se bases de dados naturais da área médica, originalmente no formato atributo-valor. Esses conjuntos de dados foram transformados para o formato relacional equivalente utilizando o módulo conversor *Kaeru*. Após a transformação, os conjuntos de dados foram submetidos ao sistema de PLI Aleph para verificar a existência de conhecimento relacional. Além disso, utilizamos o modelo proposicional induzido pelo *See5* como conhecimento adicional do domínio. Utilizando esse conhecimento adicional foi possível melhorar os modelos induzidos por Aleph, mostrando que o conhecimento proposicional pode auxiliar na definição de conhecimento adicional do domínio para o aprendizado relacional. Isso é particularmente importante quando não é possível ter um especialista do domínio para auxiliar nessa tarefa, já que um bom conhecimento do domínio é fundamental para a descoberta de conhecimento utilizando sistemas de PLI.

Neste trabalho é também apresentado um estudo de caso utilizando um conjunto de dados do mundo real no formato atributo-valor, que foi transformado para o formato relacional do sistema Aleph utilizando o módulo conversor *Kaeru*. Na fase de escolha do conhecimento do domínio foram definidos alguns predicados sugeridos pelo especialista e, também, foi definido conhecimento adicional do domínio utilizando o conhecimento induzido pelo algoritmo de aprendizado proposicional *See5*. Os resultados obtidos mostram que a utilização de conhecimento adicional do domínio obtido a partir de modelos proposicionais, pode auxiliar no aprendizado relacional.

Como trabalhos futuros será realizada a implementação do segundo módulo do ambiente de PLI. Esse segundo módulo denominado Sniffer PLI permitirá medir o erro de hipóteses relacionais induzidas utilizando o Aleph, usando *k-fold cross validation* ou usando conjuntos de treinamento e teste definidos pelo usuário. Pretende-se também continuar a investigação de novas abordagens para auxiliar na geração de conhecimento adicional do domínio para ser utilizado por sistemas de PLI já que, conhecimento do domínio relevante melhora substancialmente os resultados de AM relacional.

Parâmetros do Aleph

`set(best,+V)` *V* é uma lista relacionada a uma cláusula obtida de uma execução anterior contendo, no mínimo, o número de exemplos positivos cobertos, o número de negativos cobertos e o tamanho da cláusula encontrada em uma busca anterior. Útil quando realizando buscas iterativamente;

`set(cache_clauselength,+V)` *V* é um inteiro positivo (padrão é 3). Ajusta o limite superior do tamanho de cláusulas que podem ser consideradas para uso futuro;

`set(caching,+V)` *V* é `true` ou `false` (padrão é *false*). Se `true`, então cláusulas e suas coberturas são armazenadas para uso futuro. Somente as cláusulas com tamanho menor ou igual ao que foi ajustado em `cache_clauselength` são armazenadas;

`set(check_redundant,+V)` *V* é `true` ou `false` (padrão é *false*). Especifica se uma chamada a `redundant/2` deve ser feita para verificar por literais redundantes em uma cláusula;

`set(check_useless,+V)` *V* é `true` ou `false` (padrão é *false*). Se `true`, remove literais na cláusula *bottom* que não contribuem para estabelecer ligações de variáveis;

`set(clauses,+V)` *V* é uma lista de classes a serem preditas quando utilizando aprendizado com árvores (*Tree Learning*);

`set(clauselength,+V)` *V* é um inteiro positivo (padrão é 4). Ajusta o limite superior do número de literais em uma cláusula aceitável;

`set(clauselength_distribution,+V)` *V* é uma lista da forma [*p*₁₋₁,*p*₂₋₂,...] em que *p_i* representa a probabilidade de selecionar uma cláusula com *i* literais. Utilizada pelos métodos aleatórios de busca;

`set(clauses,+V)` *V* é um inteiro positivo. Ajusta o limite superior do número de cláusulas em uma teoria quando realizando *theory_level search*;

`set(condition,+V)` *V* é `true` ou `false` (padrão é *false*). Se `true`, então os exemplos gerados randomicamente são obtidos após o condicionamento do gerador estocástico com os exemplos positivos;

`set(confidence,+V)` *V* é um número real no intervalo de 0.0 até 1.0 (padrão é 0.95). Determina a confiança para poda de regras no aprendizado com árvores;

`set(construct_bottom,+V)` *V* é `saturation`, `reduction` ou `false` (padrão é `saturation`). Especifica o estágio em que a cláusula *bottom* deve ser construída;

`set(depth,+V)` *V* é um inteiro positivo (padrão é 10). Ajusta o limite superior da profundidade de prova (*proof depth*);

`set(explore,+V)` *V* é `true` ou `false` (padrão é *false*). Se `true`, então força a busca a continuar até que todos os elementos restantes no espaço de busca são definitivamente piores que o melhor elemento atual (normalmente a busca pararia quando todos os elementos restantes não fossem melhores que o melhor elemento). Este é um critério fraco. Toda poda interna é ajustada para `off`;

`set(evalfn,+V)` *V* é `coverage`, `compression`, `posonly`, `pbayes`, `accuracy`, `laplace`, `auto_m`, `mestimate`, `entropy`, `gini`, `sd`, `wracc` ou `user` (padrão é `coverage`). Especifica qual a função de avaliação que será usada na busca;

`set(good,+V)` *V* é `true` ou `false` (padrão é *false*). Se `true`, então armazena código Prolog das "boas" cláusulas encontradas na busca. Uma cláusula boa é qualquer cláusula com utilidade acima da especificada em `minscore`. Se em `goodfile` foi definido algum nome de arquivo então esse código é armazenado nesse arquivo;

`set(goodfile,+V)` *V* é um átomo Prolog. Define o nome do arquivo para armazenar as boas cláusulas encontradas nas buscas;

`set(gsamplesize,+V)` V é um inteiro positivo (padrão é 100). Define o tamanho do conjunto de exemplos randomicamente gerado no aprendizado utilizando somente exemplos positivos;

`set(i,+V)` V é um inteiro positivo (padrão é 2). Ajusta o limite superior para a profundidade das novas variáveis;

`set(language,+V)` V é um inteiro ≥ 1 ou `inf` (padrão é `inf`). Especifica o número de ocorrências de um predicado em qualquer cláusula;

`set(lazy_on_contradiction,+V)` V é `true` ou `false` (padrão é `false`). Especifica se a prova de teorema pode prosseguir se uma restrição é violada;

`set(lazy_on_cost,+V)` V é `true` ou `false` (padrão é `false`). Especifica se a função de custo definida pelo usuário requer que a cobertura de cláusulas seja avaliada. Isto normalmente é definido internamente e não pelo usuário;

`set(lazy_negs,+V)` V é `true` ou `false` (padrão é `false`). Se `true`, então a prova de teorema com exemplos negativos pára quando o limite ajustado em `noise` ou `minacc` é violado;

`set(lookahead,+V)` V é um inteiro positivo. Ajusta um valor para o operador automático de refinamento;

`set(m,+V)` V é um número real. Ajusta um valor do *m-estimate*;

`set(minacc,+V)` V é um número real entre 0.0 e 1.0 (padrão é 0.0). Ajusta o limite inferior da precisão mínima de uma cláusula aceitável;

`set(mingain,+V)` V é um número real (padrão é 0.5). Especifica o ganho mínimo esperado para dividir uma folha quando construindo árvores;

`set(minpos,+V)` V é um inteiro positivo (padrão é 1). Ajusta o limite inferior do número de exemplos positivos que devem ser cobertos por uma cláusula. Se a melhor cláusula cobre um número menor de exemplos positivos, ela não é adicionada a teoria atual;

`set(minposfrac,+V)` V é um número real entre 0.0 e 1.0 (padrão é 0.0). Ajusta o limite inferior de exemplos positivos cobertos por uma cláusula aceitável como sendo a fração dos exemplos positivos cobertos pela cabeça dessa cláusula. Se a melhor cláusula tem taxa inferior a este número então ela não é adicionada a teoria atual;

`set(minscore,+V)` V é um número real ou `-inf` (padrão é `-inf`). Ajusta o limite inferior da utilidade de uma cláusula aceitável. Na construção de cláusulas, se a melhor cláusula tem utilidade abaixo deste número, então ela não é adicionada a teoria atual;

`set(moves,+V)` V é um inteiro ≥ 0 . Ajusta o limite superior do número de movimentos permitidos quando realizando uma busca randomica local.

`set(newvars,+V)` V é um inteiro positivo ou `inf` (padrão é `inf`). Ajusta o limite superior do número de variáveis que podem ser introduzidas no corpo de uma cláusula;

`set(nodes,+V)` V é um inteiro positivo (padrão é 5000). Ajusta o limite superior do número de nós a serem explorados quando realizando a busca por uma cláusula aceitável;

`set(noise,+V)` V é um inteiro ≥ 0 (padrão é 0). Ajusta o limite superior do número de exemplos negativos que podem ser cobertos por uma cláusula aceitável;

`set(openlist,+V)` V é um inteiro ≥ 0 ou `inf` (padrão é `inf`). Ajusta o limite superior da busca em largura para ser usada em uma busca gulosa (*greedy search*);

`set(optimise_clauses,+V)` V é `true` ou `false` (padrão é `false`). Se `true`, realiza *query optimisations* descrita por V.S. Costa, A. Srinivasan, e R.C. Camacho em *A note on two simple transformations for improving the efficiency of an ILP system*.

`set(portray_hypothesis,+V)` V é `true` ou `false` (padrão é `false`). Se `true`, executa a meta `portray(hypothesis)`, que deve ser definida pelo usuário;

`set(portray_literals,+V)` V é `true` ou `false` (padrão é `false`). Se `true`, executa a meta `portray(literals)` no qual `Literals` é algum literal. Essa meta deve ser definida pelo usuário;

`set(portray_search,+V)` V é `true` ou `false` (padrão é `false`). Se `true`, executa a meta `portray(search)` que deve ser definida pelo usuário;

`set(print,+V)` V é um inteiro positivo (padrão é 4). Define o limite superior do número máximo de literais mostrados no *trace*;

`set(proof_strategy,+V)` V é `restrict_sld` ou `sld` (padrão é `restrict_sld`). Se `restrict_sld`, então os exemplos cobertos foram determinados forçando a cláusula induzida a ser a primeira cláusula na prova de resolução SLD. Se `sld`, então essa restrição não é forçada;

`set(prooftime,+V)` V é um inteiro positivo ou `inf` (padrão é `inf`). Ajusta o limite superior de tempo (em segundos) para testar se um exemplo é coberto;

`set(prune_tree,+V)` V é `true` ou `false` (padrão é `false`). Determina se as regras construídas utilizando o aprendizado de árvore devem ser podadas utilizando poda pessimista;

`set(record,+V)` V é `true` ou `false` (padrão é `false`). Se `true`, então as linhas de execução do Aleph (*trace*) são escritas em um arquivo; O nome do arquivo é dado por `recordfile`.

`set(recordfile,+V)` V é um átomo Prolog. Define o nome do arquivo no qual serão gravadas as linhas de execução (*trace*) do Aleph;

`set(refine,+V)` V é `user`, `auto` ou `false` (padrão é `false`). Especifica qual o tipo de operador de refinamento. Se `user`, então o usuário deve especificar o operador de refinamento com `refine\2`;

`set(rls_type,+V)` V é `gsat`, `wsat`, `rrr` ou `anneal`. Especifica qual o método randomico de busca a ser utilizado;

`set(rulefile,+V)` V é um átomo Prolog. Define o nome do arquivo no qual serão armazenadas as cláusulas encontradas em uma teoria (usado por `write_rules/0`);

`set(samplesize,+V)` V é um inteiro ≥ 0 (padrão é 0). Define o número de exemplos selecionados randomicamente pelos comandos `induce` ou `induce_cover`;

`set(scs_percentile,+V)` V é um número na faixa de (0,100] (usualmente um número inteiro). Define que qualquer cláusula no topo do V -percentile de cláusulas são consideradas "boas" quando realizando seleção estocástica de cláusulas;

`set(scs_prob,+V)` V é um número na faixa de (0, 1.0]. Define a probabilidade mínima de se obter uma "boa" cláusula quando realizando seleção estocástica de cláusula;

`set(scs_sample,+V)` V é um número inteiro positivo que determina o número de cláusulas selecionadas randomicamente no espaço de hipóteses na busca clausal;

`set(search,+V)` V é `bf`, `df`, `heuristic`, `ibs`, `ils`, `rls`, `scs`, `id`, `ic` ou `ar` (padrão é `bf`). Especifica qual a estratégia de busca;

`set(searchtime,+V)` V é um inteiro ≥ 0 ou `inf`. Especifica o limite superior de tempo (em segundos) para uma busca;

`set(stage,+V)` V é `saturation`, `reduction` ou `command` (padrão é `command`). Define o estágio da execução. Isso normalmente é definido internamente e não pelo usuário;

`set(store_bottom,+V)` V é `true` ou `false` (padrão é `false`). Armazena a cláusula *bottom* construída para um exemplo para futura reutilização;

`set(temperature,+V)` V é um número real diferente de 0. Ajusta a `temperature` para busca quando utilizando *annealing*. Necessário que a `search` esteja definida como `rls` e `rls_type` para setar `anneal`;

`set(test_pos,+V)` V é um átomo Prolog ou uma lista de átomos. Define o arquivo ou a lista de arquivos que contém os exemplos positivos para testar;

`set(test_neg,+V)` V é um átomo Prolog ou uma lista de átomos. Define o arquivo ou a lista de arquivos que contém os exemplos negativos para testar;

`set(train_pos,+V)` V é um átomo Prolog ou uma lista de átomos. Define o arquivo ou a lista de arquivos que contém os exemplos positivos para treinamento;

`set(train_neg,+V)` V é um átomo Prolog ou uma lista de átomos. Define o arquivo ou a lista de arquivos que contém os exemplos negativos para treinamento;

`set(tree_type,+V)` V é `classification`, `class_probability` ou `regression`;

`set(tries,+V)` V é um inteiro positivo. Define o número máximo de reinícios permitido para os métodos randômicos de busca;

`set(typeoverlap,+V)` V é um número real no intervalo (0, 1.0]. Usado por `induce_modes/0` para determinar se a um par de diferentes tipos pode ser dado o mesmo nome;

`set(verbosity,+V)` V é um inteiro ≥ 0 (padrão é 1). Define o nível de detalhes de execução do Aleph;

`set(version,+V)` V é a versão atual do Aleph; é definido internamente;

`set(+P,+V)` Ajusta qualquer parâmetro P definido pelo usuário para o valor V. Este parâmetro é útil para anexar anotações aos experimentos. Por exemplo `set(experiment,'Execução 1 com conhecimento B0')`.

O Aleph possui um número muito grande de parâmetros que podem ser ajustados. A relação de todos esses parâmetros e os comandos disponíveis podem ser encontrados no manual do Aleph (Srinivasan, 2000), que pode ser obtido em:

www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/index.html

Conhecimento do Domínio do Conjunto de Dados Processamento de Sêmen

```
e_classea(Att_id,baixa,6):-  
    class_a(Att_id,CLASS_A), menor_igual(CLASS_A,6).  
e_classea(Att_id,media_baixa,6):-  
    class_a(Att_id,CLASS_A), maior(CLASS_A,6).  
e_classea(Att_id,media_alta,17):-  
    class_a(Att_id,CLASS_A), menor_igual(CLASS_A,17).  
e_classea(Att_id,alta,17):-  
    class_a(Att_id,CLASS_A), maior(CLASS_A,17).  
no_leucocitos(Att_id,baixa,5.2):-  
    num_leu(Att_id,NUM_LEU), menor_igual(NUM_LEU,5.2).  
no_leucocitos(Att_id,alta,5.2):-  
    num_leu(Att_id,NUM_LEU), maior(NUM_LEU,5.2).
```

Tabela B.1: Conhecimento adicional com-ad1

```

concentracao_esp(Att_id,baixa,29.7):-
    concentracao(Att_id,CONCENTRACAO),
    menor_igual(CONCENTRACAO,29.7).
concentracao_esp(Att_id,alta,29.7):-
    concentracao(Att_id,CONCENTRACAO),
    maior(CONCENTRACAO,29.7).
aha_p(Att_id,baixa,58):-
    hp(Att_id,HP), menor_igual(HP,58).
aha_p(Att_id,media_baixa,58):-
    hp(Att_id,HP), maior(HP,58).
aha_p(Att_id,media_alta,72):-
    hp(Att_id,HP), menor_igual(HP,72).
aha_p(Att_id,alta,72):-
    hp(Att_id,HP), maior(HP,72).
vital(Att_id,baixa,63):-
    vitalidade(Att_id,VITALIDADE),
    menor_igual(VITALIDADE,63).
vital(Att_id,alta,63):-
    vitalidade(Att_id,VITALIDADE),
    maior(VITALIDADE,63).

```

Tabela B.2: Conhecimento adicional com-ad2

```

idad(Att_id,maior_que,30):-
    idade(Att_id,IDADE),maior(IDADE,30).
idad(Att_id,baixa,37):-
    idade(Att_id,IDADE),menor_igual(IDADE,37).
idad(Att_id,maior_que,37):-
    idade(Att_id,IDADE), maior(IDADE,37).
idad(Att_id,menor_que,45):-
    idade(Att_id,IDADE), menor_igual(IDADE,45).
peaha(Att_id,baixa,7.5):-
    ph(Att_id,PH), menor_igual(PH,7.5).
peaha(Att_id,alta,7.5):-
    ph(Att_id,PH), maior(PH,7.5).
krug(Att_id,baixa,11):-
    kruger(Att_id,KRUGER),menor_igual(KRUGER,11).
krug(Att_id,alta,11):-
    kruger(Att_id,KRUGER), maior(KRUGER,11).
e_classec(Att_id,baixa,26):-
    class_c(Att_id,CLASS_C), menor_igual(CLASS_C,26).
e_classec(Att_id,alta,26):-
    class_c(Att_id,CLASS_C), maior(CLASS_C,26).

```

Tabela B.3: Conhecimento adicional com-ad3

```

motil(Att_id,baixa,80):-
    motilidade(Att_id,MOTILIDADE),
    menor_igual(MOTILIDADE,80).
motil(Att_id,alta,80):-
    motilidade(Att_id,MOTILIDADE),
    maior(MOTILIDADE,80).
concentracao_t(Att_id,baixa,95.2):-
    concentracao_total(Att_id,CONCENTRACAO_TOTAL),
    menor_igual(CONCENTRACAO_TOTAL,95.2).
concentracao_t(Att_id,alta,95.2):-
    concentracao_total(Att_id,CONCENTRACAO_TOTAL),
    maior(CONCENTRACAO_TOTAL,95.2).
motilidade_prog(Att_id,baixa,69):-
    mot_prog(Att_id,MOT_PROG),
    menor_igual(MOT_PROG,69).
motilidade_prog(Att_id,alta,69):-
    mot_prog(Att_id,MOT_PROG),
    maior(MOT_PROG,69).
krug(Att_id,baixa,5):-
    kruger(Att_id,KRUGER),
    menor_igual(KRUGER,5).
krug(Att_id,alta,5):-
    kruger(Att_id,KRUGER),
    maior(KRUGER,5).
e_classe(Att_id,baixa,27):-
    class_c(Att_id,CLASS_C),
    menor_igual(CLASS_C,27).
e_classe(Att_id,alta,27):-
    class_c(Att_id,CLASS_C),
    maior(CLASS_C,27).

```

Tabela B.4: Conhecimento adicional com-ad4

```

e_classea(Att_id,baixa,8):-
    class_a(Att_id,CLASS_A), menor_igual(CLASS_A,8).
e_classea(Att_id,alta,8):-
    class_a(Att_id,CLASS_A), maior(CLASS_A,8).
concentracao_t(Att_id,baixa,99.4):-
    concentracao_total(Att_id,CONCENTRACAO_TOTAL),
    menor_igual(CONCENTRACAO_TOTAL,99.4).
concentracao_t(Att_id,media_baixa,99.4):-
    concentracao_total(Att_id,CONCENTRACAO_TOTAL),
    maior(CONCENTRACAO_TOTAL,99.4).
concentracao_t(Att_id,media_alta,304.3):-
    concentracao_total(Att_id,CONCENTRACAO_TOTAL),
    menor_igual(CONCENTRACAO_TOTAL,304.3).
concentracao_t(Att_id,alta,304.3):-
    concentracao_total(Att_id,CONCENTRACAO_TOTAL),
    maior(CONCENTRACAO_TOTAL,304.3).
motil(Att_id,baixa,81):-
    motilidade(Att_id,MOTILIDADE),
    menor_igual(MOTILIDADE,81).
motil(Att_id,alta,81):-
    motilidade(Att_id,MOTILIDADE),
    maior(MOTILIDADE,81).
krug(Att_id,baixa,4.5):-
    kruger(Att_id,KRUGER), menor_igual(KRUGER,4.5).
krug(Att_id,alta,4.5):-
    kruger(Att_id,KRUGER), maior(KRUGER,4.5).
aha_p(Att_id,baixa,75):-
    hp(Att_id,HP), menor_igual(HP,75).
aha_p(Att_id,alta,75):-
    hp(Att_id,HP), maior(HP,75).
volum(Att_id,baixa,3.2):-
    volume(Att_id,VOLUME),menor_igual(VOLUME,3.2).
volum(Att_id,alta,3.2):-
    volume(Att_id,VOLUME),maior(VOLUME,3.2).

```

Tabela B.5: Conhecimento adicional com-ad5

Referências Bibliográficas

- Baranauskas, J. A. & G. E. A. P. A. Batista (2000). O Projeto DISCOVER: Idéias Iniciais. Comunicação pessoal. 52
- Baranauskas, J. A. & M. C. Monard (2003). Combining Symbolic Classifiers from Multiple Inducers. *Knowledge Based Systems* 16(3), 129–136. Elsevier Science. 52
- Batista, G. E. A. P. A. & M. C. Monard (2003a). An Analysis of Four Missing Data Treatment Methods for Supervised Learning. *Applied Artificial Intelligence* 17(5), 519–533. <http://www.icmc.usp.br/~gbatista>. 53
- Batista, G. E. A. P. A. & M. C. Monard (2003b). Descrição da Arquitetura e do Projeto do Ambiente Computacional DISCOVER LEARNING ENVIRONMENT — DLE. *Technical Report* 187, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_187.pdf. 52, 53, 54, 57
- Batista, G. E. A. P. A., R. C. Prati, & M. C. Monard (2004). A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. *SIGKDD Explorations* 6(1). Special issue on Learning from Imbalanced Datasets (in print). 53
- Batista, G. E. A. P. A. (2001). Sintaxe padrão do arquivo de exemplos do projeto DISCOVER. <http://www.icmc.sc.usp.br/~gbatista/Discover/SintaxePadraoFinal.htm>. 52, 55
- Batista, G. E. A. P. A. (2003). Pré-processamento de Dados em Aprendizado de Máquina Supervisionado. Tese de Doutorado, ICMC-USP, <http://www.icmc.usp.br/~gbatista>. 52, 53

- Bernardini, F. C. (2003). Combinação de classificadores simbólicos utilizando medidas de regras de conhecimento. Qualificação de Doutorado, ICMC-USP. 53
- Blockeel, H. (1997, July). Tilde - top down induction of logical decision trees. *Technical Report*, K.U.Leuven Dept. of Computer Science. 40, 41
- Blum, A. & T. Mitchell (1998). Combining labeled and unlabeled data with co-training. Em *Proc. 11th Annu. Conf. on Comput. Learning Theory*, pp. 92–100. ACM Press, New York, NY. 8
- Boswell, T. (1990a). *Manual for CN2 version 4.1* (TI/P2154/RAB/4/1.3 ed.). The Turing Institute. 12
- Boswell, T. (1990b, January). Manual for NewId version 4.1. *Technical Report* TI/P2154/RAB/4/2.3, The Turing Institute. 54
- Buntine, W. (1988). Generalised Subsumption and Its Applications to Induction and Redundancy. *Artificial Intelligence* 36(2), 149–176. 32
- Camacho, R. (1994). Learning stage transition rules with Indlog. Volume 237 of *GMD-Studien*, pp. 273–290. Gesellschaft für Mathematik und Datenverarbeitung MBH. 41
- Caulkins, C. W. (2000). Aquisição de conhecimento utilizando aprendizado de máquina relacional. Dissertação de Mestrado, ICMC-USP. 52
- Chiara, R. & M. C. Monard (2003). Projeto e implementação de um filtro para transformar logs de servidores web em arquivos no formato padrão do sistema DISCOVER. *Technical Report* 183, ICMC-USP. 53
- Clark, P. & R. Boswell (1991). Rule Induction with CN2: Some Recent Improvements. Em Y. Kodratoff (ed.), *Fifth European Conference (EWSL 91)*, pp. 151–163. Springer-Verlag. 54
- Clark, P. & T. Niblett (1989). The *CN2* induction algorithm. *Machine Learning* 3(4), 261–283. 12
- Cohen, W. (1993). Learnability of restricted logic programs. pp. 41–72. 28
- Dehaspe, L. & L. De Raedt (1997). Mining association rules in multiple relations. pp. 125–132. Springer-Verlag. 41
- De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach*. London, UK: Academic Press. 39

- Dosualdo, D. G. (2003). Investigação de regressão no processo de mineração de dados. Dissertação de Mestrado, ICMC-USP. 52
- Džeroski, S. & I. Bratko (1992). Handling Noise in Inductive Logic Programming. Em *Proceedings of the Second International Workshop on Inductive Logic Programming*, Tokyo. ICOT. 33
- Džeroski, S., L. Todorovski, & T. Urbančič (1995). Handling Real Numbers in ILP: a Step Towards Successful Behavioral Cloning. Em *Proceedings of the Eighth European Conference on Machine Learning*, Berlin, pp. 283–286. 39, 40
- Ferro, M., H. D. Lee, & S. C. Esteves (2002, June). Intelligent Data Analysis: A Case Study of the Diagnostic Sperm Processing. Em *Proceedings ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (CSITeA-02)*, Foz do Iguaçu, Brazil, pp. 116–120. 85
- Ferro, M. & M. C. Monard (2004). Descrição da Implementação do Módulo Conversor *Kaeru* para Transformar Dados no Formato Atributo-Valor para o Formato Relacional do Sistema de Programação Lógica Indutiva Aleph. *Technical Report* 234, ICMC-USP. 57, 60
- Ferro, M. (2002). Aplicação de métodos de aprendizado de máquina simbólico para extração de conhecimento de bases de dados reais: Um estudo de caso do processamento de sêmen. Monografia para Obtenção do Grau de Bacharel, Unioeste. 85
- Flach, P. A. (2000). The use of functional and logic languages in machine learning. Em M. Alpuente (ed.), *Ninth International Workshop on Functional and Logic Programming (WFLP2000)*, pp. 225–237. Universidad Politecnica de Valencia. 18
- Gamma, E., R. Helm, R. Johnson, & J. Vlissides (1995). *Design Patterns: Elements of Resusable Object-Oriented Software*. Addison Wesley. 53
- Geromini, M. R. (2002). Projeto e Desenvolvimento da Interface Gráfica do Sistema DISCOVER. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP. 52
- Karalič, A. & I. Bratko (1997). First-order regression. *Machine learning*. 39, 41
- Lavrač, N. & S. Džeroski (1994). *Inductive Logic Programming: Techniques and Applications*. New York: Ellis Horwood. 10, 11, 15, 26, 27, 35, 39, 40
- Lavrač, N. & S. Džeroski (2001). *Relational Data Mining*. Berlin: Springer-Verlag. 15, 18, 23, 24, 33, 35

- Lavrač, N. & P. A. Flach (2001). An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic (TOCL)* 2(4), 458–494. 20, 21, 40
- Lavrač, N. (1998). Computacional logic and machine learning: A roadmap for inductive logic programming. *Technical Report*, J. Stefan Institute, Ljubljana, Slovenia. 21, 23, 24
- Lee, H. D. (2000). Seleção e construção de features relevantes para o aprendizado de máquina. Dissertação de Mestrado, ICMC-USP. 85
- Lee, H. D. (2003). Seleção e transformação de atributos para a extração de conhecimento de bases de dados. Qualificação de Doutorado, ICMC-USP. 52
- Martins, C. A. (2003). Uma Abordagem para Pré-processamento de Dados Textuais em Algoritmos de Aprendizado. Tese de Doutorado, ICMC-USP. 52
- Matsubara, E. T., C. A. Martins, & M. C. Monard (2003). Pretext: Uma ferramenta para pré-processamento de textos utilizando a abordagem *bag-of-words*. *Technical Report* 209, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_209.zip. 52
- Matsubara, E. T. (2004). O algoritmo de aprendizado semi-supervisionado CO-TRAINING e sua aplicação na rotulação de documentos. Dissertação de Mestrado, ICMC-USP. <http://www.icmc.usp.br/~edson/artigos/dissertacao.zip>. 8
- Melanda, E. A. (2002). Pós-processamento de conhecimento de regras de associação. Qualificação de Doutorado, ICMC-USP. 52
- Melanda, E. A. (2004). *Pós-processamento de Regras de Associação*. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, USP, São Carlos, SP, Brasil (a ser defendida). 53
- Michalski, R. S., J. G. Carbonell, & T. M. Mitchell (1983). *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company. 10
- Milaré, C. R. (2003). Extração de conhecimento de redes neurais artificiais utilizando sistemas de aprendizado simbólico e algoritmos genéticos. Tese de Doutorado, ICMC-USP. 52
- Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence* 18, 203–226. Reprinted in Shavlik and Dietterich (eds.), 1990. Readings in Machine Learning, Morgan Kaufmann Publishers, Inc. 29

- Mitchell, T. M. (1998). *Machine Learning*. McGraw-Hill. 8
- Monard, M. C. & J. A. Baranauskas (2003a). *Conceitos sobre Aprendizado de Máquina* (1 ed.), Capítulo 4, pp. 89–114. Volume 1 of 1 Rezende (2003). 12
- Monard, M. C. & J. A. Baranauskas (2003b). *Indução de Regras e Árvores de Decisão* (1 ed.), Capítulo 5, pp. 115–140. Volume 1 of 1 Rezende (2003). 13
- Muggleton, S. H. & W. Buntine (1988). Machine Invention of First-Order Predicates by Inverting Resolution. Em *Proceedings of the Fifth International Conference on Machine Learning*, pp. 339–352. Morgan Kaufmann. 32
- Muggleton, S. H. & C. Feng (1990). Efficient Induction of Logic Programs. Em *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, pp. 368–381. Ohmsha. 31, 39
- Muggleton, S. H. (1991). Inductive Logic Programming. *New Generation Computing* 8, 295–318. 20, 22, 37
- Muggleton, S. & L. D. Raedt (1994). Inductive Logic Programming. *Journal of Logic Programming* 19–20, 629–679. 21, 25, 28, 29, 40
- Muggleton, S. (1995). Inverse Entailment and Progol. *New Generation Computing* 13, 245–286. 38, 39, 41
- Muggleton, S. (1999). Inductive logic programming: Issues, results and the challenge of learning language in logic. *Artificial Intelligence* 114(1-2), 283–296. 15
- Nédellec, C., C. Rouveirol, H. Adé, F. Bergadano, & B. Tausend (1996). Declarative bias in ILP. Em L. D. Raedt (ed.), *Advances in Inductive Logic Programming*, Volume 32 of *Frontiers in Artificial Intelligence and Applications*, pp. 82–103. IOS Press. 28
- Nienhuys-Cheng, S.-H. & R. Wolf (1997). *Foundations of Inductive Logic Programming*, Volume 1228 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag. 28, 40
- Pazzani, M. and Brunk, C. & G. Silverstein (1991). A Knowledge-Intensive Approach to Learning Relational Concepts. Em *Proceedings of the Eighth International Workshop on Machine Learning*, San Francisco, pp. 432–436. Morgan Kaufmann. 38
- Pazzani, M. & D. Kibler (1991). The Utility of Knowledge in Inductive Learning. *Machine Learning* 8, 295–318. 33

- Pila, A. D. & M. C. Monard (2002). Rules Induced by Symbolic Learning Algorithms Using Rough Sets Reducts for Selecting Features: An Empirical Comparison with Other Filters. Em A. Zapico & J. M. Santos (eds.), *Proceedings Argentine Symposium on Artificial Intelligence, ASAI'2002*, Santa Fe, Argentina, pp. 206–217. 52
- Pila, A. D. (2001). Seleção de de atributos relevantes para aprendizado de máquina utilizando a abordagem de rough sets. Dissertação de Mestrado, ICMC-USP. 52
- Plotkin, G. D. (1970). A Note on Inductive Generalization. Em B. Meltzer & D. Michie (eds.), *Machine Intelligence*, Volume 5, pp. 153–163. Edinburgh University Press. 30
- Plotkin, G. D. (1971). A Further Note on Inductive Generalization. Em B. Meltzer & D. Michie (eds.), *Machine Intelligence*, Volume 6, New York. Elsevier. 31
- Prati, R. C., J. A. Baranauskas, & M. C. Monard (2001a). Extração de Informações Padronizadas para a Avaliação de Regras Induzidas por Algoritmos de Aprendizado de Máquina Simbólico. *Technical Report* 145, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip. 14
- Prati, R. C., J. A. Baranauskas, & M. C. Monard (2001b). Uma proposta de unificação da linguagem de representação de conceitos de algoritmos de aprendizado de máquina simbólicos. *Technical Report* 137, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip. 52, 53, 55
- Prati, R. C., G. E. A. P. A. Batista, & M. C. Monard (2004). Class Imbalances *versus* Class Overlapping: an Analysis of a Learning System Behavior. Em *Mexican International Conference on Artificial Intelligence, LNAI 2972*, pp. 312–321. Springer-Verlag. 53
- Prati, R. C. (2003). O *Framework* de Integração do Sistema DISCOVER. Dissertação de Mestrado, ICMC-USP. 52
- Pugliesi, J. B. (2001). O pós-processamento em extração de conhecimento de bases de dados. Exame de Qualificação de Doutorado, ICMC-USP. 52
- Pugliesi, J. B. (2004). *Pós-processamento de Regras de Regressão*. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, USP, São Carlos, SP, Brasil. 53
- Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning* 1, 81–106. Reprinted in Shavlik and Dietterich (eds.), 1990. Readings in Machine Learning, Morgan Kaufmann Publishers, Inc. 12, 54

- Quinlan, J. R. (1987). Generating Production Rules from Decision Trees. Em *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Italy, pp. 304–307. 54
- Quinlan, J. R. (1988). *C4.5 Programs for Machine Learning*. CA: Morgan Kaufmann. 12, 54, 55, 60
- Quinlan, J. R. (1990). Learning Logical Definitions from Relations. *Machine Learning* 5(3), 239–266. 33, 39, 41
- Rezende, S. O., J. B. Pugliesi, & F. M. Varejão (2003). *Sistemas Baseados em Conhecimento* (1 ed.), Capítulo 2, pp. 13–49. Volume 1 of 1 Rezende (2003). ISBN 85-204-1683-7. 15
- Rezende, S. O. (2003). *Sistemas Inteligentes: Fundamentos e Aplicações*. Barueri, SP, Brasil: Editora Manole. 113, 115
- Robinson, J. (1965). A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM* 12(1), 23–41. 32
- Rulequest-Research (1999). Data Mining Tools See5 and C5.0. <http://www.rulequest.com/see5-info.html>em03/1999. 70
- Russel, S. & P. Norvig (2003). *Artificial Intelligence: A Modern Approach* (2 ed.). Prentice Hall. 14, 20, 38
- Sammut, C. A. & R. B. Banerji (1986). Learning Concepts by Asking Questions. Em R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Volume 2, Los Altos, California, pp. 167–191. Morgan Kaufmann. 39
- Sanches, M. K. (2003). Aprendizado de máquina semi-supervisionado: proposta de um algoritmo para rotular exemplos a partir de poucos exemplos rotulados. Dissertação de Mestrado, ICMC-USP, <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-12102003-140536>. 8
- Shapiro, E. Y. (1983). *Algorithmic Program Debugging*. The MIT Press. 33, 39, 40
- Srinivasan, A. (2000). The aleph manual. *Technical Report*, Oxford University. <http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph>. 41, 62, 104
- Wall, L., T. Christiansen, & R. L. Schwartz (1996). *Programming Perl* (2 ed.). O'Reilly & Associates. 52

Weiss, S. M. & C. A. Kulikowski (1991). *Computer Systems that Learn*. San Mateo, CA: Morgan Kaufmann. 7

Wrobel, S. (1997). An algorithm for multi-relational discovery of subgroups. In J. Komorowski & J. Zytkow (eds.), *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD-97)*, Berlin, pp. 78–87. Springer-Verlag. 41