UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Aprendizado Relacional por um Modelo Neural

por

JULIANA DELGADO SANTOS HERNANDEZ

Dissertação submetida à avaliação, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Paulo Martins Engel
Orientador

Porto Alegre, maio de 2001.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Hernandez, Juliana Delgado Santos

Aprendizado Relacional por um Modelo Neural / por Juliana Delgado Santos Hernandez. – Porto Alegre: PPGC da UFRGS, 2001.

74 p.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001. Orientador: Engel, Paulo Martins.

1. Aprendizado relacional. 2. Mineração de dados. 3. Redes Neurais 4. Programação lógica indutiva. I. Engel, Paulo Martins. II. Título.

Agradecimentos

A realização de um Mestrado é algo indescritível. Passamos por desafios, vitórias, conquistas, decepções... Independente de tudo, sempre conseguimos forças para seguirmos adiante.

Durante este trabalho tive apoio de várias pessoas e gostaria de agradecer a todas. Expresso aqui a gratidão àquelas que estiveram mais de perto.

Inicialmente, gostaria de agradecer à Prof. Carla, pois sem ela eu não estaria realizando o Mestrado. Desde que cheguei de Belo Horizonte, recebi uma ajuda inestimável e vou agradecê-la pelo resto de minha vida!

Agradeço também a todos os meus colegas das disciplinas que cursei, pelo aprendizado em conjunto e trabalhos realizados.

Não posso deixar de lembrar do pessoal da Biblioteca, sempre dispostos a ajudar! Um agradecimento especial a Ida, que me ajudou bastante com muita paciência e prontidão.

Agradeço bastante ao meu orientador, Prof. Paulo Engel, e ao Hércules Prado pelas valiosas discussões e ajuda durante a realização deste trabalho.

Por último, gostaria de agradecer ao apoio, amor e carinho que sempre tive da minha família e família do meu marido, que mesmo de longe, sempre estiveram junto de mim. Também agradeço ao amor, compreensão, amizade e atenção do meu marido Pedro. Sem ele, com certeza eu não teria forças para concluir este Mestrado.

Sumário

Lista de Abreviaturas

BD	Banco de Dados
BDR	Banco de Dados Relacional
DCBD	Descoberta de Conhecimento em Banco de Dados
FOLONET	*First Order Neural Network*
INSS	*Incremental Neuro-Symbolic System*
KBANN	*Knowledge Based Neural Networks*
MD	Mineração de Dados
MGGR	Menor generalização geral relativa
MNC	Modelo Neural Combinatório
PL	Programação em Lógica
PLI	Programação Lógica Indutiva
RNs	Redes Neurais

Lista de Símbolos

$\oplus$	Exemplo positivo.
$\ominus$	Exemplo negativo.
θ	Letra grega teta.
$\neg p$	Negação de p.
$q \leftarrow p$	p implica q.
$\in$	Pertence a.
$q \wedge p$	q e p
$\cup$	União
*	Multiplicação
exp	Exponencial
$\geq$	Maior ou igual

Lista de Figuras

Lista de Tabelas

Resumo

As técnicas que formam o campo da Descoberta de Conhecimento em Bases de Dados (DCBD) surgiram devido à necessidade de se tratar grandes volumes de dados. O processo completo de DCBD envolve um elevado grau de subjetividade e de trabalho não totalmente automatizado. Podemos dizer que a fase mais automatizada é a de Mineração de Dados (MD). Uma importante técnica para extração de conhecimentos a partir de dados é a Programação Lógica Indutiva (PLI), que se aplica a tarefas de classificação, induzindo conhecimento na forma da lógica de primeira ordem. A PLI tem demonstrado as vantagens de seu aparato de aprendizado em relação a outras abordagens, como por exemplo, aquelas baseadas em aprendizado proposicional. Os seus algoritmos de aprendizado apresentam alta expressividade, porém sofrem com a grande complexidade de seus processos, principalmente o teste de cobertura das variáveis.

Por outro lado, as Redes Neurais Artificiais (RNs) introduzem um ótimo desempenho devido à sua natureza paralela. O maior problema em relação às RNs é que geralmente são "caixas pretas", o que torna difícil a obtenção de uma interpretação razoável da estrutura geral da rede na forma de construções lógicas de fácil compreensão. Várias abordagens híbridas simbólico-conexionistas (por exemplo, o MNC [MAC 890], KBANN [SHA 94], [TOW 94] e o sistema INSS [OSO 98]) têm sido apresentadas para lidar com este problema, permitindo o aprendizado de conhecimento simbólico através de uma RN. Entretanto, estas abordagens ainda lidam com representações atributo-valor.

Neste trabalho é apresentado um modelo que combina a expressividade obtida pela PLI com o desempenho de uma rede neural: a *FOLONET* (*First Order Neural Network*).

PALAVRAS-CHAVE: Aprendizado Relacional, Mineração de Dados, Redes Neurais, Programação Lógica Indutiva.

TITLE: "*RELATIONAL LEARNING BY A NEURAL MODEL*"

Abstract

The techniques that form the field of Knowledge Discovery in Databases (KDD) fulfill the need of treating great volumes of data. The full process of KDD involves a high degree of subjectivity and is not totally automatized. We can say that the phase in this process that is more automatized is Data Mining (DM). An important technique for extraction of knowledge from data is Inductive Logic Programming (ILP), that can be applied to classification tasks, inducing knowledge in the form of first-order logic. The ILP community has for long demonstrated the advantages of its learning apparatus over other approaches, like the attribute-value based ones. ILP learners present a well recognized expressivity, while suffering from a high order complexity, mainly in the variable's coverage algorithm.

On the other hand, artificial neural networks (ANNs) present a good performance due to their parallel nature. The main drawback related to ANNs remains in the fact that most of the time they are black boxes, and then it becomes difficult to achieve a reasonable interpretation of the network's general structure in a form of logical constructions that are easy to understand by humans. Some hybrid connectionist-symbolic approaches (e.g., CNM system [MAC 89], KBANN [SHA 94], [TOW 94] and the INSS system [OSO 98]) have been presented to cope with this problem, allowing to learn symbolic knowledge by means of an ANN. However, these approaches still deal with attribute-value representations.

In this work we present a model that combines the expressivity obtained by ILP learners with the performance of ANNs: *FOLONET* (*First Order Neural Network*).

KEYWORDS: Relational Learning, Data Mining, Neural Networks, Inductive Logic Programming.

1 Introdução

As técnicas que compõem o processo de Descoberta de Conhecimento em Bancos de Dados (DCBD) surgiram como resposta à necessidade de se tratar volumes crescentes de dados. O processo de DCBD é tipicamente interativo: alguém que tenha um problema e uma base de dados relevante para a resolução do mesmo, interage com um sistema de DCBD na busca por padrões que possam ser usados para a solução do problema, não se excluindo mudanças de rumos previamente traçados em função de resultados parciais alcançados durante a mineração [PRA 98a].

É importante ressaltar que freqüentemente DCBD e Mineração de Dados (MD) são confundidas. Entre os pesquisadores predomina o conceito de DCBD como um processo mais geral que vai desde a aquisição dos dados, tratamento dos mesmos, extração de padrões (a mineração, propriamente dita) até a sua interpretação e incorporação em bases de conhecimento. Por outro lado, no meio dos usuários, a MD tem sido usada para se referir ao processo completo.

Neste trabalho, utiliza-se como base as definições apresentadas por [FAY 96], considerada como uma das fontes mais autorizadas:

Mineração de Dados: é um passo do processo de DCBD, consistindo de algoritmos que, sob alguma limitação de eficiência computacional aceitável, produz uma particular enumeração de padrões[1] sobre um conjunto de dados.

Descoberta de Conhecimento em Banco de Dados: é o processo que utiliza algoritmos de mineração de dados para extrair *o que é* conhecimento de acordo com especificações de medidas e limites, usando um banco de dados F juntamente com algum pré-processamento, sub-amostragem, e transformações requeridas sobre F.

1.1 O Processo de Descoberta de Conhecimento

O processo de DCBD pode ser apresentado como uma seqüência de etapas cuja execução se dá de forma iterativa e interativa. A sua característica iterativa se deve ao fato de que se pode avançar nas etapas e posteriormente voltar a uma que já tenha sido executada e a interativa pelo fato de se basear intensamente nas entradas e respostas providas pelo usuário [PRA 98a].

Conforme classificação proposta por G. John [JOH 97] ajustada às definições para DCBD e MD adotadas, o processo completo de DCBD consta das seguintes etapas:

- Definição e Entendimento do Problema;
- Obtenção dos Dados;
- Limpeza e Exploração dos Dados;

[1] Segundo Fayyad et. al. [FAY 96], padrão é uma expressão E em uma linguagem L que descreve fatos contidos em um subconjunto F_E de um conjunto de fatos F. E é chamado de padrão se é mais simples do que a enumeração de todos os fatos contidos em F_E.

- Engenharia dos Dados;
- Engenharia do Algoritmo;
- Mineração;
- Interpretação e Validação dos Resultados.

O processo de DCBD é ilustrado na figura 1.1.

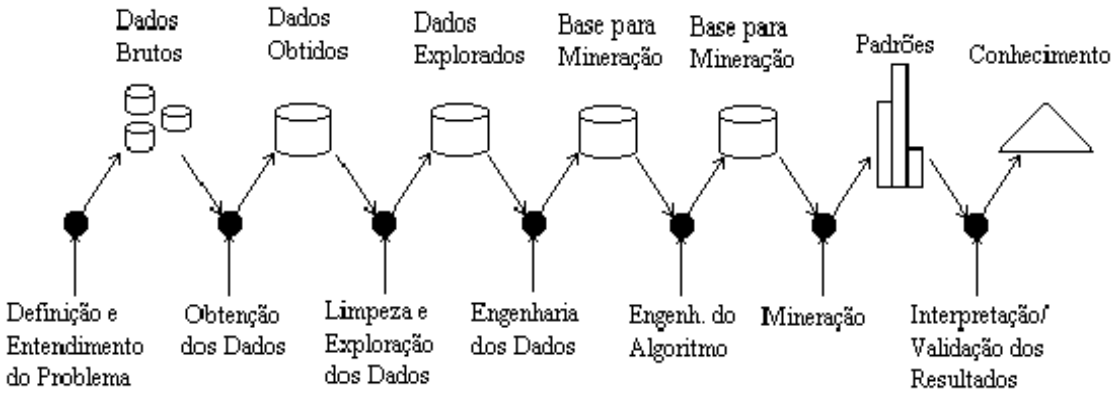


FIGURA 1.1 - O processo de DCBD

O processo completo de DCBD envolve um elevado nível de subjetividade e de trabalho não totalmente automatizado. Podemos dizer que a fase mais automatizada é a de mineração.

1.2 O Processo de Mineração de Dados

Os dois objetivos principais em um processo de mineração na prática são: predição e descrição [FAY 96]. Estes dois objetivos têm um mapeamento direto nos dois tipos básicos de algoritmos de aprendizado de máquina: supervisionado ou não supervisionado, respectivamente.

A predição visa estabelecer o valor de um ou mais atributos em uma base de dados, a partir de outros atributos presentes. A abordagem preditiva não implica necessariamente a previsão de um valor futuro – "a característica importante é que ela faz uma estimativa sobre o valor de um ou mais atributos desconhecidos, dados os valores de outros atributos conhecidos" [JOH 97]. Por outro lado, a descrição visa apontar padrões potencialmente interessantes nos dados sem uma associação com um conceito *a priori*.

Há diversos métodos de MD e a seguir tem-se uma breve explanação de um subconjunto das técnicas mais populares.

- Árvores de Decisão

Uma árvore de decisão é um modelo de classificação que utiliza uma seqüência de pequenas decisões elementares, baseadas em perguntas que admitem respostas do tipo *sim* ou *não* sobre os atributos dos objetos sendo classificados, para a identificação da classe a que pertence cada objeto. Uma estrutura de árvore de decisão possui:

- Folhas, indicando classes;
- Nodos de decisão, representando os atributos dos indivíduos; e
- Arcos ligando, hierarquicamente, nodos de decisão entre si ou a uma folha.

Os ramos de saída de um nodo correspondem a todas as possibilidades de saída do teste no nodo. Os exemplos em um nodo na árvore são, então, particionados ao longo dos ramos e cada ramo obtém seu correspondente subconjunto de exemplos.

Um algoritmo bastante conhecido para geração de árvores de decisão é o ID3 de Quinlan [QUI 86] com versões de extensão chamadas C4 [QUI 90a]. ID3 inicia com todos os exemplos do treinamento no nodo raiz da árvore e, então, um atributo é selecionado para particionar estes exemplos. Para cada valor do atributo, um ramo é criado e o subconjunto correspondente de exemplos que têm o valor do atributo especificado pelo ramo é movido para o nodo filho recentemente criado. O algoritmo é aplicado recursivamente para cada nodo filho até que todos os exemplos em um nodo sejam de apenas uma classe, ou todos os exemplos em um nodo tenham o mesmo valor para todos os atributos. Cada nodo folha na árvore de decisão representa uma classe.

- Redes Neurais

Redes Neurais (RNs) são sistemas paralelos distribuídos compostos por unidades de processamento simples (nodos ou neurônios) que computam determinadas funções matemáticas (normalmente não-lineares). Tais unidades são dispostas em uma ou mais camadas e interligadas por um grande número de conexões, geralmente unidirecionais. Na maioria dos modelos estas conexões estão associadas a pesos, os quais armazenam o conhecimento representado no modelo, e servem para ponderar a entrada recebida por cada neurônio da rede. O funcionamento destas redes é inspirado em uma estrutura física concebida pela natureza: o cérebro humano.

A solução de problemas através das RNs é bastante atrativa, já que a forma como estes são representados internamente pela rede e o paralelismo natural inerente à arquitetura das RNs criam a possibilidade de um desempenho superior ao dos modelos convencionais [BRA 98]. Diversas aplicações de RNs em DCBD têm sido apresentadas devido a esta característica marcante.

- Programação Lógica Indutiva

Programação Lógica Indutiva (PLI) é uma tecnologia que combina princípios de aprendizado de máquina indutivo com princípios da programação em lógica e tem como objetivo induzir regras gerais a partir de observações específicas e conhecimento anterior [LAV 94].

PLI é uma área de pesquisa que está na interseção de aprendizado de máquina e programação em lógica. O resultado desta junção é um aparato formal para induzir descrições relacionais na forma de programas em lógica a partir de observações e, possivelmente, de conhecimento prévio. Esta técnica tem mostrado seu potencial de aplicação nas seguintes áreas: aquisição de conhecimento, síntese de programa indutivo, engenharia de dados indutiva, e descoberta de conhecimento em bancos de dados.

O principal objetivo da PLI é desenvolver teorias, técnicas e aplicações de aprendizado indutivo a partir de observações e conhecimento anterior. Inserida no Aprendizado Indutivo de Conceitos, PLI extrai regras gerais ou uma teoria subjacente aos exemplos, representando uma abordagem ao principal problema dos sistemas especialistas: o da aquisição do conhecimento.

1.3 Objetivo e Motivação

A DCBD está preocupada com a identificação de padrões interessantes e com suas descrições num modo conciso e significativo. É reconhecido que o aprendizado de máquina pode ser aplicado a problemas de DCBD [FRA 91]. Sistemas de aprendizado atributo-valor, como o C4.5 [QUI 93] e CN2 [CLA 89] têm sido aplicados a problemas de DCBD, entretanto, estas abordagens possuem várias limitações. Os padrões que são descobertos são expressos em linguagens de atributo-valor que têm o poder de expressão da lógica proposicional. Estas linguagens são limitadas e não permitem a representação de objetos complexos nem relações entre eles ou seus componentes.

PLI pode ser vista como aprendizado de máquina em uma linguagem de primeira ordem. Esta técnica é importante para a descoberta de conhecimento em banco de dados relacionais, visto que pode descrever padrões envolvendo mais de uma relação.

A Mineração de Dados tem sido apontada como uma das principais áreas de aplicação da PLI [DZE 96], na qual esta técnica já mostrou seu potencial através de diversos sistemas de aprendizado, como FOIL [QUI 90b], GOLEM [MUG 90] e LINUS [LAV 94]. Aplicações como diagnóstico preventivo de doenças reumáticas, classificação biológica da qualidade da água em rios, predição da estrutura local de uma proteína a partir de sua seqüência de aminoácidos são exemplos de investidas bem sucedidas destes sistemas.

Mesmo tendo sido bem aplicada a tarefas de MD, deve ser ressaltado que apesar das bases de dados usadas nas aplicações de PLI serem reais, resultados de estudos científicos, elas não são muito grandes para os padrões de hoje. Isso aponta para a necessidade de se aprimorar estas técnicas visando o processamento de grandes bases de dados. Neste ponto reside a principal motivação do trabalho: estudo de melhoramento das técnicas de MD com o objetivo de se utilizar grandes quantidades de dados.

Por outro lado, abordagens de aprendizado de máquina baseadas em Redes Neurais apresentam um bom desempenho em relação a grandes quantidades de dados, porém têm sua expressividade limitada à da lógica proposicional [MAC 89], [TOW 94], [OSO 98].

O principal objetivo deste trabalho é o desenvolvimento de uma plataforma para experimentação, a fim de se obter uma proposta para aprendizado indutivo que futuramente possa ser utilizada em grandes bancos de dados e que tenha expressividade da lógica de primeira ordem.

O presente texto apresenta todo o estudo feito durante a realização deste trabalho, bem como a proposta desenvolvida e os principais resultados. Ele está dividido em 6 capítulos organizados conforme abaixo:

- Capítulo 1: consiste desta Introdução;

- Capítulo 2: descreve brevemente a Programação Lógica Indutiva, técnica utilizada como referência para a obtenção da expressividade da lógica de primeira ordem;

- Capítulo 3: discute algumas propostas de redes neurais para o aprendizado simbólico, dando ênfase ao MNC [MAC 89] que foi a base para o desenvolvimento da proposta obtida a partir deste trabalho;

- Capítulo 4: apresenta a *FOLONET* (*First Order Logic Neural Network*), que é uma proposta de rede neural desenvolvida durante o trabalho para aprendizado relacional em bancos de dados;

- Capítulo 5: mostra os principais resultados obtidos da implementação inicial da *FOLONET* e propostas para sua extensão;

- Capítulo 6: apresenta as principais conclusões do trabalho bem como sugestões de trabalhos futuros para melhoria da *FOLONET*.

2 Programação Lógica Indutiva

Programação Lógica Indutiva (PLI) tem sido definida como a interseção entre Aprendizado de Máquina e Programação Lógica [MUG 91]. Em função disto, PLI emprega tanto técnicas do Aprendizado de Máquina quanto da Programação Lógica.

Do Aprendizado de Máquina, PLI herda seus objetivos: desenvolver ferramentas e técnicas para induzir hipóteses de observações (exemplos) e obter novo conhecimento da experiência. Utilizando lógica computacional como o mecanismo de representação para hipóteses e observações, PLI pode superar as duas maiores limitações das técnicas clássicas de aprendizado de máquina [MUG 94]:

- uso de um limitado formalismo de representação de conhecimento (essencialmente lógica proposicional). Esta limitação é importante porque muitos domínios de aplicações só podem ser expressos em lógica de primeira ordem.

- dificuldades em usar conhecimento prévio substancial no processo de aprendizagem.

Da Lógica Computacional, PLI herda seu formalismo de representação, sua orientação semântica e várias técnicas já bem estabelecidas. Ao contrário de outras abordagens do aprendizado indutivo, PLI está interessada nas propriedades das regras de inferência, na convergência dos algoritmos e na complexidade dos procedimentos.

Este capítulo mostra as principais características da Programação Lógica Indutiva assim como alguns exemplos de suas aplicações. A seguir são apresentados alguns conceitos básicos para o melhor entendimento da PLI.

2.1 Conceitos Básicos

A Programação Lógica Indutiva está centrada no aprendizado indutivo que expressa uma forma de raciocínio que parte do específico para o geral. No caso de aprendizado indutivo a partir de exemplos, regras gerais ou uma teoria subordinada aos exemplos são derivadas.

O aprendizado indutivo tem sido aplicado com sucesso a uma variedade de problemas de predição e classificação, tais como diagnóstico de doenças ou predição de propriedades mecânicas do aço tendo como base suas características químicas. Estes problemas podem ser formulados como tarefas de aprendizado de conceitos a partir de exemplos, referenciadas como *aprendizado indutivo de conceitos*, no qual regras de classificação para um conceito específico devem ser induzidas a partir de instâncias daquele conceito. Para se entender o aprendizado indutivo de conceitos, é necessária a definição de *conceito* [LAV 94]:

- **Conceito:** Seja *U* um conjunto de objetos ou observações. Então um conceito *C* é um subconjunto de *U*. Por exemplo, *U* pode ser o conjunto de clientes de um banco e *C* representa os clientes preferenciais.

Dizemos que a descrição do conceito cobre a descrição do objeto, ou que a descrição do objeto é coberta pela descrição do conceito, se a descrição do objeto satisfaz a descrição do conceito. Estabelecemos esta noção de cobertura através da idéia de *exemplo*:

- **Exemplo:** Durante o aprendizado de um conceito *C*, um exemplo *e* é uma descrição rotulada de um objeto. Quando *e* é uma instância de *C*, tem-se um exemplo positivo, sendo rotulado com $\oplus$. Caso contrário, tem-se um exemplo negativo, sendo rotulado com $\ominus$. Para facilitar a notação, utilizaremos e^+ para exemplos positivos e e^- para exemplos negativos.

Dadas estas definições iniciais, podemos estabelecer o problema do aprendizado indutivo de conceitos da seguinte forma:

Aprendizado Indutivo de Conceitos: Dado um conjunto *E* de exemplos positivos e negativos de um conceito *C*, encontrar uma hipótese *H*, tal que:

- Todo exemplo positivo $e^+ \in E$ é coberto por *H*;

- Nenhum exemplo negativo $e^- \in E$ é coberto por *H*.

Na formulação do problema do Aprendizado Indutivo de Conceitos estão envolvidas as idéias de completude e consistência: a hipótese *H* é completa (cobre todos os exemplos positivos) e consistente (não cobre exemplos negativos) em relação aos exemplos *E*.

Muitos sistemas de PLI utilizam esta definição para o problema a ser tratado e, se não há algum conhecimento anterior sobre o problema, eles utilizam exclusivamente os exemplos no processo de aprendizagem. Entretanto, são muitas as tarefas que requerem uma grande quantidade de conhecimento prévio.

A formulação do problema de aprendizado indutivo de conceitos é restabelecido para a inclusão do conhecimento prévio

Aprendizado Indutivo de Conceitos com Conhecimento Prévio: Dado um conjunto *E* de exemplos de treinamento e o conhecimento prévio *B*, encontrar uma hipótese *H* tal que ela seja completa e consistente em relação a *B* e aos exemplos *E*.

A noção de completude e consistência também são atualizadas:

- *H* é completa se todos os exemplos positivos são cobertos por $B \cup H$.

- *H* é consistente se nenhum exemplo negativo é coberto por $B \cup H$.

Terminologia de Banco de Dados Dedutivo e Programação Lógica

Esta seção introduz a terminologia básica de programação em lógica e BD Dedutivo, bastante utilizada na PLI.

Uma *relação n*-ária *p* é um conjunto de tuplas, ou seja, um subconjunto do produto cartesiano de *n* domínios $D_1 \times D_2 \times ... \times D_n$, onde um *domínio* (ou um tipo) é um conjunto de valores. Assume-se que uma relação é finita, a não ser que seja dito o contrário. Um conjunto de relações forma um *banco de dados relacional* (BDR).

Uma cláusula de programa lógico é uma cláusula da seguinte forma:

$$T \leftarrow L_1, ..., L_m$$

onde *T* é um átomo e $L_1, ..., L_m$ são da forma *L* ou não *L*, onde *L* é um átomo.

Um conjunto de cláusulas de programa com o mesmo *símbolo predicado p* na cabeça forma uma *definição de predicado*. Um predicado pode ser definido extensionalmente como um conjunto de fatos ou intensionalmente como um conjunto de cláusulas de banco de dados [ULL 97]. Assume-se que cada predicado é definido extensionalmente ou intensionalmente. Quando o corpo da cláusula é vazio, temos, então, a definição de um fato.

Uma relação em um banco de dados equivale a uma definição de predicado em um programa lógico. Da mesma forma, uma tupla em um banco de dados equivale a um fato em um programa lógico. A tabela 2.1 relaciona os termos de banco de dados e da programação lógica.

TABELA 2.1 - Terminologia de BD e PL

Terminologia de BD	Terminologia de PL
Nome de relação *p*	Símbolo de predicado *p*
Atributo da relação *p*	Argumento do predicado *p*
Tupla $\langle a_1, ..., a_n \rangle$	Fato $p(a_1, ..., a_n)$
Relação *p* – conjunto de tuplas	Predicado *p* – definido extensionalmente por um conjunto de fatos
Relação *q* – definida como uma visão do BD	Predicado *q* – definido intensionalmente por um conjunto de regras (cláusulas)

2.2 Sistemas PLI

Sistemas PLI, assim como outros sistemas de aprendizado indutivo, podem ser divididos ao longo de várias dimensões, como se segue:

- *Top-Down* X *Bottom-Up*: uma distinção útil entre sistemas PLI diz respeito à direção em que a busca ocorre.

  - *Top-Down*: começa com uma hipótese H tal que o conjunto $H \cup B$ é muito genérico e, então, o especializa. Um sistema *Top-Down* pode algumas vezes adaptar-se localmente aos exemplos através de um passo de generalização. Tal generalização pode ser necessária para corrigir um passo anterior e maior de especialização, o que torna a hipótese final bastante fraca. Após a correção, o sistema continua sua busca *Top-Down*.

  - *Bottom-Up*: começa com uma hipótese H, tal que $H \cup B$ é muito específico e, então, o generaliza. Analogamente ao sistema *Top-Down*, um sistema *Bottom-Up* pode algumas vezes realizar um passo de especialização. Entretanto, um sistema pode geralmente ser classificado de uma maneira natural como *Top-Down* ou *Bottom-Up*, dependendo da direção geral de sua busca.

- Aprendizado de único ou vários predicados: podemos distinguir os sistemas de acordo com o número de predicados envolvidos no processo de aprendizagem.

  - Aprendizado de um único predicado: todos os exemplos dados são instâncias de somente um predicado P, e o objetivo da tarefa de aprendizado é encontrar um conjunto de cláusulas que implica $P(x_1, ..., x_n)$ somente para aquelas tuplas $\langle x_1, ..., x_n \rangle$ cujo significado "pertence" ao conceito denotado por P. Em outras palavras, o conjunto de cláusulas deve reconhecer as instâncias de P. Embora todos os exemplos tenham o mesmo predicado P, outros símbolos predicados (predefinidos no conhecimento prévio) podem ser utilizados para formar uma hipótese correta.

  - Aprendizado de múltiplos predicados: os exemplos são instâncias de mais de um predicado. É importante ressaltar que um aprendizado de múltiplos predicados não pode ser separado em vários problemas de um único predicado porque os diferentes predicados em uma tarefa de aprendizado de múltiplos predicados podem estar relacionados.

- Aprendizado *Batch* ou Incremental: a distinção entre aprendizado *batch* e incremental se dá pela maneira que os exemplos são dados ao sistema.

  - Aprendizado *Batch*: todos os exemplos são dados logo no início da execução do sistema. Isto faz com que se tenha a vantagem de verificar erros nos exemplos dados (ruído), que podem ser medidos e tratados pela aplicação de técnicas estatísticas ao conjunto de todos os exemplos.

  - Aprendizado Incremental: os exemplos são dados um por um. Neste caso, o sistema ajusta a cada momento sua hipótese aos exemplos já fornecidos, antes de obter o próximo exemplo.

- Sistemas Interativos e Não-Interativos: a distinção está no fato de haver ou não interação com o usuário.

  - Sistemas Interativos: podem interagir com o usuário a fim de obter alguma informação adicional. Por exemplo, o sistema pode perguntar ao usuário se alguma afirmação é verdadeira ou falsa. Desta forma, um sistema interativo pode inserir alguns exemplos durante a busca.

  - Sistemas Não-Interativos: nestes sistemas não há a possibilidade de interação com um usuário.

Embora estas dimensões sejam independentes em princípio, há sistemas de PLI que estão situados em dois extremos do espectro. Por um lado, há os sistemas *batch* não-interativos que aprendem um único predicado, enquanto por outro lado, há sistemas interativos e incrementais que aprendem múltiplos predicados. Os primeiros sistemas são chamados de sistemas PLI Empíricos enquanto que os segundos são chamados de sistemas PLI Interativos ou Incrementais.

Abaixo segue uma breve descrição de alguns sistemas PLI Empíricos, que já mostraram seu potencial para aplicações práticas.

2.2.1 FOIL

FOIL [QUI 90b] é uma melhora do sistema de aprendizado de árvore de decisão ID3 [QUI 86]. Seu processo de aprendizado utiliza uma abordagem de cobertura que supõe que as cláusulas da hipótese são aprendidas uma a uma. Cada nova cláusula *C* construída pelo sistema deve ser de tal forma que, junto com a hipótese atual (que corresponde a todas as cláusulas já obtidas anteriormente) e conhecimento prévio, implica alguns exemplos positivos que não são implicados sem ela. Além disso, a cláusula *C* e o conhecimento prévio não podem implicar exemplos negativos. Após a construção de uma cláusula, o sistema a adiciona à hipótese atual e remove do conjunto de exemplos aqueles positivos que foram cobertos por ela. Este processo continua até que não exista mais exemplos positivos no conjunto de exemplos.

O sistema FOIL se enquadra na categoria de sistemas *Top-Down*. Cada cláusula da hipótese é construída começando da cabeça $P(x_1, ..., x_n)$ (onde P é o predicado que se deseja aprender), que é então especializada por um operador de refinamento que adiciona novos literais ao corpo da cláusula. A busca através de um grafo de refinamento (ou seja, a seleção de literais que serão adicionados) é guiada por uma heurística de ganho de informação.

2.2.2 GOLEM

GOLEM [MUG 90] é um sistema *Bottom-Up* que emprega o conceito de cláusulas de *menor generalização geral relativa* (mggr). As variáveis no corpo da mggr devem ser determinadas, ou seja, seus valores devem ser, direta ou indiretamente, determinados unicamente pelos valores das variáveis na cabeça da mggr.

Mesmo sob a restrição de determinação, as mggrs são geralmente cláusulas longas que contêm literais irrelevantes no seu corpo. GOLEM utiliza, então, exemplos negativos e declarações de modo que especificam os argumentos de entrada e saída de um predicado para reduzir o tamanho das cláusulas. Se a eliminação de um literal do corpo não causa a cobertura de exemplos negativos pela cláusula, então o literal irrelevante é descartado e o processo de remoção de literais continua com a cláusula remanescente.

Tanto os exemplos de treinamento quanto o conhecimento prévio no sistema GOLEM são restritos a fatos concretos. Ao contrário do sistema FOIL, termos contendo símbolos funcionais são permitidos nos exemplos e também no conhecimento prévio.

2.2.3 LINUS

LINUS [LAV 94] é um sistema que integra vários sistemas de aprendizado atributo-valor. Ele pode ser visto como um conjunto de ferramentas de PLI e de algoritmos de aprendizado no qual um ou mais algoritmos são selecionados a fim de se encontrar a melhor solução para um dado problema.

A principal idéia do sistema LINUS é transformar problemas de aprendizado relacional (problemas de PLI) em uma tarefa de aprendizado proposicional (aprendizado atributo-valor). O algoritmo que resolve problemas PLI transformando-os em uma forma proposicional consiste das seguintes etapas:

- A tarefa de aprendizado é transformada da forma relacional para a de atributo-valor;

- O problema de aprendizado transformado é, então, resolvido por um sistema de aprendizado atributo-valor, por exemplo, CN2 [CLA 89] que utiliza um tipo de operador de refinamento para aprender regras *Se-Então* apropriadas.

- A solução induzida pelo sistema de aprendizado atributo-valor é traduzida de volta para a forma relacional.

2.2.4 CLAUDIEN

CLAUDIEN [RAE 97] é um sistema PLI que se adapta bem ao paradigma de Descoberta de Conhecimento em Bancos de Dados e Mineração de Dados. Isto se deve ao fato de descobrir padrões que são válidos nos dados. Assim , CLAUDIEN realiza uma tarefa de indução nova, que é chamada de indução característica a partir de observações fechadas, e que está relacionada a formalizações de indução em lógica.

Na indução característica a partir de observações fechadas os padrões são representados pela teoria de cláusulas e os dados usam interpretações de *Herbrand*. Este sistema também emprega um mecanismo de *bias* declarativo para definir o conjunto de cláusulas que podem aparecer na hipótese.

CLAUDIEN foi o primeiro sistema que trabalhou no ambiente não monotônico, derivando eficientemente hipóteses de cláusulas de banco de dados. Ele é baseado em uma busca de aprofundamento iterativa que parte do geral para o específico, utilizando um operador de refinamento. Ao mesmo tempo, oferece uma abordagem natural para o aprendizado empírico de múltiplos predicados, que necessita interação com o usuário. Realmente, no ambiente não monotônico é fácil o aprendizado de múltiplos predicados, pois se duas cláusulas c_1 e c_2 são válidas, então sua conjunção também é válida. Isto é contrário ao que ocorre no ambiente normal, onde a conjunção de duas cláusulas que contribuem individualmente para a solução pode violar o requerimento de conhecimento prévio e hipótese consistentes em relação aos exemplos negativos.

2.3 Áreas de Aplicação

Os algoritmos de PLI obtêm como saída regras que são facilmente entendidas pelas pessoas. Isto faz com que sejam particularmente apropriados para tarefas de formação de teoria científica, na qual a compreensão do conhecimento induzido é essencial para o avanço na pesquisa científica.

Outra característica importante da PLI é a utilização de um formalismo de lógica relacional que permitiu o sucesso da sua aplicação em vários domínios nos quais os conceitos a serem aprendidos não podiam ser facilmente descritos em linguagens atributo-valor. Estas aplicações incluem, entre outras: projeto de malha de elementos finitos [DOL 92], construção automática de modelos qualitativos e previsão de estrutura secundária de proteína [LAV 94].

Além destas áreas de aplicação, Nada Lavrac [LAV 94] apresenta algumas aplicações que incluem:

- Aprendizado de regras para diagnóstico precoce de doenças reumáticas: o correto diagnóstico em um estado precoce de uma doença reumática é um problema difícil. Isto se deve ao fato de que sintomas, manifestações clínicas e laudos de laboratórios de várias doenças reumáticas são similares e não são específicos. O diagnóstico ainda pode ser incorreto devido à interpretação subjetiva de dados clínicos e de laboratórios. A avaliação médica das regras induzidas mostra que a utilização de conhecimento prévio melhora substancialmente a qualidade das regras induzidas de um ponto de vista médico.

- Modelagem do problema de se relacionar a estrutura de um componente químico e as propriedades químicas de seus constituintes a uma propriedade específica de um componente;

- Aprendizado de regras de diagnóstico a partir de modelos qualitativos: a idéia básica é utilizar um modelo qualitativo de um sistema para gerar comportamentos daquele sistema. Se há uma falha no sistema, ela irá refletir no comportamento gerado pelo mesmo. Este "comportamento" de falha pode ser usado para fornecer exemplos a um programa de aprendizado que pode, então, aprender regras de diagnóstico relacionando o comportamento observado do sistema a causas internas.

Como se pode observar, a PLI tem sido utilizada em diversas áreas do conhecimento e tem mostrado suas potencialidades. Contudo, ainda há restrições com relação a sua eficiência e esta será discutida na próxima seção.

2.4 Discussão

Quando se compara técnicas de atributo-valor e PLI, há um dilema entre o poder de expressão e eficiência. Técnicas de PLI são tipicamente muito mais expressivas, mas também são pouco eficientes. Entretanto, a PLI é capaz de solucionar problemas que estão fora do escopo do aprendizado proposicional.

Os requisitos computacionais dos sistemas de PLI são maiores do que os dos aprendizes proposicionais devido às seguintes razões [BLO 2000]: primeiro, o espaço de cláusulas considerado pelos sistemas de PLI tipicamente é bem maior do que os dos sistemas proposicionais e também pode ser infinito. Segundo, o teste de cobertura de uma cláusula é muito mais complexo que no aprendizado atributo-valor. Nestes sistemas, um exemplo corresponde a uma única tupla em um banco de dados relacional. Já nos sistemas de PLI, um exemplo pode corresponder a várias tuplas de múltiplas relações. Por esta razão, o teste de cobertura em PLI necessita de um sistema de banco de dados para resolver consultas complexas. Terceiro, o teste de cobertura de um exemplo em aprendizado atributo-valor geralmente é feito localmente, ou seja, independentemente dos outros exemplos. Portanto, mesmo que o conjunto de dados seja enorme, um teste de cobertura específico pode ser realizado eficientemente. Isto contrasta com a grande maioria dos sistemas de PLI, tal como FOIL [QUI 90b], no qual o teste de cobertura é feito globalmente, ou seja, para testar a cobertura de um exemplo, todo o conjunto de exemplos e o conhecimento prévio devem ser considerados. Testes de cobertura globais são muito mais caros do que os testes locais.

Em uma abordagem mais recente para a PLI, chamada de aprendizado por interpretações [RAE 98b], [BLO 2000] e [LAE 2000], assume-se que cada exemplo é um pequeno banco de dados (ou uma parte do banco de dados global), e testes de cobertura locais são realizados. Algoritmos que utilizam testes locais tipicamente têm complexidade linear no número de exemplos. Além disso, como cada exemplo pode ser carregado independentemente dos outros exemplos, não há necessidade de se utilizar um sistema de banco de dados mesmo quando o conjunto de dados não pode ser totalmente carregado na memória principal.

Mesmo com as tentativas de melhora no desempenho dos sistemas PLI, os conjuntos de dados considerados são pequenos de acordo com os padrões gerais dentro da comunidade de Mineração de Dados. Tendo em vista este fato, procuramos sistemas de redes neurais a fim de encontrar um desempenho satisfatório para as tarefas de MD.

No próximo capítulo apresentamos as principais características das RNs que são utilizadas para aprendizado simbólico, bem como alguns sistemas avaliados. A ênfase é dada ao Modelo Neural Combinatório, que foi a base para o desenvolvimento de uma nova proposta de Rede Neural para aprendizado relacional.

3 Propostas de Redes Neurais para Aprendizado Simbólico

O foco central das aplicações de Mineração de Dados é entender o conhecimento contido em grandes bancos de dados. Freqüentemente, para atingir este objetivo, devem ser aplicados métodos de aprendizado de máquina a fim de se construir modelos de dados por indução.

As redes neurais fornecem algoritmos poderosos para aprendizado de máquina e conotam um caráter implícito de conhecimento que é adquirido de um conjunto de padrões de treinamento e distribuído ao longo das conexões da estrutura dentro de seu aprendizado. Uma característica importante é o paralelismo que contribui para aprofundar as capacidades de aprendizado, já que os elementos de cálculo individuais na rede são capazes de ajustar suas conexões para realizar o melhor mapeamento possível para o conjunto de treinamento dos casos.

Enquanto o paralelismo melhora o aprendizado, ele torna difícil a obtenção de uma interpretação razoável da estrutura geral da rede na forma de construções lógicas explícitas de fácil compreensão, tais como regras conjuntivas do tipo Se-Então.

Para superar estas dificuldades, foram propostos algoritmos para extração de conhecimento de RNs, além de arquiteturas mais adequadas para representação de conhecimento de RNs.

Uma abordagem para compreender uma hipótese representada por uma RN treinada é traduzir sua hipótese numa linguagem mais compreensível. Os métodos de extração de conhecimento diferem em vários aspectos :

- Linguagem de representação: regras de inferência conjuntiva (Se-Então), regras *m*-de-*n*, regras *fuzzy*, árvores de decisão, autômatos finitos.

- Estratégia de extração, que define como o método explora o espaço de descrições candidatas e em que nível ele descreve o comportamento da rede:

  - Métodos globais: extraem regras que descrevem o comportamento da rede como um todo. As classes de saída são caracterizadas diretamente em termos das entradas.

  - Métodos locais: extraem o comportamento das unidades individuais (ocultas) da rede. As regras das unidades individuais são combinadas num conjunto de regras que descreve o comportamento da rede com um todo.

- Exigências da rede: exigências sobre a arquitetura e treinamento que o método de extração impõe à rede. Limita os modelos de rede para os quais o método é aplicável.

Uma rede com classes de saída discretas e características de entrada com valores discretos pode ser descrita exatamente por um conjunto finito de regras simbólicas *se-então*. As regras simbólicas extraídas especificam condições que devem ser satisfeitas pelas características de entrada para produzir um estado de saída específico.

Este capítulo apresenta três abordagens para Aprendizado Simbólico através de Sistemas Híbridos Simbólico-Conexionistas: a rede KBANN [TOW 94], o sistema INSS [OSO 98] e o Modelo Neural Combinatório [MAC 89]. Sistemas híbridos têm sido apontados como a próxima geração de Sistemas de Mineração de Dados [ANA 98]. O principal objetivo é combinar duas ou mais técnicas de forma que se possa superar as limitações das técnicas quando utilizadas individualmente. No caso dos sistemas que serão apresentados, a idéia é inserir regras simbólicas obtidas anteriormente em uma rede neural e refiná-las utilizando algoritmos padrões de aprendizagem de RNs e exemplos de treinamento classificados. O passo final é extrair regras compreensíveis e refinadas da rede neural treinada.

3.1 Rede KBANN

KBANN (*Knowledge-Based Artificial Neural Networks*) [TOW 94] é um sistema de aprendizado híbrido baseado em Redes Neurais Artificiais para tarefas de classificação, combinando conhecimento simbólico prévio do domínio e conhecimento extraído de exemplos.

Inicialmente são inseridas informações simbólicas da teoria do domínio do problema em uma RN. A seguir, este conhecimento inicial é refinado através de algoritmos de aprendizado de RNs, utilizando os exemplos de treinamento. Finalmente, uma teoria refinada do domínio é extraída da rede treinada [SHA 94]. A figura 3.1 ilustra os algoritmos que formam o sistema KBANN.

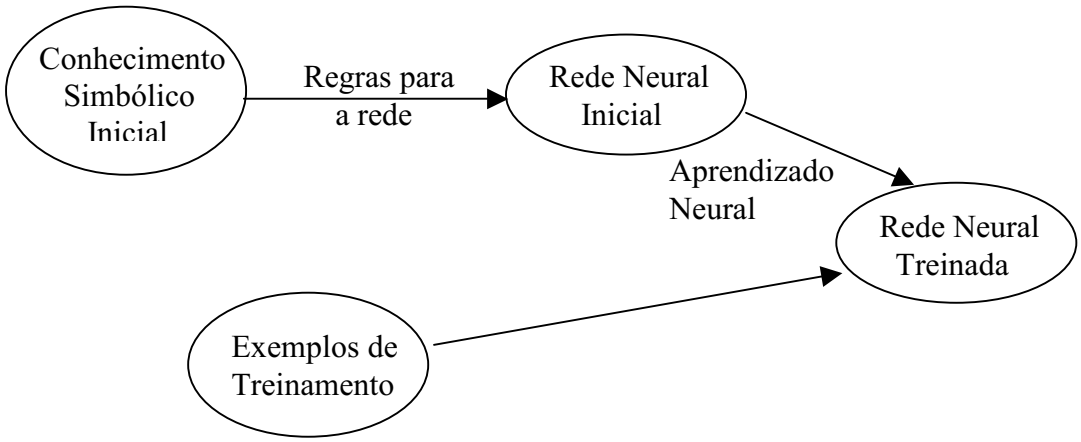


FIGURA 3.1 - Fluxo do refinamento de teoria pelo sistema KBANN

O sistema KBANN utiliza redes neurais multicamadas conectadas para frente (*feedforward*) que são treinadas pelo algoritmo de *Backpropagation*. Os neurônios possuem uma função logística de ativação e seguem o modelo funcional do perceptron:

$$u_i = \sum_{j \in \{\text{unidades conectadas}\}} w_{ji} \cdot a_j$$

$$a_i = \frac{1}{1 + \exp(-(u_i - \theta_i))}$$

onde u_i é a soma ponderada das entradas a_j do neurônio i, w_{ji} é o peso sináptico na conexão entre os neurônios i e j, a_i é a ativação do neurônio i e θ_i é o limiar do neurônio i.

3.1.1 Inserção de conhecimento na KBANN

O primeiro passo da rede KBANN é a tradução de um conjunto de regras aproximadamente corretas em uma rede baseada em conhecimento. As regras a serem traduzidas são expressas como cláusulas de *Horn* (cláusulas com no máximo um literal positivo [NIE 97]).

A tradução de bases de conhecimento para redes neurais segue a seguinte correspondência:

TABELA 3.1 - Correspondência entre bases de conhecimento e redes neurais

Base de Conhecimento	Redes Neurais
Conclusões Finais	Unidades de Saída
Fatos Fundamentais	Unidades de Entrada
Conclusões Intermediárias	Unidades Ocultas
Dependências	Conexões ponderadas

Há duas limitações principais no conjunto de regras utilizado por KBANN:

- os algoritmos de aprendizado de RNs utilizados não tratam variáveis do cálculo de predicados e, por isso, as regras devem ser proposicionais;

- as regras devem ser acíclicas, para simplificar o treinamento das redes resultantes.

Além destas limitações, os conjuntos de regras são geralmente estruturados hierarquicamente, ou seja, as regras fornecem conclusões intermediárias que descrevem conjunções úteis das características de entrada. Estas conclusões intermediárias podem ser utilizadas por outras regras para determinar a conclusão final ou outra conclusão intermediária. Esta estrutura hierárquica é que cria as características derivadas para uso do sistema de aprendizado baseado em exemplos.

3.1.2 Introdução de Regras na Rede Neural

A transformação de um conjunto de regras para uma rede KBANN é feita através de sete passos descritos a seguir:

1. Reescrita

Este primeiro passo tem como objetivo explicitar a estrutura hierárquica do conjunto de regras, tornando possível a tradução direta das regras para uma rede neural. Para tal, as disjunções devem ser expressas como regras múltiplas com o mesmo conseqüente, cada uma com apenas um antecedente.

Se houver mais de uma regra para um conseqüente, então cada regra para este conseqüente com mais de um antecedente deve ser reescrita como duas regras. Uma dessas regras tem o conseqüente original e um único antecedente, correspondente a um novo termo gerado neste processo. A outra regra tem como conseqüente este novo termo gerado e, como antecedente, os antecedentes da regra original.

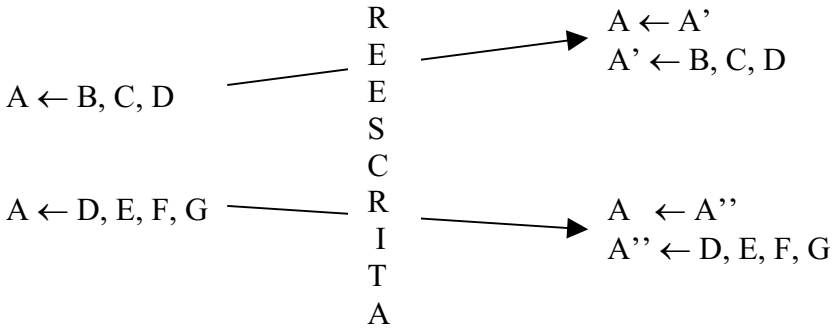


FIGURA 3.2 - Etapa de Reescrita na transformação de regras para uma rede neural

2. Mapeamento

Nesta etapa, é necessário estabelecer um mapeamento entre um conjunto de regras transformado e uma rede neural. Utilizando a tabela 3.1 de correspondências entre uma base de conhecimento e uma rede neural, a KBANN cria uma rede com uma correspondência de um-para-um com os literais do conjunto de regras.

Os pesos e os níveis de limiar são especificados de forma que a rede tenha os mesmos resultados que as regras nas quais se baseou.

No final deste passo, a rede KBANN tem informações do conjunto de regras relativas às entradas relevantes e características derivadas. Entretanto, não há garantia de que o conjunto de regras faça referência a todas as características relevantes ou forneça um conjunto significativo de características derivadas. Os próximos passos, então, aumentam a rede KBANN com novas ligações, unidades de entrada e até mesmo neurônios escondidos.

A figura abaixo mostra uma rede neural resultante da aplicação dos passos 1 e 2 em um conjunto de regras.

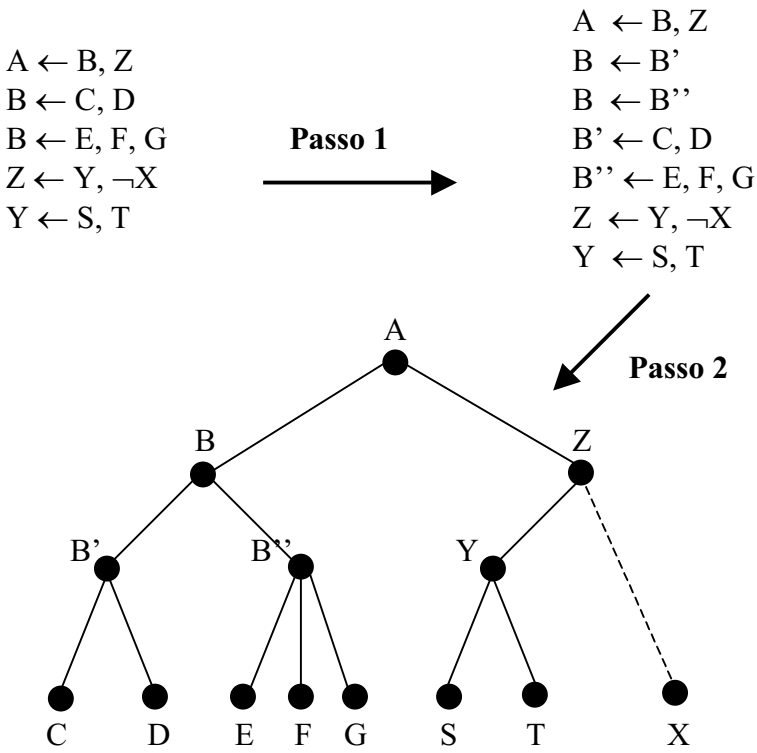


FIGURA 3.3 - Aplicação dos passos 1 e 2 e RN resultante

3. Numeração

Cada unidade que representa um conseqüente na rede deve ser numerada segundo seu nível hierárquico. KBANN define o nível de cada unidade sendo o tamanho do maior caminho a uma unidade de entrada.

Esta técnica de numeração assume que toda cadeia de raciocínio esteja completa, ou seja, toda conclusão intermediária é uma parte de um caminho direto de uma ou mais entradas para uma ou mais saídas. Entretanto, não há exigência de que toda cadeia de raciocínio seja completa. Para resolver este problema, basta anexar os antecedentes desconectados a todas as unidades de entrada e conseqüentes desconectados a todas as unidades de saída com pesos bem baixos.

4. Adição de neurônios ocultos

Opcionalmente, pode-se adicionar unidades ocultas para dar à rede KBANN a capacidade de aprender características não especificadas no conjunto de regras inicial, mas que tenham sido sugeridas por um especialista.

5. Adição de neurônios de entrada

Neste passo, a rede é expandida com características de entrada não referenciadas nas regras, mas que são consideradas relevantes por um especialista. Isto é necessário devido ao fato de que um conjunto de regras que não é completo pode não identificar toda característica de entrada para o aprendizado completo e consistente do conjunto de treinamento.

6. Adição de arcos

Arcos com peso zero são adicionados com a ligação de todas as unidades de um nível *n* a todas as unidades do nível *n*-1, utilizando-se a numeração estabelecida na etapa 3.

A figura abaixo mostra a aplicação dos passos 3 a 6 no exemplo apresentado na figura 3.3:

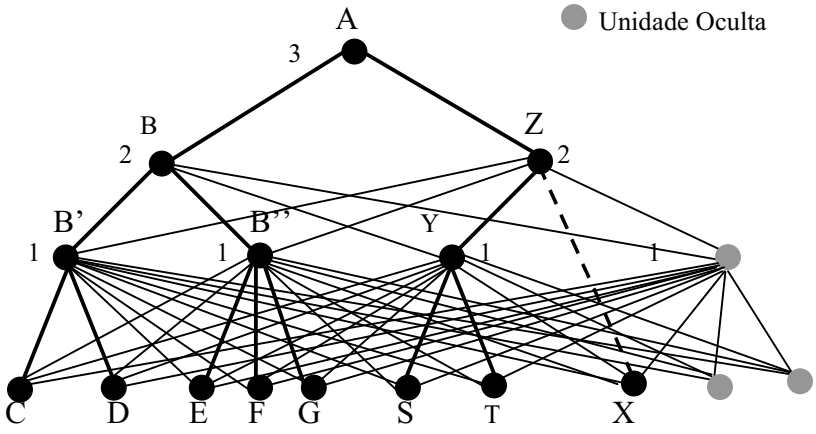


FIGURA 3.4 - Aplicação dos passos 3 a 6

7. Perturbação

O passo final é a perturbação de todos os pesos da rede adicionando um número aleatório a cada um. Esta perturbação é muito pequena para afetar o comportamento da rede antes do treinamento, mas é suficiente para evitar problemas causados por simetria.

3.1.3 Refinamento da KBANN

O treinamento da KBANN se dá através do algoritmo de *Backpropagation*, um método de aprendizado neural padrão. Entretanto, como a rede começa com saídas binárias (0 ou 1), o algoritmo produz variações muito pequenas, mesmo que a resposta de uma unidade de saída seja incorreta.

Uma solução possível é utilizar a *entropia cruzada* (Equação 3.1) como função de custo E a ser minimizada pelo algoritmo, ao invés do *erro médio quadrado* (Equação 3.2). A função de *entropia cruzada* interpreta o sinal de treinamento e as saídas da rede como probabilidades condicionais e tenta minimizar a diferença entre estas probabilidades.

$$E = - \sum_{i=1}^{N} [(1 - d_i) * \log_2(1 - a_i) + d_i * \log_2(a_i)] \qquad \text{Equação 3.1}$$

$$Erro = \frac{1}{N} \sum_{i=1}^{N} (d_i - a_i)^2 \qquad \text{Equação 3.2}$$

3.2 Sistema INSS

Sistemas híbridos simbólico-conexionistas, como o sistema KBANN [TOW 94], exploram sua capacidade de utilizar ao mesmo tempo conhecimento da teoria (representado por um conjunto de regras simbólicas) e conhecimento empírico (conjunto de exemplos observados). Estes sistemas permitem uma transferência bidirecional entre os módulos simbólico e conexionista, como mostra a figura a seguir [OSO 98]:

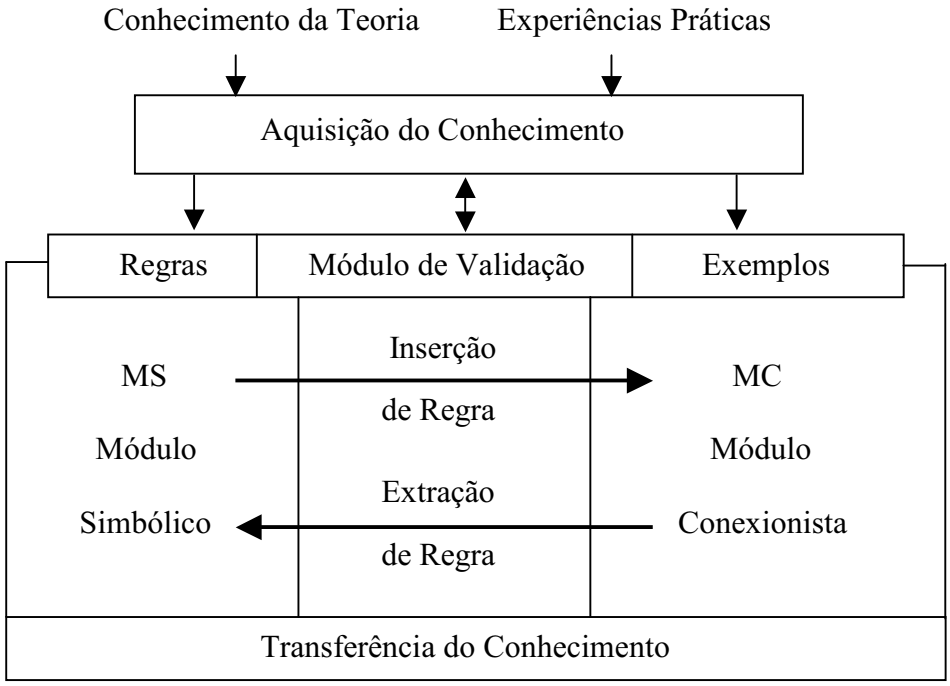


FIGURA 3.5 - Transferência de conhecimento em sistemas híbridos simbólico-conexionistas

O sistema KBANN é capaz de organizar uma base de conhecimento na forma de uma rede neural, aprendendo a partir de exemplos e, em seguida, extraindo novas regras. Apesar disso, este sistema possui alguns empecilhos devido à escolha de seu modelo de RN e do método de aprendizagem, o algoritmo *Backpropagation*.

Surgiu, então, o sistema INSS [OSO 98] (*Incremental Neuro-Symbolic System*) para melhorar a rede KBANN e superar suas principais limitações. Este sistema realiza os mesmos processos do KBANN, porém cada um deles é realizado incrementalmente. Além disto, o algoritmo de aprendizagem é o *Cascade-Correlation* que funciona adicionando novos neurônios durante o aprendizado.

3.2.1 Descrição do Sistema

O sistema INSS é composto de cinco módulos:

- Módulo Simbólico (Máquina de Inferência Simbólica)

Este módulo consiste da linguagem CLIPS (*C Language Integrated Production System*) desenvolvida pelo STB-NASA.

O sistema dispõe de facilidades para a transferência de regras e exemplos entre a sintaxe específica utilizada pela linguagem CLIPS e a sintaxe usada nas suas ferramentas (NeuComp, NeuSim e Extract).

- NeuComp

O módulo NeuComp é responsável pela construção da rede neural a partir de regras. Nele se encontra o processamento de regras de produção simples de ordem 0 que são equivalentes a formas Se-Então, tais como:

Se <condição> (verdadeiro/falso) E/OU

<condição> (verdadeiro/falso)...

Então <conclusão>

O resultado da tradução das regras é uma rede composta de um conjunto de unidades ligadas a conexões com pesos. Este processo segue o método descrito por [TOW 94]. Antes do aprendizado, esta rede produz como saída os mesmos resultados obtidos pelo conjunto de regras processado inicialmente pelo sistema.

Além das regras utilizadas pelo sistema KBANN, o sistema INSS permite regras de produção da ordem 0^{+}, ou seja, regras que incluem intervalos de valores. Para isso foi preciso implementar funções de comparação como *MaiorQue*, *MenorQue* e *Igual* e, como resultado, obteve-se regras do tipo:

Se MaiorQue(Sensor_S1, 1.0) E

MenorQue(Sensor_S1, Sensor_S2)

Então Conclusão_C1

Como as regras simbólicas permitem algum estabelecimento inicial de conhecimento e com isso oferecem uma estrutura à rede, consegue-se resolver dois problemas relacionados às RNs: simplificação da escolha do número e distribuição dos neurônios e obtenção de uma boa designação aos valores iniciais dos pesos das conexões.

- NeuSim

Este módulo consiste do aprendizado da rede neural através do algoritmo *Cascade-Correlation*, ao invés do algoritmo *Backpropagation* utilizado pelo sistema KBANN.

O algoritmo *Cascade-Correlation* permite um aprendizado mais rápido com melhores resultados de desempenho. A figura 3.6 mostra como exemplo a evolução da estrutura da rede quando se aplica este algoritmo de aprendizagem.

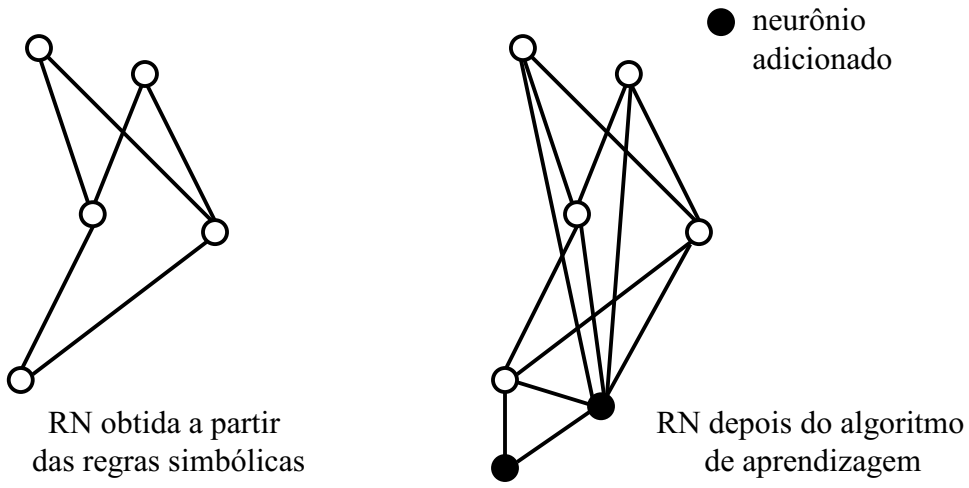


FIGURA 3.6 - Evolução da estrutura de uma RN usando o algoritmo *Cascade-Correlation*

Ao contrário de algoritmos estáticos, como o *Backpropagation*, o *Cascade-Correlation* é uma técnica de geração para construção e aprendizado de uma rede neural. Ao invés de simplesmente ajustar os pesos em uma rede de topologia fixa, ele inicia com uma rede mínima composta de neurônios de entrada e saída. Durante o aprendizado, o algoritmo verifica se a rede está reduzindo o seu erro com a topologia atual. Em caso negativo, ele adiciona um neurônio oculto cujas ativações de saída se correlacionam melhor sobre todos os casos de treinamento com o erro existente da rede. O algoritmo de aprendizagem reduz o erro de saída da rede passo a passo através de um processo cíclico de aprendizagem dos neurônios de saída e adição/aprendizagem de neurônios ocultos. Esta característica de adição de novos neurônios permite que se possa completar, modificar ou refinar o conhecimento inicial dado à rede neural.

- Extract

O módulo de extração das regras é uma versão melhorada do algoritmo *SUBSET* de extração de regras em RNs. A melhoria está no fato de que o processo de extração é mais simples porque só uma parte da rede é analisada, não sendo necessário obter todo o conhecimento e sim, o conhecimento adquirido. Além disso, foram desenvolvidas heurísticas para simplificação da rede, o que reduz a complexidade do procedimento de extração.

- Valid

O módulo de validação identifica exemplos e regras provavelmente incorretas. Depois disto, estas inconsistências devem ser submetidas a um analista.

A figura 3.7 apresenta as principais relações entre os módulos do sistema INSS:

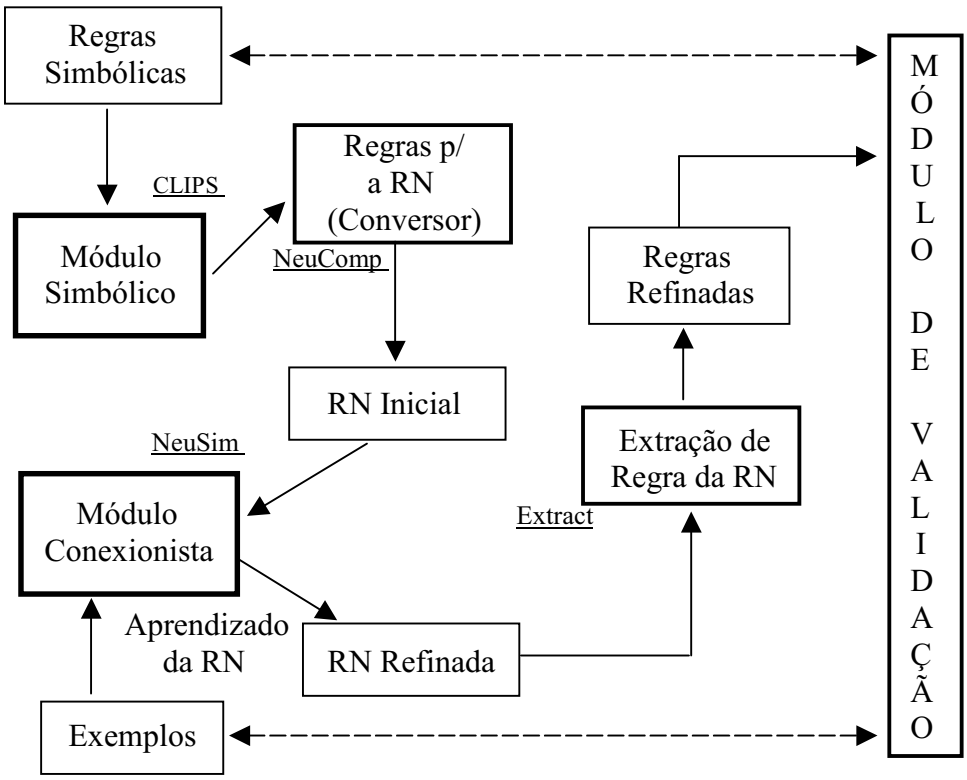


FIGURA 3.7 - Módulos do Sistema INSS

3.2.2 Vantagens do Sistema INSS

O sistema INSS apresenta algumas vantagens em relação ao sistema KBANN e suas melhorias permitem eliminar alguns empecilhos:

- O INSS permite trabalhar com regras simbólicas incompletas e até mesmo com conjunto de regras incorretas. O sistema KBANN restringe o aprendizado a mudanças menores no conjunto de regras, enquanto que o INSS pode adicionar novas regras ou realizar mudanças nas já existentes;

- O algoritmo de aprendizagem *Cascade-Correlation* é mais rápido que o baseado em *Backpropagation* do sistema KBANN. Além disso, ele permite a construção incremental da rede, melhorando os pesos das conexões e a topologia da rede;

- O algoritmo de extração não necessita analisar toda a estrutura da RN. Analisando somente os novos neurônios obtém-se o conhecimento adquirido atual, o que reduz a complexidade do processo de extração;

- O sistema permite o processamento de regras simbólicas de ordem 0^+, não sendo necessário restringir o sistema ao uso de entradas binárias ou a pré-processar entradas contínuas com o objetivo de discretizá-las.

3.3 Modelo Neural Combinatório

O Modelo Neural Combinatório (MNC) introduzido por Machado [MAC 89] constitui-se de uma arquitetura híbrida para sistemas inteligentes que integra conhecimento simbólico e não-simbólico. O modelo é capaz de reconhecer padrões de dados simbólicos de alta dimensão, realizando mapeamentos complexos deste espaço de entrada para um espaço de saída menor. Isto é feito através da utilização de correlações de alta ordem que incorporam relações invariantes entre os dados de entrada e as hipóteses de saída. O modelo foi inspirado a partir das redes neurais, lógica *fuzzy* e engenharia do conhecimento.

Este modelo apresenta características que são desejáveis em um sistema de classificação [BEC 98]:

- simplicidade do aprendizado da rede neural: o modelo aproveita a capacidade de generalização inerente às redes neurais em um mecanismo que exige o mínimo de parametrização por parte do usuário, em comparação com outros modelos neurais;

- capacidade de explanação: o MNC é capaz de mapear o conhecimento da rede neural para uma representação simbólica;

- treinamento em alta velocidade: o aprendizado é realizado com uma única passada sobre os dados;

- possibilidade de aprendizado incremental: o conhecimento extraído anteriormente pode ser melhorado com novos exemplos;

- lida com a incerteza de forma flexível: é possível a extração de medidas de confiança das regras a partir dos pesos no modelo neural;

- facilidade para integração de diversas fontes de conhecimento;

- habilidade para a identificação de padrões em espaços multidimensionais, realizando mapeamentos para espaços de saída de menores dimensões.

3.3.1 Descrição do Modelo

Como é freqüente em sistemas conexionistas, o modelo é formado por um conjunto de unidades de processamento (células ou neurônios) que são conectadas em uma rede de acordo com uma topologia definida. Estas unidades executam um processamento muito simples, que podem ser de dois tipos: *células AND* e *células OR*.

Neste modelo, cada neurônio representa um conceito simbólico do domínio do problema a ser tratado, tais como hipóteses, evidências, etc. Cada célula calcula um valor numérico pertencente ao intervalo [0, 1], chamado seu *estado de ativação*, baseado na informação de entrada da célula. A saída da célula é seu estado de ativação, que pode ser interpretado como o grau de confiança do conceito representado pela célula. Quando o neurônio opera no modo *booleano*, seu estado de ativação só pode ter os valores 0 e 1. No caso de o valor pertencer ao intervalo [0, 1], ele trabalha no modo *fuzzy*.

A rede neural é formada através de arcos (“sinapses”) que conectam as unidades de processamento. Estas sinapses são caracterizadas por pesos que variam de 0 (não conectada) a 1 (totalmente conectada). Estes pesos funcionam como fator atenuante do sinal de entrada, que é um valor numérico entre 0 e 1 vindo do usuário ou da saída de um neurônio. Quando a sinapse atua diretamente no sinal de entrada é denominada como *sinapse excitatória*. Caso contrário, ela aplica a negação *fuzzy* no sinal de entrada e é denominada como *sinapse inibitória*. A saída do sinal transmitido por uma sinapse é chamada de *Fluxo Evidencial*.

No MNC, a rede não permite ciclos (rede *feedforward*) e possui pelo menos três camadas:

- Camada de Entrada: recebe dados simbólicos do usuário ou do ambiente que possuem uma crença que varia de 0 a 1;

- Camadas Intermediárias: são camadas combinatórias compostas de *células AND* que representam diferentes combinações ou padrões dos dados de entrada;

- Camada de Saída: formada por *células OR* que representam as diferentes hipóteses existentes no domínio do problema. Estas células implementam o mecanismo de competição entre os diferentes caminhos vindos das camadas mais baixas.

O modelo lida com as duas variáveis fundamentais do processamento do conhecimento: a adesão entre variáveis simbólicas representando conceitos e a crença que o especialista coloca em cada conceito. Além disso, ele pode ser caracterizado pelas propriedades das unidades de processamento, da rede e propriedades dinâmicas, que serão discutidas nas próximas seções.

3.3.2 Unidades de Processamento

A figura 3.8 apresenta os dois tipos básicos das unidades de processamento: *células AND* e *OR*:

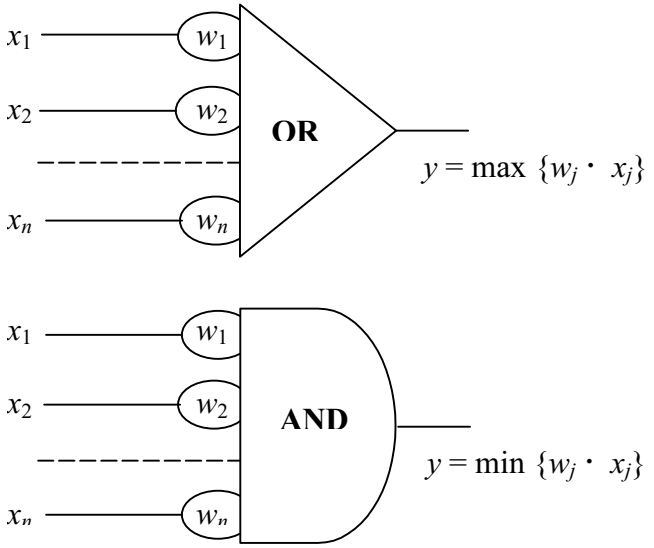


FIGURA 3.8 - Células OR e AND (com sinapses excitatórias)

- x_j ($j = 1..n$) representa as entradas para as células. A entrada de um neurônio pode ser informações do usuário ou a saída de outra célula. Os valores possíveis são de 0 a 1.

- y é o estado de ativação calculado e também a saída da célula. Possui um valor contínuo que pode variar de 0 a 1 e que expressa o grau de possibilidade de um conceito representado pelo neurônio.

- w_i representa o peso da conexão da entrada i até a unidade. Este peso pode ser interpretado como o grau de pertinência do conceito representado na entrada ao conceito representado na saída.

O estado de ativação das células OR e AND é calculado baseado nas entradas $x_1, x_2, ..., x_n$, respectivamente:

$$y_{OR} = \max \{ w_i \cdot x_i \cdot t_i + w_i \cdot (1\text{-}x_i) \cdot (1 - t_i) \}$$
$$y_{AND} = \min \{ w_i \cdot x_i \cdot t_i + w_i \cdot (1\text{-}x_i) \cdot (1 - t_i) \}$$

onde $t_i = 1$ para sinapse excitatória e $t_i = 0$ para sinapse inibitória..

Pode-se definir um modelo de neurônio mais simples no qual os pesos das sinapses são iguais a 1 e as células operam no modo *booleano*. Neste modelo, as células funcionariam como portas lógicas *booleanas* AND e OR.

3.3.3 Propriedades da Rede

A figura 3.9 apresenta uma rede neural combinatória simples, composta de três camadas.

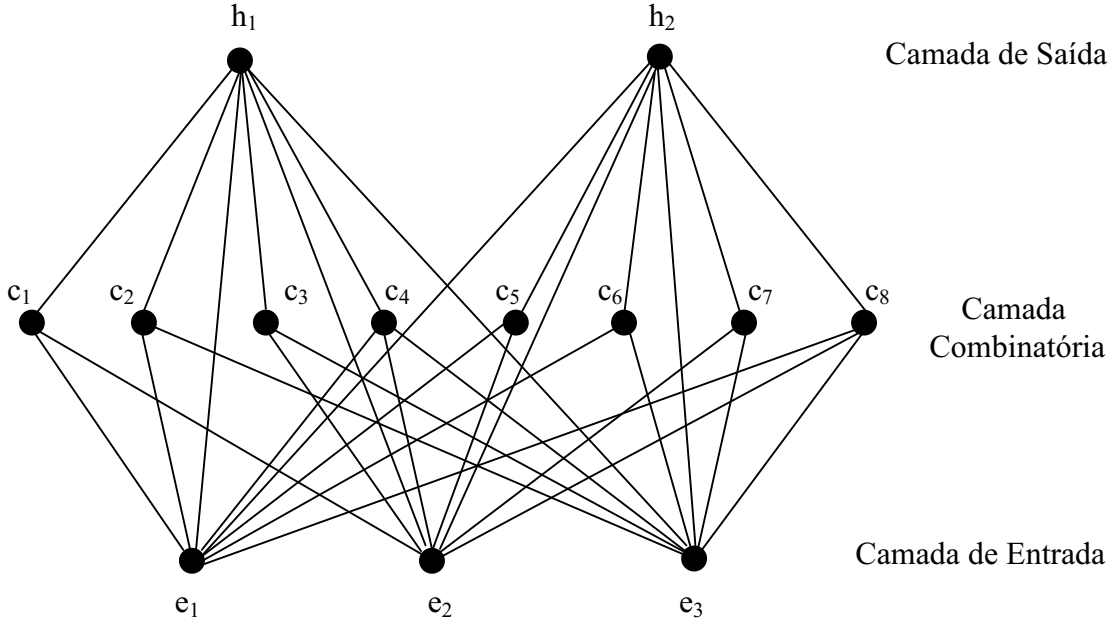


FIGURA 3.9 - Versão completa do MNC para 3 evidências de entrada e 2 hipóteses de saída

Os próximos tópicos mostram as principais características da rede.

- Entrada de Dados Simbólicos

Cada neurônio da camada de entrada representa um conceito de evidência na forma de uma tripla: objeto, atributo e valor. É importante ressaltar que uma evidência da entrada pode ter valores distintos e que devem ser tratados separadamente pela rede. Com isso, cada valor possível é representado por um neurônio diferente. O grau de confiança destes valores correspondem aos estados de ativação dos neurônios de entrada.

No modelo, as células somente enviam os valores para as camadas superiores, não realizando nenhum processamento destes sinais. Esta abordagem apresenta algumas habilidades:

- Representação da situação de ignorância, ou seja, a evidência não está disponível. Podem ser utilizadas entradas com valor 0.

- Codificação de atributos complexos envolvendo múltiplos valores possíveis.

- Raciocínio não-monotônico ao representar somente evidências positivas. Não há representação de evidências negativas. Utiliza-se a suposição de "mundo fechado" para tratá-las: se não há crença de que a evidência é positiva, então ela é negativa e não é necessário representá-la.

- Permite que o usuário trabalhe com três valores possíveis (positivo, negativo e desconhecido) ou com raciocínio probabilístico (quando a soma das crenças dos diferentes valores de um atributo é igual a 1) ou com raciocínio não probabilístico (quando a soma é diferente de 1).

- Associação dos Dados Simbólicos

A Camada Combinatória é uma camada oculta que representa diferentes combinações dos dados de entrada. A versão completa do modelo inclui todas as combinações das evidências da entrada desde C(n, 2) até C(n, n), onde n é o número de células da camada de entrada.

Não foi feita a inclusão das combinações de tamanho 1 para evitar repetição desnecessária de células. Além disso, limitou-se o tamanho das combinações com tamanho menor ou igual a um limiar m. Machado [MAC 89] sugere um critério baseado no "número mágico" de Miller, sete mais ou menos dois, para estabelecer o limite superior da ordem das combinações.

Estas limitações são necessárias devido à explosão combinatória da versão completa do MNC que requer ($2^n - n$) células na camada oculta, o que é bastante inviável no caso de várias células na camada de entrada.

3.3.4 O Processo de Aprendizagem

A rede opera em dois modos principais:

- Modo de consulta

No modo de consulta as células operam em paralelo. As evidências disponíveis juntamente com suas crenças são propagadas através da rede desde a camada de entrada até os nodos de hipótese, de acordo com as regras de inferência apresentadas anteriormente para os neurônios AND e OR.

O tempo computacional total é diretamente proporcional ao número de camadas, já que se trata de uma rede *feedforward*.

- Modo de aprendizagem

A rede constrói representações internas complexas de seu ambiente usando um mecanismo de punição e recompensa.

O algoritmo de aprendizagem encontra pesos que fazem com que o modelo apresente o comportamento desejado baseado em um conjunto de exemplos. Este algoritmo deve ser capaz de modificar os pesos de forma que as unidades internas que não são parte da entrada e nem da saída possam representar características importantes do domínio da tarefa. A seguir apresenta-se o algoritmo de aprendizagem do MNC.

Algoritmo de aprendizagem

O algoritmo parte de uma rede não treinada com a topologia descrita anteriormente e de um conjunto de treinamento contendo exemplos com sua classificação correta. O aprendizado é feito de forma supervisionada e é capaz de ser concluído em apenas um passo sobre todo o conjunto de treinamento.

Considerando-se todos os pesos da rede não treinada iguais a 1, tem-se o algoritmo:

TABELA 3.2 - Algoritmo de Aprendizagem do MNC

Atribua a cada arco da rede um acumulador com valor inicial 0;

Para cada exemplo do conjunto de treinamento faça:

Propague as crenças dos nodos de entrada até a camada de hipóteses;

Para cada arco que alcança um nodo de hipótese faça:

Se o nodo de hipótese corresponde à classe correta do exemplo

Então propague de volta deste nodo até os nodos de entrada aumentando o acumulador de cada arco atravessado por este fluxo evidencial (Recompensa);

Senão propague de volta do nodo de hipótese até os nodos de entrada diminuindo o acumulador de cada arco atravessado por este fluxo evidencial (Punição)

Este algoritmo produz uma rede treinada que corresponde à rede inicial com números variando de $-T$ a $+T$ anexados aos seus arcos, onde T é o número de exemplos do conjunto de treinamento.

Depois do processamento deste algoritmo, ainda é necessário que a rede passe por um mecanismo de poda e normalização. O objetivo da poda é remover todos os arcos fracos e sinapses negativas (aquelas em que o valor dos acumuladores é baixo ou negativo).

A seguir, apresenta-se o algoritmo que, baseado em um limiar, é capaz de podar e normalizar a rede de forma a torná-la operacional:

- Remova da rede todos os arcos cujos acumuladores são menores do que um limiar;

- Remova todas as células das camadas de entrada e combinatória que não estão mais conectadas às células de hipótese;

- Atribua novos valores aos pesos dos arcos iguais aos valores obtidos dividindo-se os acumuladores pelo maior valor acumulador da rede.

A escolha do limiar de poda é crucial para o desempenho da rede. Ele pode ser determinado empiricamente testando o modelo sistematicamente para vários valores de limiar e selecionando a rede que obtém o melhor desempenho.

Exemplo

Este exemplo mostra como o MNC se comporta para diagnóstico médico. O conjunto de treinamento contém casos de pacientes com as doenças D1 ou D2 e apresentando alguns dos sintomas s1, s2, s3 ou s4.

TABELA 3.3 - Pacientes com doenças e sintomas associados

Nome	Sintomas	Doença
John	s1, s2, s3	D1
Diana	s1, s2, s4	D1
Mary	s1, s3, s4	D2
Peter	s2, s3, s4	D2

Dado este conjunto de treinamento, o algoritmo parte de uma rede com quatro células de entrada (representam os sintomas) e uma camada intermediária com combinações de tamanho menor ou igual a três (cada paciente possui no máximo três sintomas neste exemplo). Neste caso, a camada intermediária conterá 28 células.

A tabela 3.4 apresenta a aplicação do algoritmo de aprendizagem nos acumuladores e o efeito da poda com limiares 1 e 2.

TABELA 3.4 - Treinamento e poda do Modelo Neural Combinatório

	Sintomas					Acumuladores				Limiar/Poda	
Doença	s1	s2	s3	s4	Início	John	Diana	Mary	Peter	1	2
D1	X				0	1	2	1	1	1	-
		X			0	1	2	2	1	1	-
			X		0	1	1	0	-1	-	-
				X	0	0	1	0	-1	-	-
	X	X			0	1	2	2	2	2	2
	X		X		0	1	1	0	0	-	-
	X			X	0	0	1	0	0	-	-
		X	X		0	1	1	1	0	-	-
		X		X	0	0	1	1	0	-	-
			X	X	0	0	0	-1	-2	-	-
	X	X	X		0	1	1	1	1	1	-
	X	X		X	0	0	1	1	1	1	-
	X		X	X	0	0	0	-1	-1	-	-
		X	X	X	0	0	0	0	-1	-	-
D2	X				0	-1	-2	-1	-1	-	-
		X			0	-1	-2	-2	-1	-	-
			X		0	-1	-1	0	1	1	-
				X	0	0	-1	0	1	1	-
	X	X			0	-1	-2	-2	-2	-	-
	X		X		0	-1	-1	0	0	-	-
	X			X	0	0	-1	0	0	-	-
		X	X		0	-1	-1	-1	0	-	-
		X		X	0	0	-1	-1	0	-	-
			X	X	0	0	0	1	2	2	2
	X	X	X		0	-1	-1	-1	-1	-	-
	X	X		X	0	0	-1	-1	-1	-	-
	X		X	X	0	0	0	1	1	1	-
		X	X	X	0	0	0	0	1	1	-

As figuras 3.10 e 3.11 mostram a rede após a operação de poda com os limiares 1 e 2.

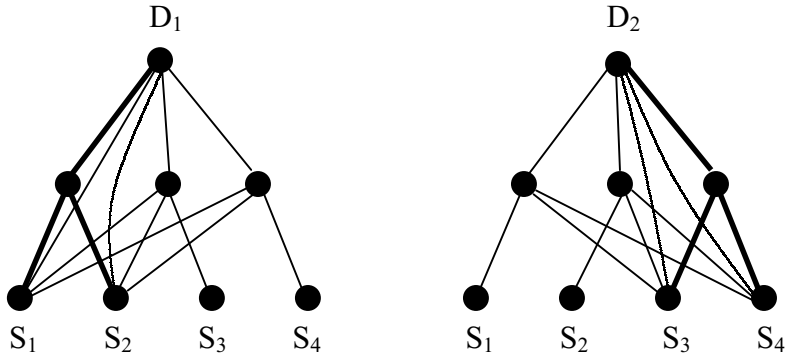


FIGURA 3.10 - Rede após a poda com limiar 1

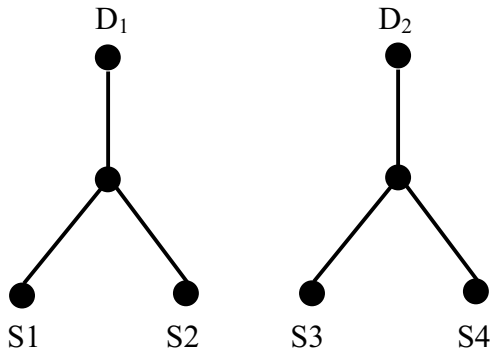


FIGURA 3.11 - Rede após a poda com limiar 2

3.3.5 Otimizações

A principal limitação do MNC é a possibilidade da explosão combinatória, já que a camada intermediária pode crescer exponencialmente. A explosão combinatória é crítica devido às restrições de processamento e memória dos computadores. Não é possível gerar todas as hipóteses do problema e, subseqüentemente, avaliar quais devem ou não permanecer. Beckenkamp et. al. [BEC 98] apresentam em seu trabalho algumas contribuições para este problema:

- Separação de evidências por hipóteses

Durante a geração dos grafos de conhecimento, são definidas quais evidências devem ser consideradas. Pode-se determinar quais evidências relacionam-se com cada hipótese do problema. Isto significa que para algumas hipóteses, um número pequeno de evidências pode ser considerado. Como a geração da estrutura da rede neural do MNC é independente para cada hipótese definida, algumas podem ter a explosão combinatória reduzida.

Isto acontece, por exemplo, em um problema de análise de crédito, no qual pode-se determinar que a evidência *sexo* é importante na avaliação de maus clientes, mas não na avaliação de bons clientes. Com isso, considerando-se um conjunto distinto de evidências relevantes para cada hipótese, pode-se reduzir significativamente o espaço de busca.

- Evitar combinações sem sentido

Não faz sentido gerar combinações de valores possíveis de uma mesma evidência. Por exemplo, suponhamos que se tenha uma evidência *Tipo Sangüíneo* em um problema de diagnóstico médico. Cada tipo sangüíneo é um possível valor para esta evidência. O MNC irá gerar para cada tipo um neurônio de entrada. Neste caso, combinações sem sentido são aquelas em que mais de um tipo sangüíneo é considerado para o mesmo paciente. E não há motivo para realizar esta combinação, já que um paciente nunca terá mais de um tipo sangüíneo.

O MNC original não considera o tipo de situação acima, apesar de estar claro a não funcionalidade dessas combinações sem sentido. Provavelmente isto tenha sido feito para simplificar sua implementação, mas compromete a eficiência do sistema.

- Paralelismo e distribuição

Como a estrutura combinatória de toda a rede do MNC é independente para cada hipótese, pode-se criar uma ou mais *threads* para gerenciar o processo de cada hipótese. Assim, o conjunto de combinações geradas será dividido nas *threads* que foram definidas. Esta solução otimiza o uso de *threads* de acordo com o tamanho da camada combinatória e as capacidades da máquina.

O algoritmo de aprendizagem do MNC também pode ser distribuído. A idéia é realizar o aprendizado de partes da rede em várias máquinas e, em seguida, unir seus resultados. O critério para a divisão de partes da rede a serem treinadas em diferentes máquinas é baseado no mesmo critério da definição das *threads* para a rede, além de aspectos de capacidade de memória e processamento das máquinas.

Outros problemas importantes apresentados pelo MNC, segundo Prado *et. al.* [PRA 98b], incluem: (a) geração da rede completamente vazia antes de se iniciar o treinamento; e (b) combinação de valores diferentes de um mesmo atributo, o que não é razoável na maioria das aplicações. Estes problemas vêm recebendo contribuições significativas, como [FEL 97], [BEC 98] e [PRA 99], que permitem a obtenção de melhor desempenho deste modelo para Mineração de Dados.

Além dos principais problemas do MNC, a expressividade das regras obtidas é limitada à da Lógica de Predicados com apenas uma variável. Esta questão é a principal motivação para o desenvolvimento da *FOLONET*, que será descrita no próximo capítulo.

4 Modelo Neural para Aprendizado Relacional

A PLI tem demonstrado as vantagens de seu aparato de aprendizado em relação a outras abordagens, como por exemplo, aquelas baseadas em aprendizado atributo-valor. Os algoritmos de aprendizado de PLI apresentam alta expressividade, porém sofrem com a grande complexidade de seus processos, principalmente o teste de cobertura das variáveis.

Por outro lado, as Redes Neurais Artificiais introduzem um ótimo desempenho devido à sua natureza paralela. O maior problema em relação às RNs é que geralmente são "caixas pretas", o que torna difícil a obtenção de uma interpretação razoável da estrutura geral da rede na forma de construções lógicas de fácil compreensão.

Tem sido mostrado que a combinação de métodos simbólicos e conexionistas é promissora em aprendizado de máquina [SHA 94], [SHA 96].Várias abordagens híbridas simbólico-conexionistas já foram propostas, como o MNC [MAC 89], KBANN [SHA 94], [TOW 94] e o sistema INSS [OSO 98]. Entretanto, estas abordagens ainda lidam com representações atributo-valor.

Com isso, despontou-se um interesse em estudar um modelo que combinasse a expressividade obtida pelos algoritmos de aprendizado da PLI com o desempenho de Redes Neurais. Surgiu, então, uma rede neural híbrida para aprendizado relacional: a *FOLONET – First Order Neural Network*.

4.1 Introdução

A lógica de primeira ordem está baseada em uma estrutura matemática envolvendo predicados. Os predicados podem ser utilizados para denotar propriedades de objetos e também relações entre eles. Por exemplo, a sentença "*X* ama *Y*" pode ser denotada por um predicado *ama*(*X*, *Y*), que relaciona duas pessoas representadas pelas variáveis *X* e *Y*.

Cada predicado possui uma *aridade* associada a ele, ou seja, o número de argumentos que ele tem. Quando o predicado possui apenas um argumento, ele é chamado de predicado monádico. Por exemplo, o predicado monádico *pequeno*(*X*), definido sobre objetos de um domínio **X**, cujos elementos são representados genericamente pela variável *X*, opera sobre um único objeto e está relacionado a uma propriedade do mesmo (propriedade *pequeno*). Quando os predicados representam relações entre objetos, o número de argumentos é 2 (predicado binário) ou maior que 2 (predicado *n*-ário). Como exemplo, pode-se considerar a relação *gosta_de*(*X*, *Y*), representando a relação entre objetos pertencentes a dois domínios **X** (pessoas) e **Y** (coisas). Neste caso, a variável *X* corresponde a uma pessoa e *Y* corresponde a um presente, por exemplo. O predicado, então, é binário e está relacionando objetos.

Um predicado separa um domínio em dois conjuntos de objetos: aqueles que satisfazem o predicado e aqueles que não o satisfazem. Cada objeto ou tupla (conjunto de objetos) pode ser identificado pelo grau de pertinência aos grupos de objetos que satisfazem um determinado predicado. No caso do predicado *pequeno*(*X*), podemos representar este grau de pertinência por $\mu_{pequeno}(X)$. Se o grau de pertinência for *booleano*, teremos $\mu_{pequeno}(X) \in \{0, 1\}$.

A pertinência dos objetos a um conceito pode ter uma intensidade intermediária, ou seja, com valor no intervalo [0, 1]. Um grau de pertinência entre 0 e 1 indica uma satisfação parcial do objeto ou da tupla ao conceito relacionado com o predicado em questão. Neste caso, a lógica difusa (*fuzzy*) fornece as ferramentas para a interpretação destes conceitos.

A lógica de primeira ordem opera sobre uma base de conhecimento formada por predicados válidos num determinado contexto, utilizando regras de inferência para induzir novos predicados. Este novos predicados são descritos em geral na forma de cláusulas contendo os predicados do conhecimento prévio.

Um predicado alvo pode ser definido por um conjunto de exemplos positivos, que satisfazem o predicado alvo, e negativos, que não o satisfazem. Através deste conjunto e de regras de inferência apropriadas é possível, através de manipulação algébrica, descrever o predicado alvo a partir dos predicados do conhecimento prévio.

A seguir será apresentada a rede *FOLONET* que permite a descrição de predicados alvo a partir de conhecimento prévio formado de predicados válidos previamente conhecidos e de um conjunto de treinamento, composto de exemplos positivos e negativos (contra-exemplos) do predicado alvo.

4.2 Descrição da Rede FOLONET

O modelo da rede *FOLONET* foi inspirado na estrutura do Modelo Neural Combinatório, apresentado anteriormente. Ele é formado por um conjunto de neurônios que são conectados de acordo com a topologia definida pela rede.

O processamento dos neurônios é muito simples e pode ser de dois tipos: *células AND* e *células OR*. Cada neurônio representa um conceito simbólico do domínio do problema a ser tratado, tais como hipóteses, evidências, etc. Além disso, os neurônios operam no modo *booleano*, ou seja, seu estado de ativação só pode ter os valores 0 e 1.

Assim como no MNC, a rede *FOLONET* não permite ciclos (rede *feedforward*) e possui três camadas:

- Camada de Entrada: recebe dados simbólicos (predicados) do usuário ou do ambiente. Neste primeiro modelo, a confiança nos dados iniciais é total, não sendo necessária a entrada do grau de confiança. Isto se deve ao fato de se estar trabalhando com dados determinísticos inicialmente.

- Camada Intermediária: é uma camada combinatória composta de *células AND* que representam diferentes combinações ou padrões dos dados de entrada. Células que pertencem a esta camada representam abstrações intermediárias capazes de reduzir a complexidade computacional da tarefa de classificação.

- Camada de Saída: formada por *células OR* que representam as diferentes hipóteses existentes no domínio do problema.

Além destas camadas da rede neural, a *FOLONET* possui uma estrutura flexível para entrada dos exemplos do conjunto de treinamento. Isto é necessário devido ao fato de que cada predicado envolvido no processo de aprendizado pode ser monádico, binário ou *n*-ário. O conjunto de treinamento contém exemplos positivos e negativos do predicado alvo e, com isso, cada exemplo contém um número de variáveis de acordo com a aridade do predicado alvo. Estes exemplos serão utilizados para verificar se satisfazem ou não os predicados do conhecimento prévio no treinamento da rede. Para tal, é preciso que se tenha sempre um número de variáveis que coincida com a maior aridade presente no conhecimento prévio. Neste primeiro modelo, o conhecimento prévio só contém predicados que tenham a mesma aridade (ou menor que a) do predicado alvo.

4.2.1 Treinamento da Rede

Na rede *FOLONET*, a camada de entrada é composta de nós que implementam os predicados. A rede procura representar um predicado alvo, digamos $X(x_1, x_2, ..., x_n)$ como um conjunto de combinações dos predicados de entrada (Y, W, ..., Z) na forma de uma representação disjuntiva, do tipo soma de produtos.

O procedimento de treinamento encontra as combinações dos predicados de entrada que permitem inferir o predicado alvo, com um suporte líquido mínimo, a partir de exemplos positivos (objetos que satisfazem o predicado alvo) e exemplos negativos (objetos que não satisfazem o predicado alvo). No caso de predicados relacionais, os predicados de entrada podem tanto expressar propriedades de um objeto, como também relações entre objetos.

O algoritmo de treinamento é semelhante ao do MNC, porém a punição e a recompensa são feitas de forma diferente:

- No caso da recompensa, soma-se o valor correspondente ao número de predicados de entrada que foram combinados;

- Para a punição é atribuído o valor –1 para o nodo correspondente, tornando este neurônio inativo.

A punição do algoritmo de treinamento da FOLONET leva em conta o fato de os dados utilizados serem determinísticos. Com isso, quando é fornecido um exemplo negativo, todas as combinações dos predicados do conhecimento prévio que são satisfeitas se tornam inativas, não sendo utilizadas até o final do algoritmo de treinamento.

Este algoritmo é apresentado a seguir:

TABELA 4.1 - Algoritmo de Treinamento da *FOLONET*

Entrada:

- Arquitetura da Rede *FOLONET*;
- Conjunto de exemplos positivos e negativos;
- Conhecimento prévio: predicados que podem ser necessários na definição do predicado alvo.

Saída:

- Todas as cláusulas relativas aos arcos que satisfazem o suporte mínimo da rede.

Procedimento:

- Gerar todas as combinações possíveis dos predicados do conhecimento prévio de acordo com o número máximo de combinações definido pelo usuário.
- Para cada exemplo do conjunto de treinamento faça:
  - Marcar verdadeiro ou falso para cada nodo que representa um predicado fonte (ou uma combinação deles) quando o exemplo satisfaz ou não, respectivamente, o predicado (ou conjunto de predicados) em questão.
  - Propagar os sinais para os predicados que foram satisfeitos até o predicado alvo.
  - Para cada arco que chega ao predicado alvo, faça:
    - Se o exemplo é positivo, adicionar o valor correspondente ao número de predicados combinados ao acumulador do arco;
    - Se o exemplo é negativo, atribuir ao acumulador do arco o valor –1, tornando este caminho inativo.
- Exibir as regras correspondentes a todos os arcos que chegam com acumulador ≥ suporte mínimo da rede.

O resultado do treinamento pode ser representado como um conjunto de cláusulas envolvendo somas de produto.

É importante ressaltar que a *FOLONET* lida com o aprendizado incremental, ou seja, depois de se ter o resultado do aprendizado, novos exemplos podem ser fornecidos com o objetivo de melhorar as regras obtidas.

4.3 Implementação

O modelo da rede *FOLONET* foi implementado na linguagem *Delphi* 3.0 para a plataforma *Windows 98*. A seguir serão apresentadas as principais características de implementação de cada etapa do algoritmo de treinamento da rede, bem como algumas configurações necessárias para a execução do mesmo.

4.3.1 Entrada dos Dados

A entrada dos dados é feita através de um arquivo contendo o banco de dados da aplicação. Para a *FOLONET* é preciso que este banco de dados tenha somente uma tabela que contenha todas as informações relevantes para o aprendizado. Com isso, é preciso que o banco de dados relacional seja preparado para que possa ser submetido ao programa, sendo necessário realizar as junções específicas. O arquivo resultante deve conter colunas que futuramente representarão os predicados do conhecimento prévio.

A utilização de uma tupla universal tem sido apontada como um problema [RAE 98a], já que a tupla universal pode conter valores nulos. Para o modelo da *FOLONET* isto não é um problema, pois a existência de valores nulos, ou seja, a inexistência de um predicado específico, simplesmente significa que o nodo correspondente na rede não estará ativo. Além disso, no arquivo de treinamento os valores nulos não são representados, não sendo necessário espaço físico para armazená-los.

A figura a seguir apresenta a tela inicial do programa desenvolvido, na qual é determinado o caminho para o arquivo que contém o banco de dados do problema a ser tratado.

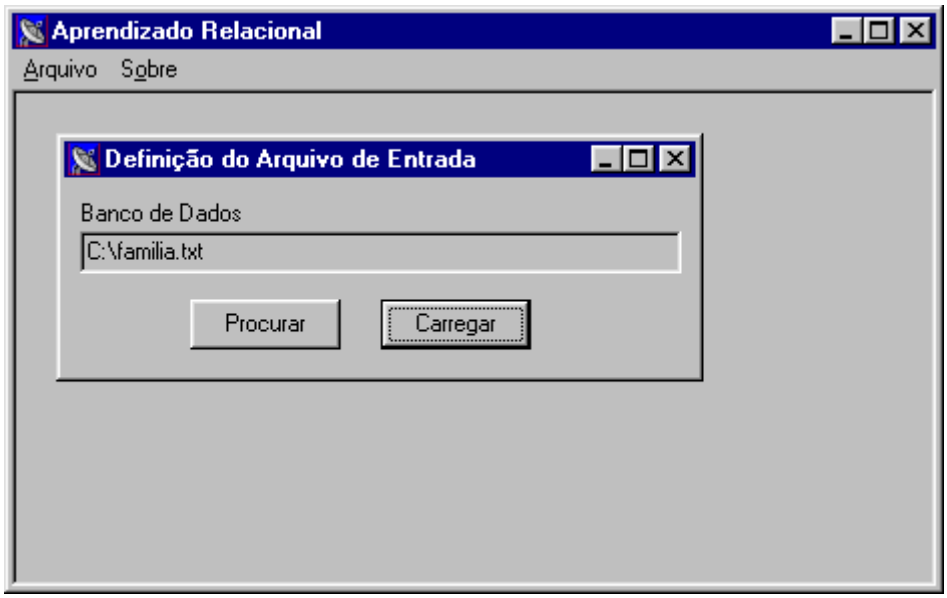


FIGURA 4.1 - Definição do arquivo de entrada para a *FOLONET*

Assim que o banco de dados é fornecido ao sistema, o usuário determina quais são os predicados que formarão o conhecimento prévio para o aprendizado. Isto é feito através de outra tela (figura 4.2) do programa da *FOLONET* que contém as principais configurações que devem ser feitas pelo usuário.

FIGURA 4.2 - Definição dos predicados que formam o conhecimento prévio

Além da definição dos predicados que formam o conhecimento prévio, o usuário deve indicar se há utilização de variáveis locais (ver Cap. 5, Seção 5.3), o número máximo de literais no corpo das regras finais obtidas e se há utilização ou não de predicados negados.

Ao definir quais predicados formam o conhecimento prévio, o usuário deve determinar quais as colunas correspondem a esses predicados, indicando quantos argumentos cada predicado possui.

Nesta primeira implementação, o número máximo de argumentos de cada predicado do conhecimento prévio é restrito ao número de argumentos dos exemplos do arquivo de treinamento, ou seja, ao número de variáveis envolvidas no predicado alvo e que estão presentes no arquivo de treinamento.

Além de definir o número de argumentos de cada predicado, o usuário deve indicar a quais características das tuplas do banco de dados os argumentos estão relacionados e também indicar se este predicado pode combinar variações dele mesmo (por exemplo, combinar uma variação com sua negação).

Esta etapa de determinação dos argumentos de cada predicado está ilustrada na figura 4.3 a seguir:

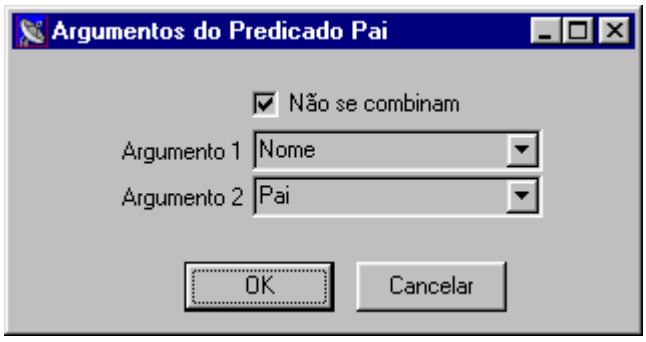


FIGURA 4.3 - Determinação dos argumentos do predicado Pai

4.3.2 Processamento dos Dados de Entrada

Depois da definição dos predicados que formam o conhecimento prévio para o aprendizado, é feita a geração da rede neural através das combinações destes predicados.

Para cada predicado definido pelo usuário, são gerados outros, a partir da permutação das variáveis envolvidas e de suas respectivas negações. Por exemplo, consideremos os predicado *pai*(*X*, *Y*) e *mulher*(*X*), definidos pelo usuário com 2 argumentos e 1 argumento, respectivamente, sendo que não combinam suas variações. O sistema irá gerar os seguintes predicados a partir dele:

TABELA 4.2 - Predicados gerados a partir de *pai*(*X*, *Y*) e *mulher*(*X*)

pai(*X*, *Y*)	*mulher*(*X*)
não pai(*X*, *Y*)	*não mulher*(*X*)
pai(*Y*, X)	*mulher*(*Y*)
não pai(*Y*, *X*)	*não mulher*(Y)

É importante observar que, como estão disponíveis duas variáveis para os predicados, são geradas as variações do predicado *mulher*(*X*) com a utilização destas variáveis, apesar deste predicado só ter um argumento. Os predicados relativos a mulher com o uso de variáveis distintas são tratados como se fossem predicados diferentes.

Concluída a geração de todos os predicados possíveis, é feita a combinação dos mesmos. Para isso, o sistema leva em conta se estes predicados combinam suas variações e o número máximo de cláusulas nas regras finais. No caso do exemplo acima, vamos supor que o usuário tenha optado por ter no máximo 3 cláusulas nas regras finais. Serão geradas, então, combinações de 1 até 3, como se pode observar na tabela 4.3.

TABELA 4.3 - Combinações geradas a partir dos predicados do conhecimento prévio

1 cláusula	*pai*(*X*, *Y*); *não pai*(*X*, *Y*); *pai*(*Y*, *X*); *não pai*(*Y*, *X*);
	mulher(*X*); *não mulher*(*X*); *mulher*(*Y*); *não mulher*(*Y*).
2 cláusulas	*pai*(*X*, *Y*) e *mulher*(*X*); *pai*(*X*, *Y*) e *não mulher*(*X*);
	pai(*X*, *Y*) e *mulher*(*Y*); *pai*(*X*, *Y*) e *não mulher*(*Y*);
	não pai(*X*, *Y*) e *mulher*(*X*); *não pai*(*X*, *Y*) e *não mulher*(*X*);
	não pai(*X*, *Y*) e *mulher*(*Y*); *não pai*(*X*, *Y*) e *não mulher*(*Y*);
	pai(*Y*, *X*) e *mulher*(*X*); *pai*(*Y*, *X*) e *não mulher*(*X*);
	pai(*Y, X*) *e mulher*(*Y*)*; pai*(*Y, X*) *e não mulher*(*Y*)*;*
	não pai(*Y*, *X*) e *mulher*(*X*); *não pai*(*Y*, *X*) e *não mulher*(*X*);
	não pai(*Y*, *X*) e *mulher*(*Y*); *não pai*(*Y*, *X*) e *não mulher*(*Y*)
	mulher(*X*) e *mulher*(*Y*); *mulher*(*X*) e *não mulher*(*Y*);
	não mulher(*X*) e *mulher*(*Y*); *não mulher*(*X*) e *não mulher*(*Y*);
3 cláusulas	*pai*(*X*, *Y*) e *mulher*(*X*) e *mulher*(*Y*); *pai*(*X*, *Y*) e *não mulher*(*X*) e *mulher*(*Y*);
	pai(*X*, *Y*) e *mulher*(*X*) e *não mulher*(*Y*); *pai*(*X*, *Y*) e *não mulher*(*X*) e *não mulher*(*Y*);
	não pai(*X*, *Y*) e *mulher*(*X*) e *mulher*(*Y*); *não pai*(*X*, *Y*) e *não mulher*(*X*) e *mulher*(*Y*);
	não pai(*X*, *Y*) e *mulher*(*X*) e *não mulher*(*Y*);
	não pai(*X*, *Y*) e *não mulher*(*X*) e *não mulher*(*Y*);
	pai(*Y*, *X*) e *mulher*(*X*) e *mulher*(*Y*); *pai*(*Y, X*) *e não mulher*(*X*) *e mulher*(*Y*)*;*
	pai(*Y, X*) *e mulher*(*X*) *e não mulher*(*Y*)*; pai*(*Y, X*) *e não mulher*(*X*) *e não mulher*(*Y*)*;*
	não pai(*Y*, *X*) e *mulher*(*X*) e *mulher*(*Y*); *não pai*(*Y*, *X*) e não *mulher*(*X*) e *mulher*(*Y*);
	não pai(*Y*, *X*) e *mulher*(*X*) e *não mulher*(*Y*);
	não pai(*Y*, *X*) e *não mulher*(*X*) e *não mulher*(*Y*);

Pode-se observar que as combinações são feitas entre os predicados relativos a *pai*(*X*, *Y*) e *mulher*(*X*) e *mulher*(*Y*), sendo que numa combinação nunca aparece mais de um predicado relativo a *pai*(*X*, *Y*). Há combinações envolvendo os predicados relativos à *mulher*(*X*) e à *mulher*(*Y*), mas não combinações entre os predicados relativos só à *mulher*(*X*) (por exemplo, não é permitida a combinação *mulher*(*X*) e *não mulher*(*X*)).

4.3.3 Dados de Saída

Com a rede neural construída para o problema, basta realizar o treinamento para a obtenção das hipóteses finais.

Para a execução do algoritmo de treinamento é preciso que o usuário especifique o arquivo de treinamento. Este arquivo contém tuplas ordenadas que satisfazem ou não o predicado alvo, ou seja, os exemplos positivos e negativos, respectivamente.

Assim que o arquivo de treinamento é fornecido ao sistema, o algoritmo de treinamento é executado, fornecendo as regras obtidas a partir dele.

Durante a execução do algoritmo de treinamento é feita uma operação de *match* de *tokens* na camada de entrada. Cada argumento dos predicados que são avaliados corresponde a um campo das tuplas ordenadas do arquivo de treinamento e, quando essas tuplas satisfazem os predicados, esses campos devem ter o mesmo conteúdo. Então, para verificar se determinadas tuplas satisfazem um predicado, é necessário realizar um *match* de *tokens* nos campos determinados pelos argumentos do predicado.

O resultado do algoritmo de treinamento são regras que cobrem todos os exemplos positivos e que não cobrem exemplos negativos. Em função disto, o arquivo de treinamento é muito importante para o resultado final, influenciando bastante na formação das hipóteses. Por este motivo, é importante que seja fornecido por um especialista que entenda bem o problema a ser tratado e que possa conseguir exemplos significativos de casos positivos e negativos, no caso de problemas complexos.

5 Resultados

Neste capítulo são apresentados os principais resultados da implementação da *FOLONET*. Foram feitos diversos testes para avaliação e obtenção dos resultados. Estes testes incluem o aprendizado atributo-valor e o aprendizado relacional, objetivo principal deste trabalho. As seções seguintes apresentam os mesmos.

5.1 Representação de Predicados Monádicos

Para o teste de predicados monádicos, aqueles com apenas um argumento, consideramos uma aplicação bancária em que se deseja representar o predicado alvo c*liente preferencial*, ou *Cliente_Preferencial*(*X*) como uma soma de produtos envolvendo os predicados monádicos: *tem cartão de crédito*, ou *Cartao_Credito*(*X*), *tem seguro*, ou *Seguro*(*X*) e *tem cheque especial*, ou *Cheque_Especial*(*X*).

Para esta aplicação, o conjunto de treinamento que corresponde a uma amostra do banco de dados é representado por clientes classificados como clientes preferenciais ou não. A tabela abaixo apresenta este conjunto de treinamento.

TABELA 5.1 - Banco de dados para aplicação bancária

Nome	Cartão de Crédito	Seguro	Cheque Especial	Cliente Preferencial
Carlos	Sim	Não	Sim	Sim
Maria	Não	Sim	Sim	Sim
João	Sim	Sim	Sim	Sim
Pedro	Não	Sim	Não	Não
Tânia	Sim	Não	Não	Não
Luís	Sim	Sim	Não	Não
Sílvia	Não	Não	Sim	Não

A rede neural gerada para este exemplo é apresentada na figura 5.1 e contém todas as combinações possíveis dos predicados de entrada e que são candidatas a formar as regras que descrevem o predicado alvo. Neste exemplo estamos desconsiderando a presença de predicados negados para a descrição do predicado alvo, pois para esta aplicação não faz muito sentido.

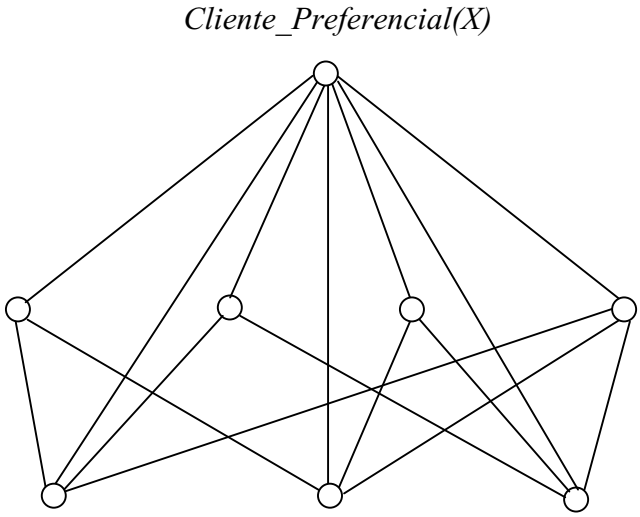


FIGURA 5.1 - Rede *FOLONET* para aplicação bancária

O conjunto de treinamento foi composto por todos os clientes do banco de dados e todos os exemplos foram separados de acordo com a sua classificação em relação ao campo de cliente preferencial.

A tabela abaixo apresenta a evolução dos acumuladores dos arcos diretos entre os predicados, após a apresentação dos exemplos do arquivo de treinamento.

TABELA 5.2 - Acumuladores para a aplicação bancária

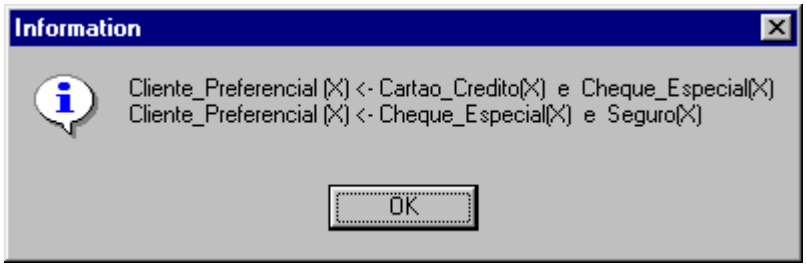
Cred	Seg	Cheq	Carlos	Maria	João	Pedro	Tânia	Luís	Sílvia
X			1	1	2	2	-1	-1	-1
	X		0	1	2	-1	-1	-1	-1
		X	1	2	3	3	3	3	-1
X	X		0	0	2	2	2	-1	-1
X		X	2	2	4	4	4	4	4
	X	X	0	2	4	4	4	4	4
X	X	X	0	0	3	3	3	3	3

A seguir é realizada a poda com suporte menor ou igual a 4. Da tabela, podemos extrair as seguintes regras:

$$Cliente\_Preferencial(X) \leftarrow Cartao\_Credito(X) \wedge Cheque\_Especial(X)$$

$$Cliente\_Preferencial(X) \leftarrow Cheque\_Especial(X) \wedge Seguro(X)$$

O resultado da implementação da *FOLONET* é o seguinte:



5.2 Aprendizado Relacional

Os testes realizados para aprendizado atributo-valor mostram que a *FOLONET* comporta-se bem para este tipo de aprendizado. Porém, a sua principal funcionalidade é o aprendizado relacional, onde os predicados podem tanto expressar propriedades de um objeto quanto relações entre eles.

Para a realização dos testes foi utilizada uma base de dados contendo a descrição das relações de parentesco de duas famílias, conforme a figura abaixo.

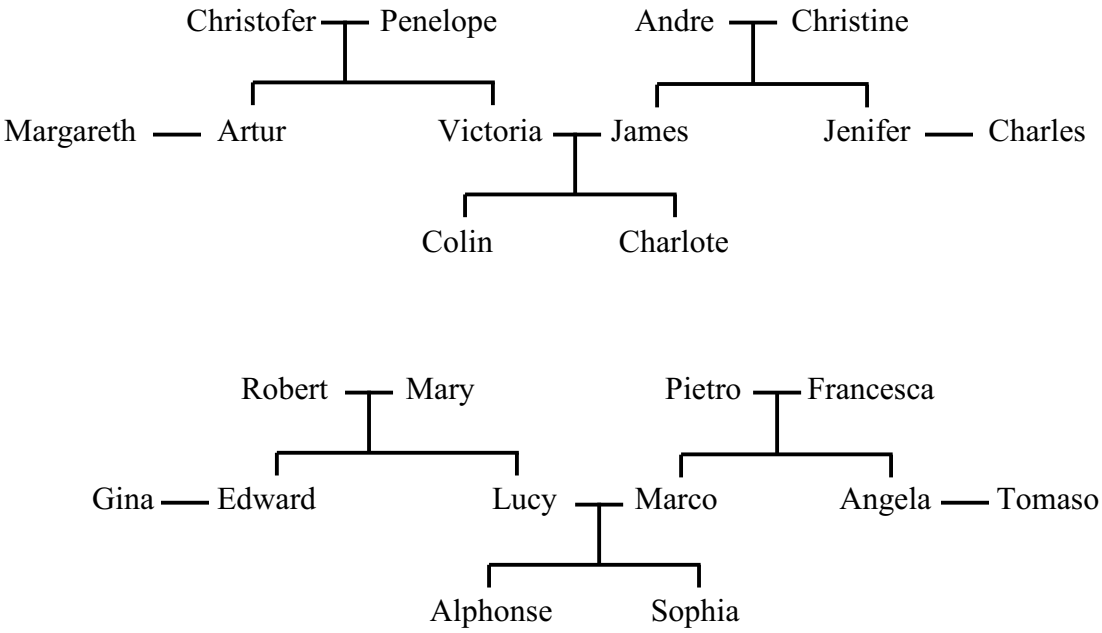


FIGURA 5.2 - Relações de Parentesco

A partir destas relações de parentesco, foi criada uma tabela com as principais informações necessárias para a obtenção do conhecimento prévio e do conjunto de treinamento. Ela pode ser vista como uma amostra de um banco de dados contendo estas relações e será o arquivo de entrada da *FOLONET*. Além deste arquivo, cada teste tem o seu arquivo de treinamento que é solicitado no momento do treinamento da rede. A tabela 5.3 a seguir apresenta a amostra do banco de dados gerada a partir das relações de parentesco.

TABELA 5.3 – Amostra de um banco de dados para relações de parentesco

Identificador	Nome	*Pai*	*Filho*	*Filha*	*Esposa*	*Mulher*
1	Christofer		Arthur	Victoria	Penelope	False
2	Penelope		Arthur	Victoria		True
3	Arthur	Christofer			Margareth	False
4	Margareth					True
5	Victoria	Christofer	Colin	Charlote		True
6	James	Andre	Colin	Charlote	Victoria	False
7	Andre		James	Jenifer	Christine	False
8	Christine		James	Jenifer		True
9	Jenifer	Andre				True
10	Charles				Jenifer	False
11	Colin	James				False
12	Charlote	James				True
13	Robert		Edward	Lucy	Mary	False
14	Mary		Edward	Lucy		True
15	Edward	Robert			Gina	False
16	Gina					True
17	Lucy	Robert	Alphonse	Sophia		True
18	Marco	Pietro	Alphonse	Sophia	Lucy	False
19	Pietro		Marco	Angela	Francesca	False
20	Francesca		Marco	Angela		True
21	Angela	Pietro				True
22	Tomaso				Ângela	False
23	Alphonse	Marco				False
24	Sophia	Marco				True

Neste exemplo, os predicados que poderão formar o conhecimento prévio são os seguintes: *X é pai de Y*, ou *pai*(*X*, *Y*), *X é filho de Y*, ou *filho*(*X*, *Y*), *X é filha de Y*, ou *filha*(*X*, *Y*), *X é esposa de Y*, ou *esposa*(*X*, *Y*) e *X é mulher*, ou *mulher*(*X*).

A fim de tratar os problemas manualmente, mostraremos uma versão simplificada no Teste 1, que consiste em descrever o predicado alvo *X é mãe de Y*, ou *mãe*(*X*, *Y*) a partir dos predicados *pai*(*X*, *Y*) e *filha*(*X*, *Y*).

Os demais testes foram feitos variando-se alguns parâmetros, como o número máximo de cláusulas nas regras obtidas, a inclusão de outros predicados no conhecimento prévio e alterações no arquivo de treinamento. Nestes testes, somente o resultado final será mostrado, não sendo apresentados os resultados intermediários.

Teste 1

Predicado alvo: *mãe*(*X*, *Y*).

Conhecimento Prévio: predicados *pai*(*X*, *Y*), *filha*(*X*, *Y*). Não combinam suas variações

Número máximo de literais no corpo das regras finais: 02

A partir dos predicados que formam o conhecimento prévio foi construída a rede neural que é apresentada na figura 5.3. Nesta figura não foram representados, para simplificação, todos os neurônios gerados nem seus arcos correspondentes.

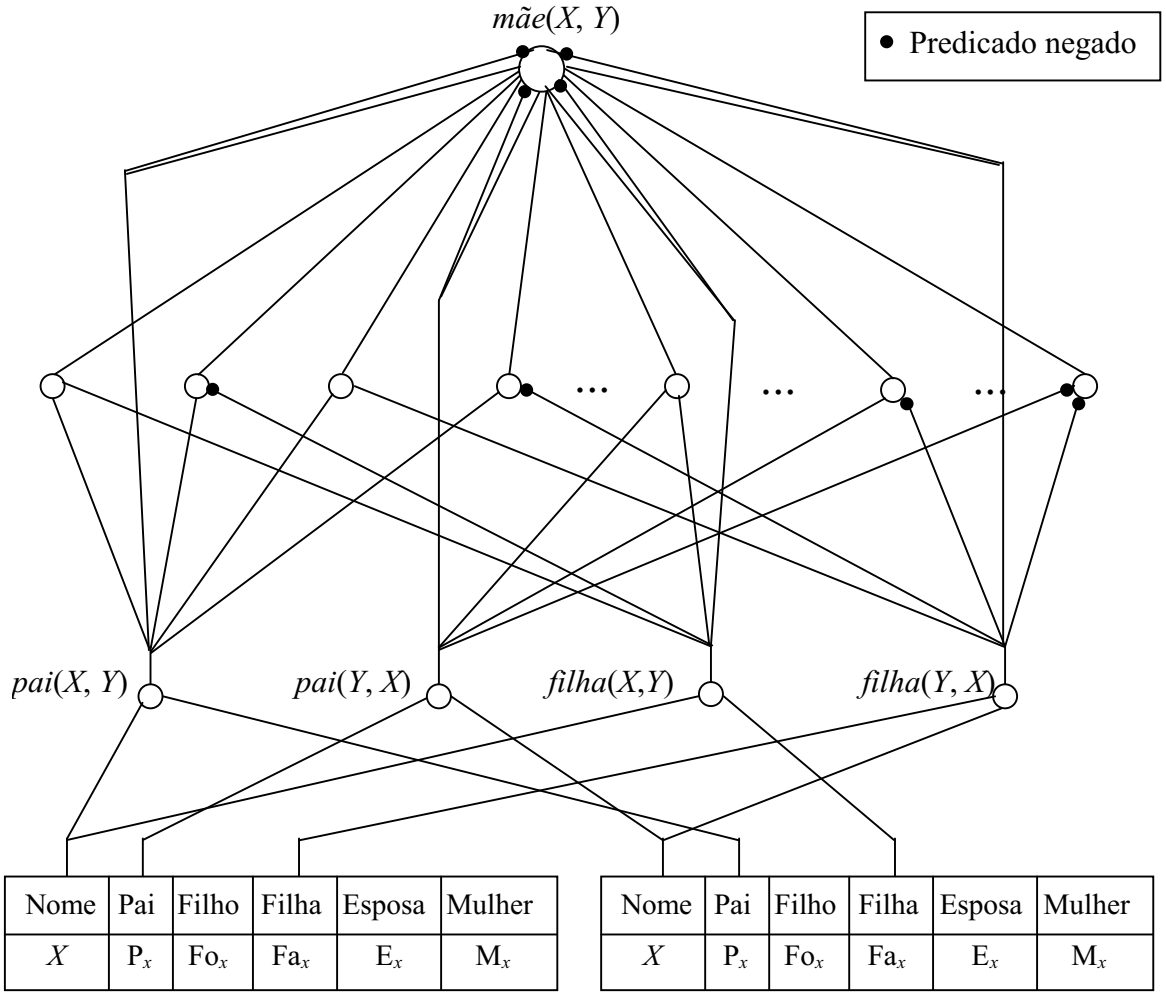


FIGURA 5.3 - Rede Neural obtida a partir do Teste 1

Com a rede neural estabelecida é necessário a execução do algoritmo de treinamento. O conjunto de treinamento é apresentado na tabela 5.4.

TABELA 5.4 – Conjunto de treinamento para o Teste 1

Exemplos Positivos	*Mãe*(Penelope, Victoria)
	Mãe(Victoria, Charlote)
	Mãe(Christine, Jenifer)
	Mãe(Mary, Lucy)
Exemplos Negativos	*Mãe*(Penelope, Charlote)
	Mãe(Christofer, Victoria)

Estes exemplos do conjunto de treinamento correspondem aos seguintes pares ordenados na amostra do banco de dados representada pela Tabela 5.3:

$E^+ = \{(2, 5), (5, 12), (8, 9), (14, 17)\}$ e $E^- = \{(2, 12), (1, 5)\}$.

A tabela abaixo mostra como os valores dos acumuladores para os predicados (ou combinações de predicados) são afetados pela submissão de cada exemplo à *FOLONET* . A última coluna corresponde aos valores de suporte da rede depois de todos os exemplos terem sido apresentados.

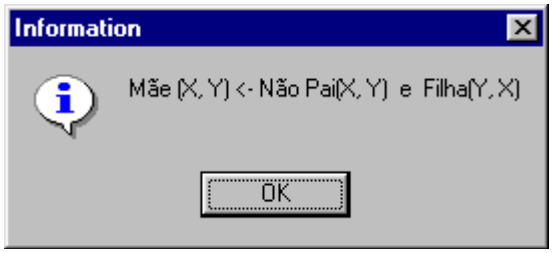
TABELA 5.5 - Resultado da submissão dos exemplos à *FOLONET*

p(X,Y)	*¬p(X,Y)*	*p(Y,X)*	*¬p(Y,X)*	*f(X,Y)*	*¬f(X,Y)*	*f(Y,X)*	*¬f(Y,X)*	(2,5)	(5,12)	(8,9)	(14,17)	(2,12)	(1,5)
X								0	0	0	0	0	-1
	X							1	2	3	4	-1	-1
		X						0	0	0	0	0	0
			X					1	2	3	4	-1	-1
				X				0	0	0	0	0	0
					X			1	2	3	4	-1	-1
						X		1	2	3	4	4	-1
							X	0	0	0	0	-1	-1
X				X				0	0	0	0	0	0
X					X			0	0	0	0	0	-1
X						X		0	0	0	0	0	-1
X							X	0	0	0	0	0	0
	X			X				0	0	0	0	0	0
	X				X			2	4	6	8	-1	-1
	X					X		2	4	6	8	8	8
	X						X	0	0	0	0	-1	-1
		X		X				0	0	0	0	0	0
		X			X			0	0	0	0	0	0
		X				X		0	0	0	0	0	0
		X					X	0	0	0	0	0	0
			X	X				0	0	0	0	0	0
			X		X			2	4	6	8	-1	-1
			X			X		2	4	6	8	8	-1
			X				X	0	0	0	0	-1	-1

Depois de concluído o treinamento da rede é feita a poda com um limiar. Na primeira implementação da *FOLONET* este limiar é determinado pelo maior valor obtido nos acumuladores, que neste exemplo é 8. Com isso, obtemos a seguinte regra:

$$Mãe(X, Y) \leftarrow \neg Pai(X,Y) \wedge Filha(Y, X)$$

O programa que implementa a *FOLONET* gera a seguinte saída:



Nos testes a seguir apresentamos as configurações iniciais, ou seja, quais predicados formam o conhecimento prévio, qual o número máximo de cláusulas nas regras de saída e qual é conjunto de treinamento. Não serão apresentados os passos de treinamento e poda. Será apresentado o resultado da execução do programa que implementa a rede *FOLONET*.

Teste 2

Conhecimento Prévio: predicados *pai*(*X*, *Y*), *filha*(*X*, *Y*) e *mulher*(*X*).

Número máximo de literais no corpo das regras finais: 02

Conjunto de Treinamento: Tabela 5.4

Resultado Final:

Teste 3

Conhecimento Prévio: predicados *pai*(*X*, *Y*), *filha*(*X*, *Y*) e *mulher*(*X*).

Número máximo de literais no corpo das regras finais: 03

Conjunto de Treinamento: Tabela 5.4

Resultado Final:

Teste 4

Conhecimento Prévio: predicados *pai*(*X*, *Y*), *filha*(*X*, *Y*) e *mulher*(*X*).

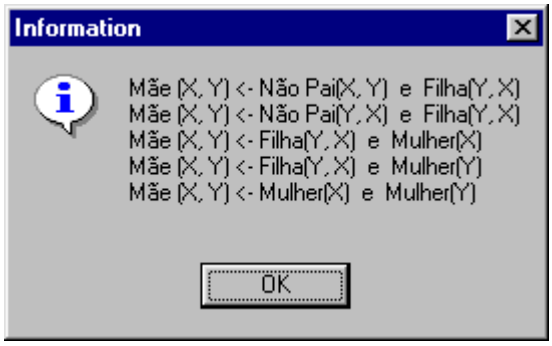
Número máximo de literais no corpo das regras finais: 02

Conjunto de Treinamento: fornecido na tabela 5.6

TABELA 5.6 - Conjunto de treinamento para o Teste 4

Exemplos Positivos	*Mãe*(Penelope, Victoria)
	Mãe(Victoria, Charlote)
	Mãe(Christine, Jenifer)
	Mãe(Mary, Lucy)
	Mãe(Lucy, Sophia)
	Mãe(Francesca, Angela)
Exemplos Negativos	*Mãe*(Christofer, Charlote)
	Mãe(Christofer, Colin)
	Mãe(Victoria, Artur)
	Mãe(Victoria, James)

Resultado Final:



Teste 5

Conhecimento Prévio: predicados *pai*(*X*, *Y*), *filha*(*X*, *Y*) e *mulher*(*X*).

Número máximo de literais no corpo das regras finais: 03

Conjunto de Treinamento: Tabela 5.6

Resultado Final:

Com estes testes concluiu-se que o resultado final depende do número máximo de literais permitidos que o usuário fornece e, principalmente, dos exemplos contidos no conjunto de treinamento. Os exemplos positivos reforçam algumas regras e os negativos descartam outras.

A influência do arquivo de treinamento é muito grande no resultado final obtido. Por esta razão sua definição deve ser feita por um especialista e com muito cuidado no caso de problemas complexos.

5.3 Utilização de Variáveis Locais

Com a rede *FOLONET* consegue-se realizar aprendizado relacional, obtendo assim, uma expressividade de lógica de primeira ordem.

A implementação desta rede ainda possui algumas limitações e uma das mais importantes é a utilização de apenas uma variável local. Segundo Flach e Lavrac [FLA 2000], variáveis locais são aquelas que ocorrem somente no corpo da regra, enquanto que variáveis globais são aquelas que ocorrem na cabeça da regra. No contexto da *FOLONET*, as variáveis globais são aquelas que estão presentes no predicado alvo e as variáveis locais não aparecem no mesmo.

O uso das variáveis locais é importante, visto que várias relações não são obtidas só com a utilização de variáveis globais. Um exemplo que podemos obter das relações de parentesco é a relação *avô*. Esta relação envolve uma pessoa que não está presente diretamente na relação *avô*, como podemos observar a seguir:

$avô(X, Y) \leftarrow pai(X, Z),\ pai(Z, Y)$

ou

$avô(X, Y) \leftarrow pai(X, Z),\ mãe(Z, Y)$

Nas cláusulas acima, há a inclusão da variável *Z*, que representa uma outra pessoa que relaciona as outras duas presentes no predicado alvo.

A implementação inicial do modelo da rede *FOLONET* inclui a utilização de apenas uma variável local. A seguir apresenta-se uma possível solução para a inclusão deste tipo de variável na implementação da rede *FOLONET*, utilizando a tabela 5.3 (amostra de um banco de dados contendo as relações de parentesco) como base para os exemplos. Depois será apresentado o resultado de um teste realizado com o predicado *avô*(*X*, *Y*), que utiliza apenas uma variável local.

5.3.1 Solução Proposta

A variável local é uma variável que representa algum objeto que se relaciona com pelo menos dois objetos que estão ligados através do predicado alvo (que são representados por variáveis globais). Não faria sentido a existência de uma variável local que se relacionasse com apenas um objeto do predicado alvo. O que pode ocorrer é uma forma de transitividade, onde uma variável local se relaciona com um objeto do predicado alvo e outra variável local e, esta, por sua vez, se relaciona com outro objeto do predicado alvo, e assim, sucessivamente.

Por exemplo, na relação *bisavô*(*X*, *Y*) uma das possíveis definições seria:

$$bisavô(X, Y) \leftarrow pai(X, W),\ pai(W, Z), pai(Z, Y)$$

onde *W* (variável local) se relaciona com *X* (global) e *Z*, que é outra variável local que se relaciona com *Y* (global).

Levando-se em conta esta forma de definição, apresenta-se uma proposta para o uso de variáveis locais na rede *FOLONET*. A idéia é utilizar o mesmo algoritmo da rede *FOLONET* e estendê-lo para suportar variáveis locais. Com o objetivo de facilitar o entendimento da proposta, serão utilizadas como exemplo as definições das relações de parentesco *avô*(*X*, *Y*) e *bisavô*(*X*, *Y*) a partir do conhecimento prévio formada pelo predicado *pai*(*X*, *Y*).

$$avô(X, Y) \leftarrow pai(X, Z),\ pai(Z, Y)$$

$$bisavô(X, Y)\ ) \leftarrow pai(X, W), pai(W, Z), pai(Z, Y)$$

O uso das variáveis locais é necessário quando não se encontram predicados do conhecimento prévio que são satisfeitos com os objetos dos exemplos positivos do arquivo de treinamento. No caso do predicado alvo ser *avô*(*X*, *Y*) e supondo que temos no arquivo de treinamento o exemplo positivo *avô*(Andre, Colin), não conseguimos definir o predicado alvo a partir de *pai*(*X*, *Y*) (predicado do conhecimento prévio) porque as pessoas Andre e Colin não se relacionam diretamente pelo predicado pai.

Desta forma, não se consegue encontrar uma definição do predicado alvo a partir dos predicados do conhecimento prévio. O problema está no fato de que estes objetos que satisfazem o predicado alvo podem estar relacionados com outros objetos que não aparecem no conjunto de treinamento. No exemplo acima, a pessoa que se relaciona com Andre e Colin através do predicado pai é James: *pai*(Andre, James) e *pai*(James, Colin). Porém, James não se encontra no arquivo de treinamento para o exemplo positivo (Andre, Colin).

O que se pode fazer é tentar encontrar no banco de dados objetos que se relacionam com os exemplos positivos do arquivo de treinamento, quando estes não são suficientes para encontrar uma definição do predicado alvo. Em um primeiro passo, a busca seria de um objeto que se relacione diretamente com os objetos do exemplo positivo, ou seja, de um objeto que satisfaça algum dos predicados do conhecimento prévio com cada um dos objetos do predicado alvo. No exemplo de Andre e Colin, o objetivo seria encontrar alguém que se relacione com Andre pelo predicado *pai*(*X*, *Y*) e com Colin, também pelo predicado *pai*(*X*, *Y*).

O resultado da busca pode ser positivo, como no exemplo acima, onde se encontra James, que é filho de Andre e pai de Colin, mas também pode ser que não se encontre tal objeto. Neste caso, é necessária uma nova busca por variáveis locais, mas isto não foi implementado na *FOLONET*.

Quando ocorre falha na busca, deve-se tentar encontrar outro objeto que se relaciona com um dos objetos do exemplo positivo e com um objeto que se tenha encontrado anteriormente (representado por uma variável local) e que se relaciona com o outro objeto do exemplo positivo.

Um exemplo em que acontece isso é a relação *bisavô*(*X*, *Y*). Utilizando como exemplo positivo *bisavô*(Alphonse, Colin), temos que na primeira busca não se encontra uma pessoa que se relacione ao mesmo tempo com Alphonse e Colin pelo predicado *pai*(*X*, *Y*). As pessoas encontradas nesse passo são Andre e James, onde Andre se relaciona com Alphonse - *pai*(Alphonse, Andre) - e James se relaciona com Colin - *pai*(James, Colin).

O novo passo leva em consideração estas pessoas (Andre e James) na procura de uma nova pessoa para tentar se encontrar a definição do predicado *bisavô*(*X*, *Y*). Este passo consiste em encontrar alguém que se relacione com Andre e Colin, ou seja, encontrar alguém que se relacione com a pessoa obtida no passo anterior (e que se relaciona com Alphonse) e que também se relacione com Colin, que é a outra pessoa do exemplo *bisavô*(Alphonse, Colin).

Já na primeira tentativa do segundo passo, encontra-se James, que se relaciona com Andre e Colin: *pai*(Andre, James) e *pai*(James, Colin). Assim temos:

bisavô(Alphonse, Colin) ← *pai*(Alphonse, Andre), *pai*(Andre, James), *pai*(James, Colin),

Ainda assim, pode ser que não se encontre um objeto que satisfaça os requisitos no segundo passo e o algoritmo continua, seguindo o mesmo raciocínio, até que se encontrem as variáveis locais desejáveis para a definição do predicado alvo ou até que se alcance um número limite de variáveis locais ou, ainda, até que não se encontre mais objetos que são representados por variáveis locais.

Uma observação importante deve ser feita: com a utilização desta proposta, não pode haver a negação dos predicados para a busca por variáveis locais. Isto se deve ao fato de que, ao procurar por um objeto representado por uma variável local, o algoritmo realizará a busca pelas tuplas do banco de dados que satisfazem um determinado predicado, e se há permissão de predicado negado, todas as tuplas do banco de dados que não satisfazem o predicado serão escolhidas como possíveis variáveis locais.

O uso de variáveis locais se dará somente com os exemplos positivos do arquivo de treinamento. Os exemplos negativos só servem como contra-exemplos dos predicados que não têm envolvimento com variáveis locais.

O principal problema desta proposta se encontra no fato de que para cada exemplo positivo que necessita a busca por uma variável local será feita uma consulta no banco de dados. Este é um processo de alto custo, além de retirar da rede o paralelismo característico. Conclui-se, então, que é uma proposta inviável para grandes quantidades de dados, sendo necessário o estudo de outro método que possa ser utilizado nestes casos.

A seção seguinte apresenta o resultado da execução do programa que implementa a *FOLONET* para o predicado *avô*(*X*, *Y*), que necessita de apenas uma variável local para a sua definição.

5.3.2 Resultado para uma Variável Local

O teste na rede *FOLONET* para o predicado *avô*(*X*, *Y*) foi feito com a utilização do seguinte conjunto de treinamento:

TABELA 5.7 – Conjunto de treinamento para o teste do predicado *avô*(*X, Y*)

Exemplos Positivos	*avô* (Andre, Colin)
	avô (Andre, Charlote)
	avô (Pietro, Alphonse)
	avô (Pietro, Sophia)
Exemplos Negativos	*avô* (Mary, Lucy)
	avô (Penelope, Victoria)
	avô (Andre, Arthur)

Na tentativa de se realizar o treinamento, verificou-se que havia a necessidade da utilização de variáveis locais. Com isso, o programa realizou a busca por uma variável local para os exemplos positivos, gerando um novo arquivo de treinamento, que pode ser visualizado na tabela 5.8:

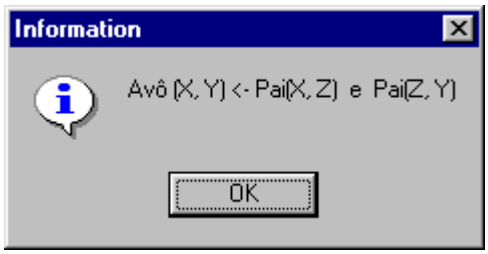
TABELA 5.8 - Novo conjunto de treinamento

Exemplos Positivos	*avô* (Andre, Colin) / James
	avô (Andre, Charlote) / James
	avô (Pietro, Alphonse) / Marco
	avô (Pietro, Sophia) / Marco

Onde James representa a variável local para os exemplos *avô* (Andre, Colin) e *avô* (Andre, Charlote) e Marco representa a variável local para os exemplos *avô* (Pietro, Alphonse) e *avô* (Pietro, Sophia).

Como pode ser visto, o novo arquivo de treinamento só apresenta exemplos positivos. Isso se deve ao fato de que, quando se utilizam variáveis locais, os exemplos negativos não são utilizados, como foi explicado anteriormente.

Com o arquivo de treinamento contendo as variáveis locais foi realizado um novo processo de aprendizagem, gerando o seguinte resultado:



Este resultado expressa a definição dada anteriormente para o predicado *avô*(*X*, *Y*) a partir do predicado *pai*(*X*, *Y*).

$$avô(X, Y) \leftarrow pai(X, Z),\ pai(Z, Y)$$

6 Conclusões e Trabalhos Futuros

Devido à grande necessidade de se tratar volumes crescentes de dados, surgiram as técnicas que compõem o processo da Descoberta de Conhecimento em Bancos de Dados. O seu processo completo envolve um elevado nível de subjetividade e de trabalho não totalmente automatizado. A fase de Mineração dos Dados é considerada a mais automatizada.

O foco central das aplicações de Mineração de Dados é a identificação de padrões interessantes e com suas descrições num modo conciso e significativo. Freqüentemente, para atingir este objetivo, devem ser aplicados métodos de aprendizado de máquina a fim de se construir modelos de dados por indução.

A Programação Lógica Indutiva é uma técnica importante para a descoberta de conhecimento em banco de dados relacionais, visto que pode descrever padrões envolvendo mais de uma relação. Esta técnica já mostrou seu potencial através de diversos sistemas de aprendizado. Além disso, a PLI tem demonstrado as vantagens de seu aparato de aprendizado em relação a outras abordagens, como por exemplo, aquelas baseadas em aprendizado proposicional. Os algoritmos de aprendizado de PLI apresentam alta expressividade, porém sofrem com a grande complexidade de seus processos, principalmente o teste de cobertura das variáveis.

Por outro lado, abordagens de aprendizado de máquina baseadas em Redes Neurais apresentam um bom desempenho em relação a grandes quantidades de dados, principalmente devido ao paralelismo inerente, porém têm sua expressividade limitada à da lógica proposicional [MAC 89], [TOW 94], [OSO 98].

Para contornar estes problemas, foi proposto um modelo de rede neural híbrida que combina a expressividade obtida pelos algoritmos de aprendizado da PLI com o desempenho de Redes Neurais: a *FOLONET – First Order Neural Network*.

A rede *FOLONET* mostrou sua capacidade de realizar aprendizado atributo-valor e, principalmente, o aprendizado relacional. As duas maiores vantagens da *FOLONET* em relação aos sistemas de aprendizado de PLI são:

- *FOLONET* permite o aprendizado incremental, evitando-se reiniciar o processo de aprendizagem quando novos exemplos são adicionados ao conjunto de treinamento, diferentemente de sistemas importantes de PLI, como os sistemas LINUS [LAV 94] e FOIL [QUI 90b];

- O tempo de execução do algoritmo de aprendizado pode ser bastante reduzido devido à utilização de uma arquitetura de Rede Neural, tirando vantagem de seu paralelismo natural. Em relação ao processo de cobertura utilizado por sistemas PLI, esta característica da *FOLONET* pode se tornar bastante interessante futuramente para utilização em grandes bancos de dados.

Por outro lado, pode-se argumentar que, devido à necessidade de se ter todos os predicados relacionados a um objeto em uma única tupla, a *FOLONET* tenha que realizar várias junções sobre as tabelas de entrada. Isto é verdade, já que se está realizando aprendizado relacional, mas esta característica também é apresentada por qualquer sistema de PLI ao realizar a cobertura de variáveis.

A utilização de uma tupla universal tem sido apontada como um problema [RAE 98a], já que a tupla universal pode conter valores nulos. Na verdade, isto não é um problema para o modelo da *FOLONET*. A existência de valores nulos, ou seja, a inexistência de um predicado específico simplesmente significa que o nodo correspondente na *FOLONET* não estará ativo. Em relação ao espaço no conjunto de treinamento, os valores nulos não requerem representação física.

Uma limitação presente neste primeiro modelo da *FOLONET* é a utilização de forma restrita de variáveis locais. Como foi visto, algumas relações necessitam a utilização destas variáveis, não sendo possível realizar o aprendizado na *FOLONET* quando for necessário mais de uma variável local. Foi apresentada uma proposta para o uso das variáveis locais, mas é um processo de alto custo para a utilização em grandes bancos de dados.

Outras propostas podem surgir em trabalhos futuros para as variáveis locais, como por exemplo, a utilização do conjunto de treinamento como fonte de busca dessas variáveis, ao invés do próprio banco de dados. Para isso, é necessário que o conjunto de treinamento tenha todas as tuplas relativas a um determinado objeto do banco de dados. Vale lembrar que um mesmo objeto pode ser representado por várias tuplas em uma relação universal.

Em relação ao uso da relação universal, pode-se investigar futuramente a utilização do próprio banco de dados relacional para o aprendizado relacional, sem a necessidade de sua conversão para uma única tabela.

Um fato importante que se avaliou foi a influência do arquivo de treinamento no resultado final da execução do algoritmo de treinamento. Neste arquivo devem estar exemplos significativos que possam refletir as principais características do problema a ser tratado. Por estas razões, é aconselhável que ele seja criado por um especialista no caso de problemas complexos.

A *FOLONET* mostrou ser um modelo bastante interessante para o aprendizado relacional. A partir deste modelo inicial, espera-se que se consiga utilizá-lo para grandes quantidades de dados, motivação pelo qual seu desenvolvimento foi proposto. Como trabalho futuro, pode-se realizar algumas otimizações em seu protótipo inicial e testes em grandes quantidades a fim de se fazer uma comparação com outras técnicas de Mineração de Dados.

Bibliografia

[ANA 98] ANAND, S.S.; HUGHES, J.G. Hybrid Data Mining Systems: The Next Generation. In: PACIFIC ASIA KNOWLEDGE DISCOVERY FROM DATABASES, PAKDD, 1998, Sidney. **Proceedings...** [S.l.:s.n], 1998.

[BEC 98] BECKENKAMP, B. G.; FELDENS, M. A.; PREE, W. Optimizations of the Combinatorial Neural Model. In: BRAZILIAN SIMPOSIUM ON NEURAL NETWORKS, 5., 1998, Belo Horizonte. **Proceedings...** Los Alamiros: IEEE Computer Society, 1998.

[BLO 2000] BLOCKEEL, H. et. al. **Scaling Up Inductive Logic Programming by Learning from Interpretations**. Belgium: Katholieke Universiteit Leuven, Department of Computer Science, 2000. (Technical Report CW 297).

[BRA 98] BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDERMIR, T. B. **Fundamentos de Redes Neurais Artificiais**. Rio de Janeiro: DCC/IM, COPPE/Sistemas, NCE/UFRJ, 1998. Trabalho apresentado na Escola de Computação, 11., 1998.

[CLA 89] CLARK, P.; NIBLETT T. The CN2 algorithm. **Machine Learning**, Dordrecht, v. 3, n. 4, p. 261-284, 1989.

[DOL 92] DOLSAK, B.; MUGGLETON, S. The application of Inductive Logic Programming to finite element mesh design. In MUGGLETON, S. (Ed.). **Inductive Logic Programming**. London: Academic Press, 1992. p.453-472.

[DZE 96] DZEROSKI, S. Inductive Logic Programming and Knowledge Discovery in Database. In: FAYYAD, U. et. al. (Eds.). **Advances in Knowledge Discovery and Data Mining**. Cambridge, Mass.: AAAI/MIT Press, 1996.

[FAY 96] FAYYAD, U. et al. From Data Mining to Knowledge Discovery: An Overview. In: FAYYAD, U. et. al. (Eds.). **Advances in Knowledge Discovery and Data Mining**. Cambridge, Mass.: AAAI/MIT Press, 1996.

[FEL 97] FELDENS, M. A. **Engenharia da Descoberta de Conhecimento em Bases de Dados**: Estudo e Aplicação na Área de Saúde. Porto Alegre: CPGCC da UFRGS, 1997. Dissertação de mestrado.

[FLA 2000] FLACH, P. A.; LAVRAC, N. The role of feature construction in inductive rule learning. In: WORKSHOP ON ATTRIBUTE-VALUE AND RELATIONAL LEARNING: CROSSING THE BOUNDARIES, ICML, 2000, Stanford. **Proceedings...** [S.l.: s.n], 2000.

[FRA 91] FRAWLEY, W.; PIATETSKY-SHAPIRO, G.; MATHEUS, C. Knowledge Discovery in Databases: An Overview. In: PIATETSKY-SHAPIRO, G.; FRAWLEY, W. (Eds.). **Knowledge Discovery in Databases**. Menlo Park, CA: The AAAI Press, 1991. p.1-27.

[JOH 97] JOHN, G. H. **Enhancements to the Data Mining Process**. Standford, EUA: Standford University, 1997. Ph.D. Thesis.

[LAE 2000] LAER, W. V; RAEDT, L. D. **How to Upgrade Propositional Learners to First Order Logic: a Case Study**. Belgium: Katholiek Universiteit Leuevn, Departament of Comupter Science, 2000. (Technical Report CW 288).

[LAV 94] LAVRAC, N.; DZEROSKI, S. **Inductive Logic Programming – Techniques and Applications**. England, UK: Elis Horwood, 1994.

[M AC 89] MACHADO, R. J.; ROCHA, A. F. **Handling Knowledge in High Order Neural Networks**: The Combinatorial Neural Network. Rio de Janeiro: IBM Rio Scientific Center, 1989. (Technical Report 076).

[MUG 90] MUGGLETON, S.; FENG, C. Efficient Induction of Logic Programs. In: CONFERENCE ON ALGORITHMIC LEARNING THEORY, 1., 1990, Tokyo. **Proceedings...** [S.l.:s.n.], 1990.

[MUG 91] MUGGLETON, S. Inductive logic programming. **New Generation Comupting**, [S.l.], v. 8, n. 4, p. 295-318, 1991.

[MUG 94] MUGGLETON, S.; RAEDT, L. D. Inductive logic programming : theory and methods. **Journal of Logic Programming**, [S.l.], v. 19, n. 20, p. 629-679, 1994.

[NIE 97] NIENHUYS-CHENG, S.; WOLF, R. **Foundations of Inductive Logic Programming**. Berlin: Spinger-Verlag, 1997. (Lectures Notes in Artificial Intelligence, 1228).

[OSO 98] OSORIO, F. S.; AMY, B. INSS: an hybrid system for constructive machine learning. In: INTERNATIONAL CONFERENCE ON NEURAL APPLICATIONS, NEURAP, 1998, Marseille. **Proceedings...** [S.l.:s.n.], 1998.

[PRA 98a] PRADO, H. **Abordagens Híbridas para Mineração de Dados**. Porto Alegre: CPGCC da UFRGS, 1998. (EQ-26).

[PRA 98b] PRADO, H. A. Do; FRIGERI, S. R.; ENGEL, P. M. A Parsimonious Generation of Combinatorial Neural Model. In: CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN, 4., Neuquém, 1998. **Proceedings ...** Neuquén: Universidad Nacional del Comahue, 1998. v.1, p. 427-436.

[PRA 99] PRADO, H. A. do et. al. Accuracy Tuning in Combinatorial Neural Model. In: PACIFIC-ASIA CONFERENCE ON KNOWLEDGE DISCOVERY FROM DATABASES AND DATA MINING, PAKDD, 1999, Beijing. **Proceedings ...** Berlin: Springer-Verlag, 1999. p. 247-251.

[QUI 86] QUINLAN, J. R. Inductin of Decision Trees. **Machine Learning**, Boston, v. 1, n. 1, p. 81-106, 1986.

[QUI 90a] QUINLAN, J. R. Probabilistic Decision Trees. In: KODRATOFF, Y.; MICHALSKI, R. (Eds.) **Machine Learning**: an Artificial Intelligence Approach. San Mateo, Calif.: Morgan Kaufmann, 1990. v. 3.

[QUI 90b] QUINLAN, J. R. Learning Logical Definitions from Relations. **Machine Learning**, Dordrecht, v. 5, p. 239-266, 1990.

[QUI 93] QUINLAN, J. R. **C4.5**: Programs for Machine Learning. San Mateo, Califórnia: Morgan Kaufmann, 1993.

[RAE 97] RAEDT, L. D.; DEHASPE, L. Clausal Discovery. **Machine Learning**, Dordrecht, v. 26, n. 2-3, p. 99-146, 1997.

[RAE 98a] RAEDT, L. D. Attribute-value Learning Versus Inductive Logic Programming: The Missing Links (Extended Abstract). In: INTERNATIONAL CONFERENCE ON INDUCTIVE LOGIC PROGRAMMING, 8., 1998, Madison, Wisconsin, USA. **Proceedings...** Madison: Springer-Verlag, 1998. *(*Lecture Notes in Artificial Intelligence, 1446).

[RAE 98b] RAEDT, L. D. et. al. Three companions for first order data mining. In DZEROSKI, S. and LAVRAC, Nada (Eds.). **Inductive Logic Programming for Knowledge Discovery in Databases**. Berlin: Springer-Verlag, 1998. Lecture Notes in Artificial Intelligence.

[SHA 94] SHAVLIK, J. W. A Framework for Combining Symbolic and Neural Learning. **Machine Learning**, Dordrecht, v. 14, p. 321-331, 1994.

[SHA 96] SHAVLIK, J. W. An Overview of Research at Wisconsin on Knowledge-Based Neural Networks. In: INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, Washington, 1996. **Proceedings...** Berlin: Springer-Verlag, 1996.

[TOW 94] TOWELL, G.; SHAVLIK, J. Knowledge-Based Neural Networks. **Artificial Intelligence**, Amsterdan, v. 70, n. 1-2, p. 119-165, 1994.

[ULL 97] ULLMAN, J. **A First Course in Database Systems**. New Jersey: Prentice-Hall, 1997.