Luiz Gustavo Moro Senko

Um Método Baseado em Lógica Paraconsistente para Detecção de Inconsistências em Classificadores à Base de Regras

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática – PPGIA, da Pontifícia Universidade Católica do Paraná, como requisito parcial para obtenção do título de Mestre em Informática.

Curitiba

2006

Luiz Gustavo Moro Senko

Um Método Baseado em Lógica Paraconsistente para Detecção de Inconsistências em Classificadores à Base de Regras

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática – PPGIA, da Pontifícia Universidade Católica do Paraná, como requisito parcial para obtenção do título de Mestre em Informática.

Área de Concentração:
Sistemas Inteligentes

Orientador: Prof. Dr. Fabrício Enembreck
Co-orientador: Prof. Dr. Bráulio C. Ávila

Curitiba
2006

Ao meu pai Luiz Carlos e a minha mãe Maria Rozeli,
pelo esforço realizado em meu benefício.

À minha irmã Susan e ao meu sobrinho Nícolas,
pelo amor e carinho sempre evidentes.

Aos amigos,
pelo estímulo e demonstração de amizade.

À Deus,
por permitir compreender o verdadeiro significado de se viver.

Agradecimentos

Ao Prof. Fabrício Enembreck pela amizade, profissionalismo e dedicação ao conduzir o desenvolvimento deste projeto.

À Prof[a]. Simone Nasser M. Ferreira, pelas idéias e sugestões que contribuíram para a realização deste projeto.

À Prof[a]. Andreia Malucelli, pelo carinho, amizade e estímulo em minha profissão e por muito auxiliar em grandes momentos de minha vida.

Ao Prof. Alessandro L. Koerich, por fazer acreditar mais em mim e em meus objetivos.

Aos amigos Márcio, Richardson, Heverson, Neander, Leonardo, Silla, Jaime, André, Rafael, Edson, Andréia, Daniele e Helyane pelo auxílio, compreensão e amizade indispensáveis para estimular a concretização deste objetivo pessoal.

A PUCPR pela oportunidade concedida.

E a todos os demais amigos, que de alguma forma, contribuíram para que fosse possível concluir este objetivo.

Senko, Luiz Gustavo Moro

S477m 2006 Um método baseado em lógica paraconsistente para detecção de inconsistências em classificadores à base de regras / Luiz Gustavo Moro Senko ; orientador, Fabrício Enembreck ; co-orientador, Bráulio C. Ávila. – 2006.

vi, 114 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2006

Inclui bibliografia

1. Exploração de dados (Computação). 2. Inconsistência (Lógica). 3. Linguagem de programação lógica. I. Enembreck, Fabrício. II. Ávila, Bráulio Coelho. III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. IV. Título.

CDD 21. ed. – 005.74

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

APRENDIZAGEM DE MÁQUINA ..... AM
LÓGICA PARACONSISTENTE ..... LP
PROGRAMAÇÃO LÓGICA EVIDENCIAL ..... PLE
PROGRAMAÇÃO LÓGICA EVIDENCIAL PARACONSISTENTE ..... PrLEP
GRAU DE CONTRADIÇÃO ..... G_{ct}
GRAU DE CERTEZA ..... G_c

Resumo

Muitos métodos em mineração distribuída têm sido desenvolvidos com o objetivo de viabilizar a aplicação de técnicas de mineração em grandes volumes de dados. As pesquisas em mineração distribuída têm como interesse principal a otimização de algoritmos e técnicas que possibilitem a análise de dados fisicamente distribuídos, propondo soluções mais viáveis financeiramente e computacionalmente.

Entre os principais métodos está a mineração distribuída orientada a dados. Este método consiste na divisão dos dados em subconjuntos menores que são minerados individualmente e produzem diversos classificadores locais para, posteriormente, combiná-los em um único classificador. O maior problema com essa técnica é a existência de dados inconsistentes nos subconjuntos que podem comprometer o desempenho e formar classificadores com opiniões contraditórias.
A utilização de técnicas de gerenciamento de incerteza permite um raciocínio mais adequado nas situações em que os dados ou conceitos obtidos nem sempre estão completos ou consistentes.

Neste trabalho foram empregados conceitos de Lógica Paraconsistente no desenvolvimento de um método capaz de tomar decisões confiáveis a partir de um conjunto de classificadores à base de regras, mesmo quando opiniões contraditórias estiverem presentes.

Os experimentos foram aplicados a diferentes bases de dados e na maior parte dos casos, o método proposto produz uma melhor classificação do que cada classificador local. Nos resultados em que o método desenvolvido foi superior, estima-se que a razão do desempenho está associada às bases formadas por maior número de atributos, que podem gerar conjuntos de regras mais expressivos e, possivelmente, uma melhor representação das características das bases.

Palavras-chave: mineração distribuída de dados, lógica paraconsistente.

Abstract

Many methods in distributed data mining have been developed with the objective to make possible the application of approaches in very large databases. The researches in distributed data mining have as main interest in the optimize algorithms and approaches to make possible the analysis in data physically distributed. In result it will consider more viable solutions financially and computationally.

Data oriented are between the main methods of distributed data mining. This method consists in partition of the data in lesser data sets that are mined individually generating local classifiers in order to combine then later, in a global classifier. The existence of inconsistent data in subsets can compromise the performance and build classifiers with contradictory opinions. The use of approaches of uncertainty management allows a more adequate reasoning in the situations where the gotten data or concepts are not always complete or consistent.

In this work concepts of Paraconsistent Logic in development of a method capable to take reliable decisions from a set of classifiers based in rules had been used, even when contradictory opinions are present.

The experiments had been applied in different databases and the most part of cases, the method produce a better classification than local classifiers. The results in which the developed method was better the reason of the performance is associated with large number of attributes in databases. Those attributes can generate expressive rules sets and, possibly, a better representation of the characteristics of these databases.

Key-words: distributed data mining, paraconsistent logic.

Capítulo 1

Introdução

Na era da informação digital, um dos grandes desafios que enfrentam as organizações é como produzir conhecimento a partir dos dados armazenados.

O armazenamento de dados está presente em todas as esferas: desde leitores eletrônicos em supermercados em códigos de barra à difundida Internet. O mundo globalizado está cada vez mais dependente da extração de conhecimento ou informações úteis a partir de processos de mineração de dados. Padrões semelhantes extraídos dos dados fornecem informações sobre o perfil de compra de clientes, características no acesso de informações on-line que permitam a detecção de fraudes, identificar comportamento que possibilite a previsão de tempo entre outros. Essas informações caracterizam-se como conhecimento útil e indispensável para a geração de ferramentas de apoio à tomada de decisões nas corporações, com o objetivo de aperfeiçoar serviços, oferecer novos e em decorrência agregar maior receita.

Com o crescimento em bases de dados e a necessidade de utilização de bases distribuídas para armazenar uma quantidade de dados cada vez mais significativa, surge a necessidade de pesquisar novas técnicas para viabilizar o processo de mineração em bases de dados distribuídas.

Diversas questões têm sido abordadas em mineração distribuída, entre as quais destacam-se: (i) o problema de escalabilidade, que consiste em propor soluções de como tratar a dimensionalidade dos dados, (ii) o projeto de otimização dos algoritmos utilizados, com o propósito de acelerar o processo de mineração em grandes bases de dados e (iii) o tratamento de inconsistências na integração de modelos, uma vez que a aprendizagem é realizada separadamente em conjuntos de dados distintos e a união desses modelos pode resultar em regras contraditórias.

A existência de informações conflitantes pode estar relacionada a vários motivos, como por exemplo, à união de base de dados geradas a partir de fontes distribuídas, ruídos nos dados, a existência de atributos com valores faltantes, erro na coleta de dados, etc. Outra consideração importante é que duas ou mais instâncias podem ser ditas inconsistentes se:

• os valores dos atributos forem os mesmos, porém a classe prevista é diferente;

• os atributos possuem valores similares, porém existe contradição entre os valores de um atributo e a classe prevista é a mesma.

De acordo com Ferreira [Ferreira et al., 2004] mesmo que duas ou mais bases sejam homogêneas, pode ocorrer inconsistências quando os valores dos atributos tenham diferentes definições, incompatibilidades nos esquemas das bases locais e diferentes distribuições de probabilidades, pois os dados são armazenados de forma independente nas bases de dados, que são altamente descentralizadas.

Inconsistências também podem ocorrer em bases centralizadas devido à presença de ruído nos dados e de atributos com valores faltantes. Neste caso, algumas soluções podem ser aplicadas na etapa de pré-processamento dos dados, mas o custo computacional e a necessidade de intervenção humana podem tornar o processo lento, difícil e inviável.

A ocorrência de inconsistências em bases de dados reduz significativamente a performance dos algoritmos de aprendizagem de máquina. Em mineração de dados, a utilização de algoritmos de aprendizado de máquina nas tarefas de aquisição de conhecimento trabalha com a experimentação e a comparação de diversos algoritmos na tentativa de se descobrir um que melhor se adapte a um domínio específico, sem uma redução significativa de performance.

Algoritmos de aprendizagem de máquina têm sido desenvolvidos sob diferentes paradigmas de aprendizado: estatístico, conexionista, baseado em instâncias, genéticos e aprendizagem simbólica [Mitchell, 1997]. Em especial, sistemas de aprendizado simbólico são utilizados em situações em que os

conceitos aprendidos precisam ser interpretados por humanos. O conhecimento induzido, neste caso, geralmente é representado por árvores de decisão ou por conjuntos de regras de produção. Algoritmos de indução têm se mostrado extremamente competitivos, em algumas situações, se comparados a outros algoritmos, como por exemplo, redes neurais.

Em mineração distribuída, a união dos modelos obtidos pode resultar em um conhecimento global inconsistente, se as regras obtidas forem formadas pelo mesmo antecedente (condições), porém prever diferentes conseqüentes (alvo), ou se as regras obtidas possuírem o mesmo conseqüente, no entanto, existam contradições entre os valores para um mesmo antecedente.

Inconsistências surgem naturalmente no mundo real e podem ocorrer em vários contextos e a automatização de um raciocínio adequado requer o desenvolvimento de teorias formais apropriadas [da Costa et al., 2000].

Em Inteligência Artificial, a Lógica Paraconsistente pode servir de base para teorias inconsistentes e não-triviais. Esse formalismo permite interpretações mais adequadas onde existam contradições e ausência de informações. A utilização da Lógica Paraconsistente permite atribuir medidas de crença a hipóteses para as quais existem certas evidências de apoio favoráveis e medidas de descrença em que a evidência seja desfavorável à informação.

O objetivo deste trabalho foi desenvolver um método baseado em Lógica Paraconsistente, em que fosse possível a integração de classificadores simbólicos provavelmente inconsistentes gerados a partir da mineração de bases de dados distribuídas. A metodologia proposta permite que vários classificadores sejam integrados e unificados em um único classificador global Paraconsistente capaz de classificar novos exemplos, produzindo uma melhor taxa de acerto do que a média de acerto dos classificadores locais.

1.1. Detalhamento do Problema

As informações armazenadas nas bases de dados são consideradas precisas se for possível assegurar que representem fielmente as informações do

“mundo real”. Em sistemas de banco de dados, no modelo de dados relacional, a imprecisão pode ocorrer nos valores que os atributos podem assumir, por exemplo, nos valores possíveis de S*alário* na relação *receber (Empregado, Salário),* ou estar presente nas tuplas, como *designar (Empregado, Projeto)* pode ser considerado uma relação precisa, no entanto a tarefa a ser desempenhada pelo empregado no projeto é incerta [Motro, 1990].

Segundo Motro [Motro, 1994], o mundo real pode ser visto como certo e concreto, mas o conhecimento obtido, às vezes pode ser considerado como incerto. A utilização do termo incerto ou impreciso refere-se a algum elemento do modelo que não possa ser afirmado com completa confiança. Nesse sentido, é possível distinguir o conceito de incerteza em diferentes categorias:

• incerteza – não é possível determinar se uma afirmação no modelo é verdadeira ou falsa. Por exemplo, pode haver incerteza no fato contido em uma base de dados “A idade de João é 38 anos”.

• imprecisão – a informação disponível no modelo não é específica. Não há uma distinção dos valores que a informação pode assumir. Por exemplo:

i. “A idade de João está entre 37 e 43 anos”;

ii. “A idade de João é 37 ou 43 anos” – imprecisão disjunta

iii. “João não tem 37 anos” – imprecisão negativa

iv. “A idade de João é” - desconhecida ou incompleta

• vago – incluem elementos, predicados ou quantificadores que não são determinados por métricas. Por exemplo, “João está na meia idade”. Os fundamentos e a formalização particular desse conceito estão baseados em Lógica *Fuzzy,* descrito nas seções posteriores.

• inconsistência – o modelo contém duas ou mais afirmações que não podem ser verdadeiras simultaneamente. Por exemplo:

“João tem entre 37 e 43 anos “ e

“João tem 35 anos”

• ambigüidade – faltam alguns elementos semânticos principais no modelo e a ausência possibilita diversas interpretações. Por exemplo:

“João tem 37”. Não é possível afirmar se João tem 37 anos, 37 meses ou 37 dias.

Em mineração distribuída de dados não é difícil encontrar situações onde existem inconsistências explícitas. Por exemplo, considere uma técnica de combinação de classificadores que tem como objetivo unificar a decisão de diversos classificadores gerados a partir de bases de dados distribuídas. Assumindo que uma instância pode pertencer a apenas uma classe, pode haver opiniões contraditórias entre os classificadores (3 deles prevêem a classe “A”, 4 deles prevêem a classe “B” e os últimos 3 prevêem a classe “C”). Neste caso, uma técnica como votação simples seria de difícil aplicação e poderia gerar erros de classificação.

A motivação deste trabalho foi aplicar os conceitos de Lógica Paraconsistente, com o objetivo de fornecer um raciocínio mais adequado no tratamento de informações inconsistentes. Ao aplicar os formalismos de Lógica Paraconsistente foi possível gerar decisões mais confiáveis a partir de um conjunto de classificadores mesmo quando opiniões contraditórias forem identificadas.

Um dos desafios em mineração de dados distribuídos é que dois ou mais itens de dados armazenados em diferentes bases podem ser inconsistentes, no contexto da tarefa que a mineração pretende resolver. No cenário de uma base de dados que avalia conceder crédito a clientes, por exemplo, uma das tuplas armazenadas pode ter a seguinte informação:

<*sexo*= masculino, *salário*= alto, *empréstimo*= não, *crédito*= bom>

E em oposição a esta tupla, em outra base, a seguinte informação pode estar presente:

<*sexo*= masculino, *salário*= alto, *empréstimo*= não, *crédito*= ruim>

As duas tuplas são consideradas inconsistentes porque possuem os mesmos valores para os atributos, mas diferentes valores para o atributo-meta *crédito,* que define conceder crédito ao cliente.

Mesmo dados locais, que têm exatamente o mesmo esquema definido da base de dados, podem conter inconsistências, pois unidades de coleta independentes e descentralizadas podem coletar dados com distribuições de probabilidade diferentes. Além disso, um conjunto de dados pode conter ruídos e ser considerado ou não inconsistente.

O problema de dados faltantes ou não conclusivos também pode caracterizar uma causa direta de dados inconsistentes, pois os atributos disponíveis não são capazes de discriminar corretamente as classes.

Além do problema de inconsistência, outros problemas devem ser tratados em mineração distribuída, como por exemplo:

- locais distintos para armazenamento dos dados;
- capacidade limitada de armazenamento em cada processador;
- necessidade de alta performance no acesso aos dados;
- alto custo na transmissão dos dados;

A partir dos problemas abordados nesta seção, apresentamos na seção seguinte novas proposições, que têm como objetivo a solução de alguns desses problemas.

1.2. Objetivos

Nas questões de mineração distribuída, um dos principais pontos a serem abordados está relacionado ao aumento da precisão na aprendizagem. Provost [Provost & Kolluri, 1999] relata que alguns autores propõem evitar o crescimento de regras supérfluas geradas, porque segundo ele, o aumento do tamanho do conjunto de regras não implica em aumento na eficiência do processo de aprendizagem, além de que o refinamento de um grande conjunto de regras completo torna maior o custo computacional. Portanto, a descoberta de pequenos

conjuntos de regras a partir de dados particionados pode ser interessante, mas provavelmente esse particionamento pode gerar inconsistências.

Dados inconsistentes são considerados um problema desafiador em mineração de dados e requer o desenvolvimento de métodos capazes de fornecer uma solução adequada. Nossa proposta está fundamentada na análise de regras inconsistentes geradas por algoritmos de indução, e utilizará um método de gerenciamento de incertezas, baseado nos conceitos de lógica paraconsistente, como solução. As regras são constituídas pelo antecedente (parte "se") que contém as condições dos valores dos atributos e pelo conseqüente (parte "então") que contém a classe predita para o exemplo que satisfaz todas as condições do antecedente da regra. As regras, descobertas na etapa de treinamento, são portanto, regras do tipo "se, então", como nos exemplos a seguir:

Regra 1:
Se *sexo=* masculino e *salário=* alto e *empréstimo=* não
então *crédito=* bom

Regra 2:
Se *sexo=* masculino e *salário=* baixo e *empréstimo=* não
então *crédito=* bom

As regras acima são consideradas inconsistentes porque apesar do conseqüente das regras serem formados pela condição *crédito=* bom*,* no antecedente elas possuem duas condições opostas (atributo *salário).*

Este problema pode acontecer porque em mineração distribuída de dados o método utilizado é a divisão do conjunto inicial em subconjuntos menores, que são minerados individualmente (esta técnica é conhecida como mineração distribuída orientada à dados). Posteriormente, cada classificador gerado sobre os subconjuntos é combinado a fim de se obter um modelo único e consistente de dados. As inconsistências também ocorrem se duas ou mais regras cobrem um

conjunto semelhante de instâncias, mas prevêem classes distintas, conforme o trabalho desenvolvido por Ferreira [Ferreira et al., 2004].

O objetivo deste projeto foi utilizar os conceitos de Lógica Paraconsistente durante a combinação dos classificadores obtidos pelo processo de mineração distribuída orientada à dados. Os formalismos desenvolvidos em Lógica Paraconsistente permitiram a construção de um modelo de raciocínio mais adequado, capaz de mensurar a inconsistência entre as regras. Desta forma, na classificação dos exemplos foi possível eleger a regra mais adequada, de acordo com um critério de decisão estabelecido, entre as demais regras existentes no conjunto.

Este trabalho está organizado da seguinte forma: no Capítulo 2 são apresentadas a definição e formas de representação em Aprendizagem Simbólica; posteriormente no Capítulo 3 são apresentadas diversas técnicas utilizadas em Mineração Distribuída. No Capítulo 4, são apresentadas Técnicas de Gerenciamento de Incertezas, entre as quais destaca-se Lógica Paraconsistente, cujo formalismo tornou-se base para o desenvolvimento deste projeto. O detalhamento da metodologia proposta é apresentado no Capítulo 5. No Capítulo 6 são apresentados os experimentos realizados e a comparação dos resultados parciais e globais obtidos com o algoritmo de indução de regras RIPPER. As conclusões finais, as contribuições com este trabalho e possíveis extensões com trabalhos futuros são apresentados no Capítulo 7.

Capítulo 2

Aprendizagem Simbólica

Um sistema de aprendizagem é um programa de computador capaz de tomar decisões a partir de experiências obtidas através de soluções em problemas anteriores.

Os sistemas de aprendizagem simbólica têm como objetivo realizar a aprendizagem e construir representações simbólicas de um conceito a partir da análise de exemplos e contra-exemplos disponíveis. Árvores de decisão e regras de produção estão entre as representações simbólicas mais estudadas e podem ser consideradas muito eficientes se comparadas a outros métodos de aprendizagem, como por exemplo, redes neurais.

Sistemas de aprendizado simbólico são utilizados nas situações em que o modelo obtido assume uma forma compreensível. Os sistemas ID3 [Quinlan, 1986] e C4 [Quinlan, 1987] para indução de árvores de decisão tornaram-se importantes contribuições nas pesquisas em Inteligência Artificial. Outra abordagem foi desenvolvida a partir da transcrição de árvores de decisão para regras [Quinlan, 1987] e que em um posterior refinamento deu origem ao algoritmo C.4.5rules [Quinlan, 1993].

2.1. Indução de Regras

Indução é uma forma de inferência lógica que permite a utilização de premissas para obter conclusões genéricas a partir de exemplos particulares. A indução pode ser caracterizada como uma forma de raciocínio que parte de um conceito específico e o generaliza [Prati, 2006].

O resultado da aplicação de inferência indutiva não preserva, necessariamente, a verdade, mesmo que o conjunto de premissas utilizado na inferência seja verdadeiro. Por esse motivo, o resultado da inferência indutiva é

geralmente chamado de hipótese, porque mesmo que não seja possível garantir a validade da hipótese, pode-se atribuir a ela um certo grau de confiança, baseado em métricas quantitativas ou qualitativas das premissas, ou seja, quando as premissas utilizadas em inferência indutiva são verdadeiras, e existe uma quantidade suficiente de premissas, é possível dizer que uma hipótese é provavelmente verdadeira. Em inferência indutiva, as premissas adicionais acrescentadas podem ocasionar uma mudança no grau de confiança do argumento e neste caso, podem aumentar ou diminuir a confiança de um argumento indutivo, ou até mesmo, invalidá-lo.

Independente do paradigma de aprendizado escolhido é necessário uma maneira de descrever exemplos, modelos e o conhecimento do domínio. [Prati, 2006]. Nessa descrição podem ser usadas as seguintes linguagens de representação:

- Linguagens de descrição de exemplos;
- Linguagens de descrição de hipóteses;
- Linguagens de descrição de conhecimento do domínio.

No paradigma de aprendizagem de máquina simbólico, as linguagens de descrição mais freqüentemente utilizadas, podem ser classificadas em ordem crescente de complexidade e capacidade expressiva: de ordem zero, baseada em atributos e baseada em lógica de primeira ordem.

Lógica de Ordem Zero ou Proposicional

Na lógica de ordem zero ou cálculo proposicional a representação é descrita através de conjunções, disjunções e negações de constantes booleanas que representam atributos individuais. Por exemplo:

$$\text{fêmea} \wedge \text{adulta} \rightarrow \text{pode\_ter\_filhos}$$

Esta linguagem tem baixo poder descritivo, não é capaz de descrever objetos sobre os quais as relações são observadas.

Lógica de Atributos

Formalmente a lógica de atributos é equivalente à lógica proposicional, mas utiliza uma notação mais expressiva e flexível. Essa forma de notação é conhecida como formato atributo-valor, em que os atributos são tratados como variáveis e podem assumir diferentes valores. Por exemplo:

$$\text{sexo= feminino} \wedge \text{idade=adulta} \rightarrow \text{classe=pode\_ter\_filhos}$$

Mesmo que a maioria dos algoritmos de aprendizado faça uso da lógica de atributos para descrever exemplos e hipóteses, sua baixa capacidade de expressão impede a representação de objetos estruturados, a relação entre objetos e entre seus componentes. Dessa maneira, aspectos relevantes dos exemplos podem não ser representados.

Lógica de Primeira Ordem

Supera as limitações de representação impostas por uma linguagem de atributos, possui maiores vantagens em relação às lógicas citadas anteriormente. A lógica de primeira ordem permite descrever e raciocinar sobre objetos e predicados que especificam propriedades de objetos ou relacionamentos entre objetos do domínio.

As cláusulas de Horn [Casanova et al., 1987] constituem-se de um importante subconjunto pertencente da lógica de primeira ordem. Uma cláusula de Horn é composta por uma regra cuja cabeça contém um único predicado e um corpo formado por zero, um ou mais predicados. O exemplo seguinte, está descrito conforme a notação de Kowalsky [Kowalsky, 1979] para a linguagem de programação lógica Prolog, descreve que uma pessoa X é irmão da pessoa Y se

X é homem e ambos X e Y possuem o mesmo pai Z, em que X, Y e Z são variáveis que representam objetos.

$$\text{irmao}(X,Y) :- \text{homem}(X), \text{pai}(Z,X), \text{pai}(Z,Y).$$

O elemento à esquerda do símbolo $:-$ é a cabeça e o que está a direita do símbolo é o corpo da cláusula. O símbolo $:-$ é equivalente à implicação lógica $\leftarrow$ e é denominado de neck[1]. As vírgulas que separam cada predicado significam conjunções lógicas. Além disso, todas as variáveis estão sempre universalmente quantificadas, ou seja, no exemplo descrito a cláusula é verdadeira para todo $X, Y, Z \in D$. As variáveis entre parênteses são chamadas de argumentos. Nota-se que se todos os predicados não possuírem argumentos a linguagem se reduz à lógica de ordem zero e se todos os predicados possuem um único argumento constante (sem variáveis envolvidas), a linguagem se reduz à lógica de atributos.

2.2. Árvores de Decisão

A construção de Árvores de decisão é um dos métodos mais consagrados em aprendizagem de máquina simbólicos supervisionado. Algoritmos que induzem árvores de decisão pertencem à família de algoritmos *Top Down Induction of Decision Trees* (TDIDT). Uma árvore de decisão é construída a partir de uma base de exemplos de treinamento, os quais possuem um número finito de atributos. Esses atributos em conjunto são utilizados para prever a classe que está associada ao exemplo. Na Tabela 1 é possível visualizar o conjunto de dados de lentes de contato [Witten et al., 2000] [Flach & Lavrac, 2003]. Na Figura 1 é mostrado um exemplo de uma árvore de decisão induzida a partir de conjunto de dados na Tabela 1.

[1] $q :- p \equiv q \leftarrow p \equiv p \rightarrow q$

Idade	Espectropia	Astigmatismo	Produção Lacrimal	Lentes
jovem	miopia	não	reduzido	nenhuma
jovem	miopia	não	normal	macia
jovem	miopia	sim	reduzido	nenhuma
jovem	miopia	sim	normal	dura
jovem	hipermetropia	não	reduzido	nenhuma
jovem	hipermetropia	não	normal	macia
jovem	hipermetropia	sim	reduzido	nenhuma
jovem	hipermetropia	sim	normal	dura
pré-presbiotico	miopia	não	reduzido	nenhuma
pré-presbiotico	miopia	não	normal	macia
pré-presbiotico	miopia	sim	reduzido	nenhuma
pré-presbiotico	miopia	sim	normal	dura
pré-presbiotico	hipermetropia	não	reduzido	nenhuma
pré-presbiotico	hipermetropia	não	normal	macia
pré-presbiotico	hipermetropia	sim	reduzido	nenhuma
pré-presbiotico	hipermetropia	sim	normal	nenhuma
presbiotico	miopia	não	reduzido	nenhuma
presbiotico	miopia	não	normal	nenhuma
presbiotico	miopia	sim	reduzido	nenhuma
presbiotico	miopia	sim	normal	dura
presbiotico	hipermetropia	não	reduzido	nenhuma
presbiotico	hipermetropia	não	normal	macia
presbiotico	hipermetropia	sim	reduzido	nenhuma
presbiotico	hipermetropia	sim	normal	nenhuma

Tabela 1 - Conjunto de dados de lentes de contato [Witten et al., 2000] [Flach & Lavrac, 2003]

Uma árvore de decisão é uma estrutura de dados definida recursivamente como:

- um nó folha que corresponde a uma classe ou
- um nó de decisão que contém um teste sobre algum atributo (condição).

Para cada resultado do teste existe uma aresta para uma subárvore. Cada subárvore tem a mesma estrutura que a árvore.

Para classificar um novo exemplo basta começar pela raiz da árvore (nó inicial da árvore), seguindo cada nó de decisão de acordo com o valor do atributo do novo exemplo até que uma folha seja alcançada. Quando uma folha é alcançada, a classificação é dada pela classe correspondente ao nó folha.

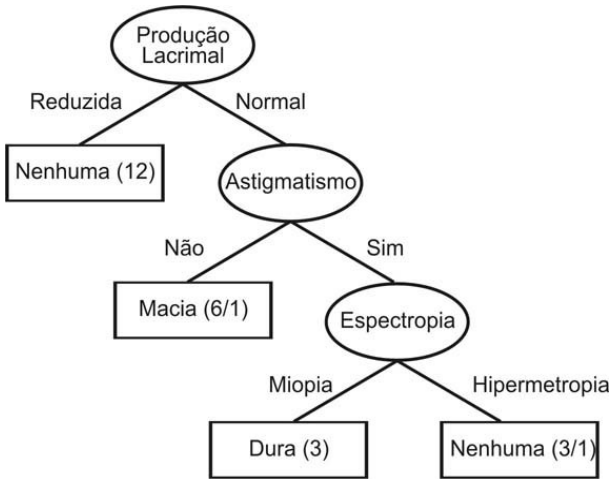


Figura 1 - Exemplo de Árvore de Decisão [Witten et al., 2000] [Flach & Lavrac, 2003]

O método para a construção de uma árvore de decisão a partir de um conjunto de treinamento E é relativamente simples. Para a construção considera-se que as classes do conjunto de exemplos de treinamento são $\{c_1, c_2, \ldots, c_n\}$ e os seguintes passos devem ser seguidos:

1. E contém um ou mais exemplos, todos pertencentes à mesma classe c_j. Nesse caso, a árvore de decisão para E é um nó folha rotulado pela classe c_j;

2. E não contém exemplos. Novamente, nessa situação, a árvore é uma folha mas a classe associada à folha dever ser determinada a partir da informação além de E. Por exemplo, a classe mais freqüente para o nó-pai desse nó pode ser utilizada;

3. E contém exemplos que pertencem a várias classes. Nesse caso, a idéia é dividir E em subconjuntos de exemplos que são (ou podem ser) formados pela maior quantidade de exemplos de uma única classe. Normalmente, um teste é escolhido baseado em um único atributo que possui resultados mutuamente exclusivos (cada sistema tem sua própria forma de escolher o atributo que será utilizado no teste). Sejam os possíveis resultados do teste denotados por $\{O_1, O_2, \ldots, O_i\}$. E é então particionado em subconjuntos $E_1, E_2, \ldots E_i$, nos quais cada E_i contém todos os exemplos em E que possuem como resultado do teste o

valor O_i. A árvore de decisão para E consiste em um nó interno identificado pelo teste escolhido e uma aresta para cada um dos resultados possíveis;

4. Os passos 1,2 e 3 são aplicados recursivamente para cada subconjunto de exemplos de treinamento de maneira que, em cada nó, as arestas levam para as subárvores construídas a partir do subconjunto de exemplos $E_i \subseteq E$.

A principal questão que envolve algoritmos de aprendizagem por árvores de decisão, ou melhor, utilizam a árvore de decisão como linguagem de descrição de hipóteses, são os critérios adotados para escolher o atributo ideal que particiona o conjunto de exemplos em cada iteração. Alguns critérios para a escolha dos atributos são:

Aleatória – seleciona qualquer atributo aleatoriamente;

Menor valor – seleciona o atributo com a menor quantidade de valores possíveis;

Maior valor – seleciona o atributo com a maior quantidade de valores possíveis;

Ganho máximo – seleciona o atributo que possui o maior ganho de informação, resultando possivelmente em subárvores de menor tamanho;

Razão de ganho – seleciona o atributo ponderando o ganho de informação esperado em relação ao nó pai, utilizado no sistema C4.5. [Quinlan, 1993]

Após a construção é possível que a árvore de decisão seja muito específica para o conjunto de treinamento. Nesse caso, ocorreu um *overfitting*, ou seja, a árvore de decisão superajustou os dados de treinamento.

Para solucionar o problema de superajuste de dados, alguns sistemas podam a árvore após a indução. Este processo é chamado de critério de pós-poda, em que é realizado a redução dos nós internos para melhorar o desempenho da árvore original. Existe também a pré-poda que é um processo efetuado enquanto a árvore de decisão está sendo construída, entretanto a pré-poda pode produzir um efeito negativo, porque atributos individuais podem não conseguir distinguir os exemplos quando são utilizados testes em conjunto para particioná-los.

2.3. Regras de Produção

Dado um conjunto de exemplos que são submetidos à tarefa de classificação, um algoritmo de aprendizagem de regras constrói um conjunto de regras do tipo *if-then*. Uma regra *if-then* tem o seguinte formato:

if condições **then** conclusão

As condições possuem uma ou mais restrições quanto aos valores que os atributos podem assumir. Os valores dos atributos podem ser de domínio discreto ou contínuo. A conclusão é atribuída a um valor particular que corresponde a classe que o exemplo está associado.

Uma sintaxe alternativa de regras muito utilizada é descrita desta forma:

$$Classe \leftarrow Condições\ ,$$

ou de forma mais geral:

$$Cabeça \leftarrow Corpo.$$

A última forma é utilizada em lógica de predicados como uma forma geral de regras na qual o *Corpo* (também chamado de antecedente) é uma conjunção de condições ou literais, e a *Cabeça* é o conseqüente que, no caso de regras *if-then*, é um simples literal (no caso geral, ela também pode ser uma disjunção de literais).

Um exemplo de um conjunto de regras induzidas utilizando o CN2 [Clark & Niblett, 1989] [Clark & Boswell, 1991] para o domínio de prescrição de lentes de contato, pode ser visualizado conforme a Figura 2 a seguir. Os números entre colchetes indicam a quantidade de exemplos do conjunto de treinamento, para cada uma das classes, cobertos pela regra. A classe com mais exemplos cobertos corresponde àquela prevista pela regra. A primeira e a terceira regra são capazes de classificar corretamente todos os exemplos, enquanto que nas demais regras isso não ocorre. Os valores que correspondem à distribuição de exemplos classificados corretamente podem ser utilizados para medir o suporte da regra. Outras conclusões podem ser obtidas a partir destes valores, como estimativa de probabilidade sobre todas as classes, ao invés da simples predição categórica. Por exemplo, se a descrição do paciente satisfaz a condição da segunda regra pode ser recomendada o uso de uma lente macia com probabilidade $5/6 = 0.8333\ldots$ ou nenhuma lente com probabilidade $1/6 = 0.1666\ldots$

```
IF   Produção Lacrimal = reduzido
THEN Lentes = nenhuma       [#macia=0, #dura=0, #nenhuma=12]

IF   Produção Lacrimal = normal
 AND Astigmatismo = não
THEN Lentes = macia         [#macia=5, #dura=0, #nenhuma=1]

IF   Produção Lacrimal = normal
 AND Astigmatismo = sim
 AND Espectropia = miopia
THEN Lentes = dura          [#macia=0, #dura=3, #nenhuma=0]

IF   Produção Lacrimal = normal
 AND Astigmatismo = sim
 AND Espectropia = hipermetropia
THEN Lentes = nenhuma       [#macia=0, #dura=1, #nenhuma=2]
```

Figura 2 - Regras de produção obtidas a partir do conjunto de dados de lentes de contato [Witten et al., 2000] [Flach & Lavrac, 2003]

Pode ser observado ainda que, na segunda regra, a condição `Produção Lacrimal = normal` é a negação da condição da primeira regra, e que essa restrição é incluída em todas as outras regras subseqüentes. Da mesma maneira, a condição `Astigmatismo = sim` presente na terceira regra é a negação da segunda condição na segunda regra. Neste sentido, o conjunto de regras pode ser equivalentemente representado por uma lista de decisão, conforme o exemplo da Figura 3.

```
IF Produção Lacrimal = reduzido THEN Lentes = nenhuma
  ELSE /*Produção Lacrimal = normal*/
    IF Astigmatismo = não THEN Lentes = macia
      ELSE /* Astigmatismo = sim*/
        IF Espectropia = miopia THEN Lentes = dura
          ELSE /*Espectropia = hipermetropia*/
            Lentes = nenhuma
```

Figura 3 - Lista de decisão induzida pelo conjunto de dados lentes de contato [Witten et al., 2000] [Flach & Lavrac, 2003]

Conjuntos de regras, listas e árvores de decisão são formas de representação muito correlacionadas e compartilham de muitas semelhanças. No entanto, existe uma diferença fundamental entre listas, árvores de decisão e conjuntos de regras não ordenados. Listas e árvores de decisão dividem o conjunto de exemplos em regiões disjuntas, ou mutuamente exclusivas, o que implica que cada exemplo é classificado por somente um única regra ou ramo da árvore. Em um conjunto de regras não ordenadas a disjunção nem sempre ocorre e se mais de uma regra cobrir o mesmo exemplo é preciso utilizar critérios mais adequados como solução. Além disso, assim como uma lista de decisão pode ser representada como uma árvore binária, uma árvore de decisão também pode ser representada como um conjunto de regras. Entretanto, regras não ordenadas não podem ser representadas como árvores, apenas como grafos.

2.3.1. Aprendizagem de Regras

Um algoritmo de aprendizagem produz uma hipótese ou modelo representado como um conjunto de regras. A construção desta hipótese geralmente envolve quatro estágios:

Construção da hipótese:

Para construir uma hipótese, o sistema de aprendizagem deve encontrar um conjunto de regras. Em aprendizagem envolvendo linguagens de representação de hipóteses equivalentes a lógica proposicional, esse estágio pode ser simplificado pela indução de regras seqüencialmente e independentemente, por exemplo, aplicando um algoritmo de cobertura. Em aprendizagem envolvendo linguagens de representação equivalente à lógica de primeira ordem a complexidade pode aumentar se a recursividade for utilizada, porque as regras não podem ser induzidas independentemente.

Construção da regra:

Uma regra é composta por $Cabeça \leftarrow Corpo$ e é construída atribuindo-se um valor de classe à $Cabeça$ e heuristicamente é construído o $Corpo$ da regra.

Construção do corpo da regra:

O corpo da regra é uma conjunção de condições. Para regras que utilizam representações baseadas em lógica proposicional, o corpo da regra é formado pela adição de condições, que inicialmente é vazio.

Construção de cada condição das regras:

As condições que compõe o corpo da regra podem ser formadas por igualdades se o domínio do atributo for nominal ou desigualdades para domínios numéricos. Esses são os casos simples, mas outras formas de construção de condições podem ser encontradas na literatura.

A construção das condições de regras pode ser vista como um caso particular de construção de atributos, se for considerado que cada condição é um atributo binário, com valor verdadeiro se o atributo satisfaz a condição e falso caso contrário.

O processo de indução de um conjunto de regras de classificação pode ser visto como um problema de busca, na qual o espaço de possíveis hipóteses é determinado pela linguagem de descrição de hipóteses utilizada. Em aprendizagem de regras do tipo *if-then*, o espaço de hipóteses é limitado por todas as possíveis regras neste formato.

No aprendizado de uma única regra, grande parte dos algoritmos de aprendizado faz uso de uma das seguintes estratégias:

Geral para específico ou abordagem *top-down* – Os algoritmos iniciam pela regra mais geral e repetidamente especializam essa regra enquanto cobrir exemplos negativos. Durante a busca, esse tipo de abordagem assegura que as regras induzidas cobrem pelo menos um exemplo positivo. Paralelamente enquanto constroem a regra, os algoritmos utilizam estratégias de refinamento sobre uma série de especializações da regra.

Específico para geral ou abordagem *bottom-up* – Iniciam pela regra mais específica que cobre exemplos positivos e gradativamente generalizam a regra, retirando condições desnecessárias. A generalização termina quando passa a cobrir exemplos negativos.

Os conjuntos de regras podem ser definidos como conjunto de regras não ordenadas e conjunto de regras ordenadas. No processo de indução de regras não ordenadas para problemas multiclasse, é eleita uma das classes como positiva e as classes restantes são agrupadas e consideradas como classe negativa. Este processo é iterativo e dessa maneira são construídos vários conjuntos de regras para cada uma das classes existentes. Esses conjuntos de regras são unidos em um único conjunto para formar um conjunto final de regras.

Conjuntos de regras não ordenadas podem apresentar sobreposição, ou seja, mais de uma regra pode cobrir o exemplo.

A diferença nos algoritmos de indução de regras ordenadas é que ao invés de remover apenas os exemplos que foram corretamente cobertos pela regra, todos os exemplos por ela cobertos são removidos. Dessa maneira, impossibilita que novas regras cubram exemplos anteriores incorretamente classificados. Conjunto de regras ordenadas são considerados mutuamente exclusivos, porque apenas uma das regras do conjunto é capaz de classificar o exemplo. A próxima seção discute os detalhes de um algoritmo de indução de regras utilizado neste trabalho.

2.3.2. RIPPER

O algoritmo de aprendizagem de indução RIPPER foi desenvolvido com o objetivo de executar aprendizagem de regras proposicionais, utilizando critérios de poda de forma incremental para reduzir erros. RIPPER é uma versão otimizada do IREP, proposto por William Cohen [Cohen, 1995].

Inicializar RS { } para cada classe da menos freqüente a mais freqüente

1.Etapa de construção:

Repetir as etapas de construção 1.1 e 1.2 até que o tamanho da descrição da regra ser maior do que o menor tamanho de descrição de regra encontrado ou não exista nenhum exemplo positivo ou a taxa de erro for maior ou igual a 50%.

1.1.Etapa de crescimento

Adicionar antecedentes ou condições à regra até a regra ser considerada perfeita (100% de precisão). O procedimento testa cada valor possível para os atributos e seleciona a condição com maior ganho de informação.

1.2.Etapa de poda

Realizar a poda incremental de cada regra de todas as seqüências finais dos antecedentes.

2.Etapa de otimização

Após gerar o conjunto inicial de regra {Ri} utilizar os procedimentos 1.1 e 1.2 para gerar e podar as regras do conjunto {Ri}. Todas as regras do conjunto {Ri} são analisadas e se existir exemplos positivos não cobertos pelas regras, novas regras são construídas seguindo as etapas anteriores.

3. As regras do conjunto {Ri} que aumentam o tamanho da descrição da regra são eliminadas e as restantes compõem o conjunto resultante.

O algoritmo RIPPER induz a classificação de regras ordenadas do tipo *if–then* para um conjunto de exemplos pré-rotulados, produz regras competitivas sobre domínios ruidosos, utiliza o critério de poda para reduzir taxas de erro, faz uso de métodos heurísticos como o princípio de descrição mínima de mensagem como critério de parada na adição de mais condições à regra. O princípio MDL (*Minimal Description Length*) [Quinlan, 1993], ou o tamanho de descrição mínima da mensagem, pode ser explicado como um modelo de comunicação em que um processo envia para um receptor a descrição de uma teoria T e um dado D do qual é derivado [Quinlan, 1993]. A descrição do tamanho da mensagem obtida consiste no custo necessário para descrever um dado.

O princípio MDL assume que a teoria derivada de um conjunto de exemplos vai minimizar a quantidade de bits necessários para codificar a mensagem completa. Em um problema de avaliação de um conjunto de regras, uma teoria é representada pelo conjunto de regras, o dado é representado pela base de validação e a mensagem é representada pela regra propriamente dita.

As regras são agrupadas pela cabeça da regra (H) (classe) e são criados k conjuntos de regras. O tamanho da informação é calculado para cada regra

pertencente à classe, dessa forma é possível calcular o tamanho da informação para todo o subconjunto. Na seqüência, são apresentadas as etapas para o cálculo do tamanho da informação do subconjunto:

• Para codificar um regra é necessário especificar cada condição presente no corpo da regra (B). A cabeça da regra (H) não precisa ser codificada, porque todas as regras presentes no subconjunto pertencem a mesma classe. O tamanho da informação em bits para um determinado conjunto de regras é $\log_2(prob)$ onde $prob$ é a probabilidade dos atributos se adequarem a regra;

• Codificar um conjunto de regras significa somar a quantidade de bits de cada regra, menos um crédito similar necessário para a ordenação das regras;

• As exceções são codificadas indicando quais dos casos cobertos pela regra S que são falsos positivos e aqueles que não são cobertos são os falsos negativos. Se as regras cobrem r dos n casos de treinamento, com fp (falsos positivos) e fn (falsos negativos), o número de bits necessários para codificar as exceções é:

$$excecao = \log_2\left(\frac{r}{fp}\right) + \log_2\left(\frac{n-r}{fn}\right)$$

O primeiro termo é a quantidade de bits necessários para indicar os falsos positivos entre os casos cobertos pela regra e o segundo termo indica a relação dos falsos negativos entre os exemplos não cobertos.

2.4. Medidas de Avaliação de Regras

Diversas medidas podem ser aplicadas para avaliar o desempenho de um modelo de classificação, no entanto, a precisão é a medida mais comum. Nem sempre é possível encontrar um modelo que tenha precisão adequada para classificar novos exemplos. As regras podem ser avaliadas com o objetivo de

identificar quais são aquelas que melhor são sustentadas pelos dados, ou ainda, podem ser avaliadas para selecionar as que possam trazer algum conhecimento surpreendente ou inesperado. A maioria das medidas de avaliação de regras estão baseadas na matriz de contingência para cada regra [Prati et al., 2002]. Considera-se cada regra no formato $Corpo \leftarrow Cabeça$,ou resumidamente $B \leftarrow H$, a matriz de confusão é mostrada na Tabela 2. [Lavrac et al.,1999].

Matriz de contingência			
	H	$\overline{H}$	
B	hb	$\overline{h}b$	b
$\overline{B}$	$h\overline{b}$	$\overline{hb}$	$\overline{b}$
	h	$\overline{h}$	n_{ex}

hb = número de exemplos para os quais H é verdade e B é verdade
$\overline{h}b$ = número de exemplos para os quais H é falso e B é verdade
$h\overline{b}$ = número de exemplos para os quais H é verdade e B é falso
$\overline{hb}$ = número de exemplos para os quais H é falso e B é falso
b = número de exemplos para os quais B é verdade
$\overline{b}$ = número de exemplos para os quais B é falso
h = número de exemplos para os quais H é verdade
$\overline{h}$ = número de exemplos para os quais H é falso
n_{ex} = número total de exemplos

Tabela 2 - Matriz de contingência para uma regra

Na Tabela 2, B denota o conjunto de exemplos para os quais a condição da regra é verdadeira e seu complemento $\overline{B}$ denota o conjunto de exemplos para os quais a condição da regra é falsa; o mesmo se aplica para H e $\overline{H}$. BH significa o conjunto de exemplos $B \cap H$ na qual ambos B e H são verdadeiros, $B\overline{H}$ representa o conjunto de exemplos $B \cap \overline{H}$ na qual B é verdadeiro e H é falso e assim por diante. Por generalidade, denota-se a cardinalidade de um conjunto A por a , ou seja, $a = |A|$. Assim, b denota o número de exemplos no conjunto B , ou seja, $b = |B|$, h representa o número de exemplos no conjunto H ,

ou seja, $h = |H|$, bh denota o número de exemplos no conjunto BH, ou seja, $bh = |BH|$ e assim por diante, e n_{ex} indica o número total de exemplos.

Denotando-se por $p(A)$ a freqüência relativa $frac|A|n_{ex} = a/n_{ex}$ associada ao conjunto A, no qual A é um subconjunto dos n_{ex} exemplos, podemos utilizar a freqüência relativa como uma estimativa de probabilidade. A notação $p(A/B)$ segue sua definição habitual de probabilidade condicional em teoria de probabilidade, dada pela equação abaixo, onde A e B são ambos subconjuntos do conjunto de n_{ex} exemplos.

$$P(A \mid B) \cong p(A \mid B) = \frac{p(A \cap B)}{p(B)} = \frac{p(AB)}{p(B)} = \frac{\frac{|AB|}{n}}{\frac{|B|}{n}} = \frac{\frac{ab}{n}}{\frac{b}{n}} = \frac{ab}{b}$$

Utilizando como base a matriz de contingência, é possível definir a maioria das medidas sobre regras, como a precisão (Acc), erro (Err), confiança negativa $(Neg\,\mathrm{Re}l)$, sensitividade $(Sens)$, especificidade $(Spec)$, cobertura (Cov) e suporte (Sup) definidas na Tabela 3.

A precisão de uma regra, também chamada de confiança, é uma medida de quanto essa regra é específica para o problema.

O erro de uma regra é o complemento da precisão. A confiança negativa de uma regra é correspondente à precisão, mas para os exemplos que não são cobertos pela regra.

A sensitividade de uma regra é semelhante ao *recall* de casos positivos usados em recuperação de informação; também conhecida como completeza, é uma medida do número (relativo) de exemplos da classe prevista em H cobertos pela regra.

Precisão	$Acc(B \rightarrow H) = P(H \mid B) = \frac{hb}{b}$
Erro	$Err(B \rightarrow H) = P(\overline{H} \mid B) = \frac{\overline{h}b}{b}$
Confiança Negativa	$Neg\,\mathrm{Re}\,l(B \rightarrow H) = P(\overline{H} \mid \overline{B}) = \frac{\overline{hb}}{\overline{b}}$
Sensitividade	$Sens(B \rightarrow H) = P(B \mid H) = \frac{hb}{h}$
Especificidade	$Spec(B \rightarrow H) = P(\overline{B} \mid \overline{H}) = \frac{\overline{hb}}{\overline{h}}$
Cobertura	$Cov(B \rightarrow H) = P(B) = \frac{b}{n}$
Suporte	$Sup(B \rightarrow H) = P(HB) = \frac{hb}{n}$

Tabela 3 – Medidas simples de avaliação de regras

A especificidade de uma regra é o correspondente à completeza, mas para os exemplos que não são cobertos pela regra. A cobertura de uma regra é uma medida do número (relativo) de exemplos cobertos pela regra. O suporte de uma regra é uma medida do número (relativo) de exemplos cobertos pela regra.

Cada uma das medidas descritas tem como objetivo avaliar um aspecto de uma regra. A precisão, por exemplo, tem como objetivo minimizar o número de exemplos incorretamente cobertos pela regra, no entanto, pode conduzir a casos extremos de uma regra ser considerada muito precisa e cobrir apenas um único exemplo. Medidas simples não são capazes de discriminar a regra mais geral entre duas regras ou mais regras com a mesma precisão, neste caso, a regra mais geral, que cobre o maior número de exemplos é a mais adequada.

Uma das maneiras de gerenciar medidas simples como precisão e generalidade é derivar medidas compostas. Geralmente escolhe-se medidas “ortogonais”, tais como a precisão e sensitividade, que dão origem à medida F

[vam Rijsbergen, 1979], definida na equação abaixo. Essa medida tem um parâmetro α que indica a importância relativa de cada uma das duas medidas.

$$F_{\alpha}(B \to H) = \frac{(\alpha+1) \times Acc \times Sens}{Sens + \alpha \times ACC}$$

Uma outra medida comumente utilizada para ponderar precisão e cobertura é a cobertura relativa com pesos [Lavrac et al., 2004], também conhecida como novidade, descrita abaixo.

$$WRacc(B \to H) = Nov(B \to H) = p(B)(p(H \mid B) - p(H)) = p(HB) - p(H)P(B)$$

Essa medida tem a propriedade de preferir regras um pouco menos precisas, mas com uma maior cobertura do que uma regra muito precisa, mas que cobre somente alguns exemplos.

Capítulo 3

Mineração Distribuída

Mineração distribuída de dados requer sistemas que possam minerar dados sobre muitas divisões separadamente. A distribuição de dados possibilita a mineração em diferentes subconjuntos, utilizando-se do paralelismo.

Segundo Freitas [Freitas & Lavington, 1998], mineração distribuída consiste em três fases:

1 - Dividir os dados para serem minerados em p subconjuntos de dados – onde p é o número de processadores disponíveis e enviar cada subconjunto a um processador distinto;

2 - Cada processador deve executar um algoritmo de mineração sobre o subconjunto de dados local. Os processadores podem executar o mesmo algoritmo de mineração ou diferentes algoritmos;

3 - Combinar o conhecimento descoberto por cada algoritmo de mineração em um conhecimento descoberto consistente global.

Técnicas em mineração distribuída [Prodomidis, 1999a], são utilizadas para resolver diferentes tipos de problemas, como:

• extensibilidade: suportam tecnologias de mineração novas e avançadas;

• portabilidade: capacidade de operar através de diferentes ambientes ou plataformas;

• escalabilidade: eficiência em minerar grandes volumes de dados;

• eficiência: determina a capacidade de utilizar fontes disponíveis;

• compatibilidade: integrar informações para base de dados similares, mas com diferentes esquemas, para gerar modelos mais precisos de bases de dados;

A escalabilidade é dependente de protocolos que permitem a transferência das informações entre os dados locais, enquanto que a eficiência é avaliada sobre o total de modelos disponíveis com relação a necessidade de minimizar o uso dos recursos. A questão principal é combinar escalabilidade e eficiência sem reduzir o

desempenho, permitir que protocolos de transferência operem de forma independente ou em colaboração, no entanto, sem depender um do outro. Protocolos de comunicação distribuída possibilitam a colaboração entre os dados locais, eliminam a centralização e sincronização de pontos de controle.

Freitas [Freitas & Lavington, 1998] relata que a mineração distribuída explora a concorrência e é uma técnica naturalmente aplicada, porque os dados a ser minerados estão fisicamente distribuídos e não podem ser centralizados em um único local, enquanto que em mineração paralela, o potencial está na otimização de algoritmos, particularmente para grandes bases de dados, em que estes dados não podem estar simultaneamente na memória principal de um único processador. Dados distribuídos é uma solução financeiramente mais viável que mineração paralela, em razão dos custos em arquiteturas de máquinas para mineração de dados em paralelo.

Uma comparação entre diferentes técnicas para aumentar a velocidade na mineração dados é ilustrada na Tabela 4. Para cada técnica presente na tabela, é indicado se a mesma reduz a quantidade de dados a ser minerados e/ou reduz o espaço de regras a ser pesquisado.

Técnica	Reduz a quantidade dos dados a ser minerados?	Reduz o espaço de regras a ser pesquisado?
Discretização	não	sim
Seleção de Atributos	sim	sim
Amostragem	sim	não
Pesquisa restrita	não	sim
Otimização de algoritmos	não	não
Mineração distribuída	sim[2]	não
Mineração paralela	não	não

Tabela 4 – Comparação entre técnicas para aumentar a velocidade de mineração dos dados

[2] Embora mineração distribuída utilize o conjunto completo de dados, a quantidade de dados que é dada para cada processador é reduzida.

Conforme a comparação entre diferentes técnicas de mineração, ilustrada na Tabela 4, a otimização de algoritmos e mineração paralela pode ser aplicada a mineração de dados, porém não reduzem a quantidade de dados ou o espaço de regras, utilizam o princípio de que o conhecimento descoberto pode ser o mesmo, sem a necessidade de utilizar técnicas para reduzir a quantidade de dados ou o espaço de regras. Mineração paralela certamente tem maior vantagem em aumentar a velocidade de mineração do que a otimização de algoritmos, principalmente quando se trata em mineração de grandes bases de dados, porque os dados a serem minerados não podem ser armazenados na memória de um único processador.

Métodos de mineração distribuída podem ser orientados a dados ou a algoritmos, como discutido nas seções posteriores.

3.1. Métodos Orientados a Algoritmos

Métodos orientados a algoritmos modificam algoritmos de mineração de dados, sem modificar os dados. Algumas técnicas orientadas a algoritmos são discutidas nesta seção.

3.1.1. Otimização de Algoritmos

No projeto de algoritmos utilizados em aprendizagem de máquina [Prodomidis et al., 1999a] considera-se que a precisão é a capacidade do algoritmo em classificar corretamente instâncias de diferentes subconjuntos de dados, e cobertura é a avaliação do desempenho do algoritmo sobre os conjuntos de dados, ou melhor, a capacidade de alcance de uma regra sobre estes exemplos.

Os critérios de poda [Prodomidis et al., 2000][Prodomidis et al., 1999b] utilizados em muitos algoritmos de aprendizagem, têm como objetivo usar métodos heurísticos capazes de otimizar a busca, tornando-se mais eficientes e

escaláveis, apresentando resultados melhores, com redução de tempo computacional se comparados a algoritmos que não utilizam técnicas de poda.

A pré–poda é um método realizado durante o treinamento e pode ser implementado de diferentes maneiras:

• analisando os classificadores candidatos disponíveis, baseado em métricas, independentemente e qualificando para posterior combinação em um meta-classificador;

• analisando os classificadores em relação a outros;

• verificando a precisão dos classificadores em relação a outros, baseado na cobertura obtida sobre o conjunto de dados.

A pós-poda avalia e poda classificadores base após o meta-classificador ter sido construído.

Estes são os objetivos da técnica de pós-poda:

• melhorar a pesquisa, sobre um espaço menor do que o anterior, sem reduzir o desempenho da predição do meta-classificador.

• minimizar o custo com a complexidade do uso da técnica de poda;

• adquirir informações combinadas de múltiplas bases de dados, para gerar meta-classificadores mais eficientes.

Segundo Willians [Williams et al., 1999], a diversidade dos formatos em que os dados podem ser armazenados e a necessidade de acessá-los de maneira oportuna requer otimizações em algoritmos que trabalham com a ortogonalidade dos dados.

Conforme Dayal entre outros autores [Dayal et al., 1999] , muitas pesquisas têm sido feitas no desenvolvimento de algoritmos, usando amostragem ou outras técnicas de redução de dados para minerar conjuntos muito grandes. Através do desenvolvimento de algoritmos escaláveis paralelos para agrupamento, classificação, regras de associação e outras tarefas de mineração percebe-se que somente a utilização de algoritmos escaláveis é insuficiente.

3.1.2. Processamento Paralelo

Conforme Provost e Kolluri [Provost & Kolluri, 1999], o processo de aprendizagem por indução é composto em dois níveis, por dois métodos principais de aprendizagem paralela: paralelização do espaço da pesquisa e combinação paralela. Os autores relatam sucesso com paralelização do espaço de busca em algoritmos de árvore de decisão, por utilizar a vantagem de multi-processadores em memória compartilhada, capazes de evitar a replicação ou comunicação de conjuntos de dados inteiros entre os processos. Utilizar parte da memória também permite o desenvolvimento de técnicas de balanceamento de carga de forma eficaz. Zaki [Zaki, 1998] utiliza a técnica de combinação paralela, de forma que as listas de atributos são divididas igualmente entre os processos e as quais retornam uma combinação estatística para uma lista mestre.

Outra técnica refere-se a regras de aprendizagem geradas pelo MetaDENDRAL [Aronis et al., 1997]. Segundo Aronis e Kolluri [Aronis et al., 1997], o MetaDENDRAL considera uma regra aceitável globalmente se e somente se ela pode ser considerada aceitável para a aprendizagem, usando o conjunto inteiro de dados. Não importa como os dados são divididos, todas as regras aceitáveis terão estatísticas aceitáveis em ao menos um subconjunto.

Freitas e Lavington [Freitas & Lavington, 1998], descrevem que uma forma de paralelização é a divisão de dados em subconjuntos, em que a aprendizagem é processada concorrentemente sobre os subconjuntos e quando uma regra é descoberta por um processador, transmite-se a todos os processadores restantes a fim de computar uma medida estatística global da qualidade da regra gerada, a qualidade no que diz respeito à precisão da predição. O uso de técnicas de paralelismo descobre regras que melhor modelam a tarefa de classificação. Duas técnicas de paralelização podem ser consideradas:

• Paralelização inter-algoritmo: cada algoritmo funciona de forma seqüencial, mas diversos algoritmos de mineração funcionam em paralelo, como se estivessem em múltiplos processadores. As limitações: (i) não é escalar em

relação ao aumento na base de dados; (ii) supõe que todos os dados podem ser acessados por todos os processadores, mas em um ambiente computacional de memória distribuída; (iii) todos os dados que estão sendo minerados teriam que ser replicados através da memória local de cada processador, com o objetivo de evitar o tráfego de dados.

• Paralelização intra-algoritmo: um algoritmo de mineração de dados funciona paralelamente em diversos processadores. Os processadores se comunicam e cooperam entre si durante a execução do algoritmo de mineração. Esta técnica supõe uma máquina paralela e uma comunicação entre processadores fisicamente distribuídos.

3.2. Métodos Orientados a Dados

Segundo Freitas e Lavington [Freitas & Lavington, 1998], três parâmetros influenciam no tamanho de um banco de dados: o número de tuplas, número de atributos e o número de valores que os atributos podem assumir. O tamanho do espaço das tuplas é o produto cartesiano dos atributos do domínio da base de dados. O tamanho do espaço das tuplas aumenta exponencialmente com o número de atributos e os atributos de domínio cardinal. É importante esclarecer que o aumento no espaço de tuplas aumenta o espaço de regras a ser pesquisado por um algoritmo de indução de regras.

O tamanho do espaço de tuplas e o tamanho dos dados armazenados em sistemas de banco de dados sugerem três técnicas para aumentar a velocidade em mineração dos dados:

• reduzir a cardinalidade do domínio de alguns atributos, em particular pode-se discretizar atributos contínuos;

• reduzir o número de atributos a ser minerado, aplicando um algoritmo de seleção de atributos à base de dados original;

• reduzir o número de tuplas a ser minerada, por extração de amostras para base de dados originais;

Estas técnicas discutidas por Freitas [Freitas & Lavington, 1998] têm como objetivo reduzir ou transformar os dados a serem minerados, portanto, não é necessário modificações nos algoritmos de mineração, as modificações são aplicadas simplesmente aos dados.

A discretização, uma etapa do pré-processamento, é uma técnica para transformar atributos contínuos em discretos, em que os valores são divididos em uma lista de intervalos e tem como objetivo aumentar a velocidade no processamento de algoritmos de mineração de dados. Algoritmos que constroem árvores de decisão, por exemplo, têm maior complexidade de tempo no processamento se os atributos não forem discretizados.

Outra técnica descrita por Freitas [Freitas & Lavington, 1998] é aplicar a seleção de atributos, em que a motivação principal é aumentar a precisão do algoritmo de mineração de dados, removendo atributos irrelevantes, reduzindo a escalabilidade de grandes bases.

Freitas [Freitas & Lavington, 1998] afirma que em base de dados reais uma quantidade muito grande de conhecimento pode ser necessária para executar transformações inteligentes na representação dos dados, o qual vai de encontro à filosofia de algoritmos de mineração de dados autônomos, inteligentes. Quanto menor o esforço do especialista mais autônomo é considerado a técnica.

Köpf entre outros autores [Köpf et al., 2000], afirmam que as características dos dados são fontes de conhecimento para seleção de pré-processamento e métodos de classificação. Segundo ele, os dados possuem características confiáveis que poderiam ser catalogadas utilizando diferentes métricas, entre medidas simples, como: número de observações, número de classes, distribuição das classes, entre outras medidas de teoria de informação.

As medidas de teoria de informação são utilizadas quando analisamos atributos categóricos, somente a distribuição dos atributos tem importância. É interessante comparar a distribuição com o atributo alvo e com a distribuição de um ou outro atributo para cada classe, para encontrar possíveis ligações. Isso pode ser feito pela medida de entropia que pode ser considerada uma analogia

qualitativa de medida de dispersão para atributos numéricos. A entropia é a medida de informação da distribuição de um atributo, a idéia básica é interpretar a entropia como distribuição da probabilidade.

As próximas seções discutem outras técnicas orientadas a dados.

3.2.1. Voto

A técnica de voto [Prodomidis et al., 2000], significa avaliar em todos os classificadores qual foi a classe mais votada para classificar cada uma das instâncias do conjunto de teste. A instância de teste será rotulada com a classe definida pela maioria dos classificadores de treinamento. De acordo com Chan [Chan & Stolfo, 1995], uma das variações da técnica de voto é o voto ponderado, que associa um peso determinado pelo desempenho da precisão do classificador base. Este peso é obtido através do conjunto de treinamento. Os pesos dos classificadores são avaliados e uma instância de teste será classificada de acordo com a classe com maior peso obtido.

Segundo Tsymbal [Tsymbal et al., 1999], a utilização da técnica do voto pode introduzir muitas imperfeições, porque quando uma nova instância é de difícil classificação, pode se obter uma previsão errada, e os votos da maioria dos classificadores base, muito provavelmente poderiam resultar neste erro. Neste caso, pode ser necessária a utilização de outra regra: a arbitrariedade, em que se a maioria dos classificadores base difere da decisão do árbitro, então a decisão do árbitro prevalece.

Littlestone e Warmuth [Littlestone & Warmuth, 1992] propuseram uma variação da técnica usando um algoritmo que considera o peso da maioria dos classificadores. Neste caso os algoritmos têm predições diferentes e os conjuntos de treinamento são usados apenas para calcular os pesos. A combinação é similar à técnica de voto ponderado. A principal diferença é como os pesos são obtidos. O algoritmo WM associa a cada classificador aprendido um peso inicial, para cada exemplo no conjunto de treinamento é processado pelos classificadores. A predição final é gerada pelo voto ponderado em que se a predição é errada, os

pesos dos classificadores que forneceram a predição são considerados incorretos e são multiplicados por um índice β onde $0 \leq \beta < 1,$ e diminui a contribuição final da predição.

3.2.2. Meta-Aprendizagem

A idéia básica de meta-aprendizagem, de acordo com Chan, Stolfo e Prodomidis [Prodomidis, 1999a], é executar um número de processos de aprendizagem de máquina sobre um número de subconjuntos de dados em paralelo e combinar os resultados dos classificadores através de uma etapa adicional de aprendizagem. Inicialmente cada tarefa de aprendizagem, chamada classificador base, faz o treinamento do classificador, em seguida, uma tarefa de aprendizagem em separado, chamada meta-aprendizagem, integra todos os classificadores base processados em um classificador meta de alto nível, em que a aprendizagem será sobre um conjunto de treinamento meta-nível. O conjunto de treinamento meta-nível é composto pelas predições dos classificadores base individuais. Para todas as predições, o conhecimento das características e o desempenho dos classificadores base são processados em um meta-classificador, em um modelo global de conjunto de dados.

Chan e Stolfo [Chan & Stolfo, 1995] propõem uma técnica que combina as a predição de modelos descobertos por algoritmos de mineração de dados locais. Os algoritmos de mineração de dados podem ser os mesmos ou algoritmos completamente diferentes. Os autores experimentaram duas técnicas básicas para combinar as predições locais geradas pelos algoritmos de mineração de dados local, a saber, a técnica do combinador e técnica do árbitro.

Na técnica combinador é aceito como entrada um conjunto de dados de treinamento (conjunto ajustado) que contém as predições feitas por cada algoritmo de mineração de dados local e pela predição correta das tuplas correspondentes (contida nos dados de treinamento e prevista pelo algoritmo de mineração local). Outra informação prevista, tal como os valores dos atributos, podem também ser

adicionadas ao conjunto de dados do meta-aprendizado, dependendo da estratégia a ser adotada para executar a meta-aprendizagem.

O uso do meta-aprendizado em um conjunto de dados para descobrir o relacionamento entre as classificações se faz pelos algoritmos de mineração locais e pelas classificações corretas. Na técnica árbitro, um árbitro-meta aprende por meio de algoritmos de mineração de dados locais. O árbitro aceita como entrada um conjunto de dados que contém tuplas, cujo valor global é predito em uma maneira inconsistente pelos algoritmos de mineração de dados locais diferentes.

Um árbitro aprende a escolher as classificações feitas pelo algoritmo de mineração de dados locais diferentes, quando um combinador pode fazer uma classificação completamente diferente das classificações feitas por qualquer algoritmo de mineração de dados local. Uma vez realizada a aprendizagem do árbitro, a classificação final é determinada fazendo uma verificação da classificação feita pelos algoritmos de mineração local e da classificação feita pelo árbitro. Estas classificações são combinadas usando algum tipo de regra de arbitragem.

Meta-aprendizagem contribui para a redução do problema de escalabilidade em aprendizagem de máquina, pois melhora a eficiência dos processos de aprendizagem dos classificadores base, por estes serem executados em paralelo. A técnica pode ser considerada escalável, porque o meta-classificador é a combinação de classificadores base combinados em um nível mais alto de aprendizagem, de uma maneira distribuída.

Segundo Chan [Prodomidis, 1999a], o uso de métricas ou propriedades diferentes qualificam este ou aquele classificador como melhor. A combinação de classificadores considerados melhores em um meta-classificador podem, provavelmente, formar classificadores com maior precisão e eficiência, sem utilizar buscas exaustivas em um espaço inteiro de possibilidades.

No entanto, Freitas [Freitas & Lavington, 1998] afirma que a precisão da predição conseguida com técnicas de meta-aprendizagem tende a reduzir

enquanto o número de subconjuntos de dados aumentar, desde que ocorra uma redução na quantidade de dados contido em cada subconjunto de dados. Além disso, técnicas de meta-aprendizagem podem reduzir a compreensibilidade na descoberta de conhecimento.

Em meta-aprendizagem [Prodomidis et al., 2000], a aprendizagem é definida pelo conhecimento aprendido, em que os classificadores base são inicialmente treinados e as predições são geradas por aprendizagem de classificadores separados.

3.2.3. Árbitro

Conforme Tsymbal, Puuronen e Terziyan [Tsymbal et al., 1999], a seleção de um classificador apropriado para estimar o valor de um atributo desconhecido de uma nova instância, tem um impacto essencial no resultado da qualidade da classificação. A técnica do árbitro foi proposta para a integração paralela de múltiplos classificadores e tem como vantagem a redução de dados, porque o conjunto de dados inteiro é dividido em pequenos subconjuntos, e os algoritmos de aprendizagem são então aplicados a esses subconjuntos.

A técnica do árbitro [Prodomidis et al., 2000] é considerada a segunda estratégia mais utilizada. O árbitro, em conjunto com a regra de arbitrariedade, escolhe o resultado final da classificação, baseado nas predições dos classificadores. (Figura 4)

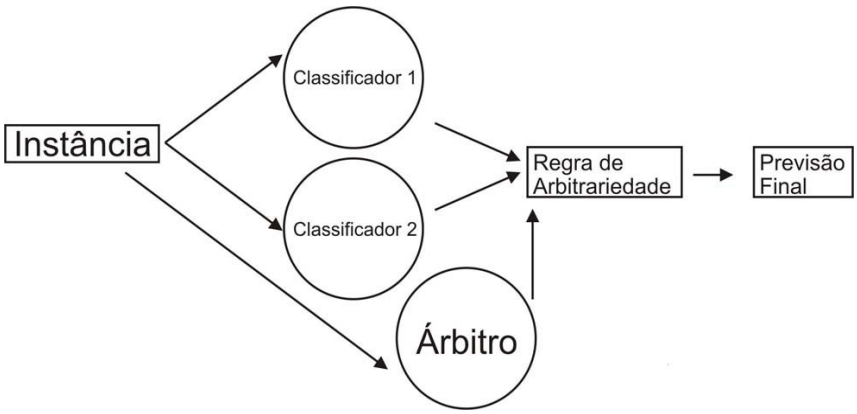


Figura 4 - Técnica do Árbitro

Gera-se um árbitro quando se aplica um algoritmo de aprendizagem nos exemplos que, não podem ser classificados por classificadores base. Classificadores base podem representar características específicas de um subconjunto, ou seja, obtido a partir da mineração de uma partição dos dados. O árbitro também utiliza uma regra para dar a decisão final. Em caso de empate, o árbitro decide a partir de amostragem ou redução/ seleção de atributos.

A técnica pode ser considerada um caso particular de *stacked generalization*, para integrar resultados de classificadores base através de um classificador de treinamento meta-nível.

Segundo Wolpert [Wolpert, 1992], *stacked generalization* é utilizada para minimizar a generalização da razão do erro para um ou mais generalizadores. *Stacked generalization* ou *stacking* cria conjuntos de modelos restringindo a cada modelo olhar somente um subconjunto de entradas. A idéia de *stacking* é utilizar um método de combinação de modelos, em que parte do conjunto de treinamento é utilizado no treinamento nível 0, ou treinamento base. No nível 1, o modelo é construído a partir do modelo do nível 0 e do restante do conjunto; ainda no nível 1 os modelos de dados treinados neste nível são generalizados. S*tacking* utiliza o método de validação cruzada e, através das partições do conjunto de aprendizagem, sobre uma partição do treinamento, observa o comportamento sobre esta outra partição na tentativa de deduzir tendências, o que significa que partições de dados podem construir também teorias aceitáveis.

Stacking utiliza predições de diferentes classificadores que são usados como entrada para a etapa de meta-aprendizagem. Por exemplo, as predições para classificadores em árvore, modelos lineares e rede neural podem ser utilizadas como entrada de um meta-classificador com rede neural na tentativa de combinar a aprendizagem obtida de diferentes modelos com o objetivo de maximizar a precisão da classificação.

3.2.4. Combinador

Chan, Stolfo e Prodomidis [Prodomidis et al., 2000] fazem uma distinção entre classificadores base, a técnica árbitro e combinador. O classificador base produz o resultado, classifica a instância de teste, aplicando um algoritmo de aprendizagem diretamente e fornece uma predição de uma classe à instância. O combinador e árbitro atuam sobre as predições produzidas pelo conjunto de classificadores base, para que o conjunto de predições seja processado de maneira hierárquica.

Na técnica combinador, os meta-dados que compõe o conjunto de treinamento são formados a partir das saídas dos classificadores base. O objetivo é aprender as relações entre as previsões dos classificadores e a classificação correta. Quando uma classificação é gerada, uma regra de composição usa as decisões dos classificadores base para gerar uma meta-instância, em que todas são adicionadas em um meta-conjunto de treinamento, para criar um meta-classificador.

Chan [Chan & Stolfo, 1995], afirma que a regra de composição varia conforme o esquema da base de dados e determina o conteúdo do conjunto de treinamento para a meta-aprendizagem, utilizado para a geração um meta-classificador (combinador).

Na classificação de uma instância, primeiramente os classificadores base geram as predições que são transformadas em uma meta-instância utilizando a regra de composição que finalmente será classificada pelo combinador – Figura 5. Pelo menos duas regras de composição são possíveis:

• os *m* atributos de previsão para uma meta-instância são as decisões dos *m* classificadores base e o atributo-meta (a classe) é a mesmo;

• atributos de previsão são as decisões dos classificadores base e os atributos de previsão originais da base de dados. O atributo-meta é o mesmo.

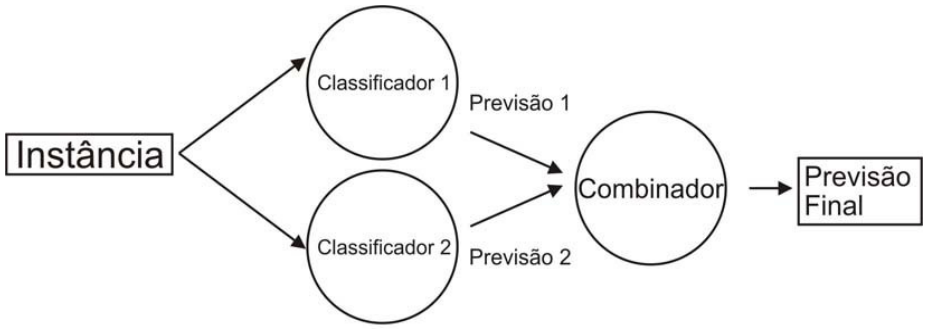


Figura 5 - Combinador

3.2.5. Multi-Esquema

Outra estratégia simples da combinação é a seleção de um classificador entre diversos. O desempenho de um classificador pode ser medido no conjunto de dados a partir do percentual de classificação considerada correta. Multi-esquema é a escolha entre todos os classificadores base qual classificador possui a melhor qualidade. Essa técnica talvez seja mais suscetível aos modelos locais, devido à construção de classificadores a partir de amostras, pois elas podem ter distribuições muito diferentes.

3.2.6. Outros Métodos

Em mineração distribuída existem diversos métodos desenvolvidos a fim de viabilizar a mineração em grandes bases de dados. Alguns foram citados nas seções anteriores. Os métodos descritos abaixo têm algumas semelhanças e são considerados importantes na extração de conhecimento implícito em grandes bases.

3.2.6.1. Bagging

Bagging foi introduzido por Breiman [Breiman, 1996] com o objetivo de combinar vários classificadores, produzindo um melhor taxa de acerto sobre os dados de teste. Bagging seleciona amostras de tamanhos iguais, a partir dos

dados de treinamento e gera classificadores para cada amostra. Cada amostra de treinamento é formada por instâncias selecionadas aleatoriamente, mas com recobrimento, ou seja, uma instância pode aparecer repetidas vezes ou nenhuma em qualquer subconjunto de treinamento. A classificação de uma instância é realizada pelo critério de votação simples.

3.2.6.2. Boosting

De acordo com Schapire [Schapire & Freund, 2001], *boosting* explora o problema da complementaridade nos classificadores, garantindo modelos mais complementares possíveis. Existem várias versões de *boosting*, mas a idéia geral é que todas as instâncias de treinamento são inicializadas com um peso. Estes pesos são utilizados para estimar o erro dos classificadores. O erro de um classificador é medido pela soma dos pesos das instâncias mal classificadas dividido pela soma de todos os pesos das instâncias. O método força a atenção do algoritmo nas instâncias mal classificadas porque os pesos destas instâncias são aumentados e o peso das instâncias classificadas corretamente é diminuído. O erro global é processado e o processo é repetido diversas vezes até que o erro global produzido pelo classificador seja aceitável.

3.2.7. Integração de Modelos

A integração de modelos consiste na união de modelos locais descobertos por cada algoritmo de mineração de dados (de seu subconjunto de dados local) em um modelo global consistente. Por exemplo, Fayyad [Fayyad et al., 1996] propõe um sistema que integra conjuntos locais de regras em um conjunto global de regras. Neste sistema, um algoritmo de construção de árvores de decisão é aplicado inicialmente a diversas amostras aleatórias dos dados que foram minerados a fim de construir rapidamente uma árvore de decisão para cada amostra. Estas árvores locais de decisão são convertidas então em conjunto de regras locais, e cada regra tem sua qualidade avaliada por um teste estatístico,

mais precisamente teste de Fischer. Este teste é aplicado a cada condição, no antecedente da regra (parte “se” da regra), a fim de identificar as condições que são estatisticamente irrelevantes para predizer a classe no conseqüente da regra (parte “então” da regra). As condições irrelevantes são removidas do antecedente da regra. Além disso, o significado do teste estatístico é aplicado também ao antecedente da regra com um todo, a fim de identificar e remover as regras irrelevantes.

Aronis entre outros autores [Aronis et al., 1997], relatam que combinar múltiplas bases requer conhecimento sobre as informações presentes nas bases e determinar quais campos são relevantes. Muitos problemas surgem ao proceder à união de tabelas, de banco de dados relacionais, a ser minerada, como por exemplo:

• a união pode requerer muito espaço;

• o espaço para a previsão do conjunto de teste pode aumentar, para cobrir o espaço de todos os valores de cada atributo;

• as tabelas alvo do ambiente distribuído podem manter apenas ponteiros para base de dados remotas que por conseqüência apontam para outras tabelas estrangeiras;

• o processo de união manual consome muito esforço e está sujeito a erros por omissão e a tendências (*bias*).

Algoritmos de indução de regras [Chan & Stolfo, 1998] constroem classificadores com altas taxas de precisão, mesmo assim, muitos fatores reduzem a qualidade na aprendizagem:

• ao usar o mesmo algoritmo, diferentes distribuições nos conjuntos de treinamento podem gerar classificadores com precisões diferentes;

• usar a distribuição natural das classes pode não obter um desempenho expressivo;

Conforme Kurgan [Kurgan, 2004], muitas pesquisas tentam propor técnicas de aprendizagem com excelente precisão, baixa complexidade computacional,

mas o tamanho do modelo de regras geradas na aprendizagem não é muitas vezes relatada.

Segundo Kurgan [Kurgan, 2004], indução é uma técnica que infere no modelo de dados, ou conhecimento, por pesquisa de regularidade entre os dados. O sistema de indução de regras toma como entrada um conjunto de exemplos de treinamento, com aprendizagem supervisionada. A saída, muitas vezes, é na forma de regras, ou árvores de decisão que podem ser convertidas em regras.

Sistemas de indução de regras têm várias vantagens sobre outras metodologias de mineração de dados:

• as hipóteses assumem formas de fácil compreensão humana;

• regras gerais poderiam facilmente ser modularizadas em uma única regra;

• conhecimentos anteriores têm origem em regras existentes;

Kurgan [Kurgan, 2004] propõem uma estrutura de meta-modelo, que utiliza o conceito de meta-mineração, em que os modelos gerados pelos classificadores base são combinados em um modelo único, mais generalizado. A meta-mineração divide a entrada de dados em subconjuntos a serem minerados, e gera um modelo para cada subconjunto. Em seguida, toma os modelos gerados e generaliza em um meta-modelo, com as seguintes características:

• os resultados gerados são mais compactos e descobrem conhecimento interessante;

• redução da complexidade das regras geradas;

• o meta-modelo melhora em relação a escalabilidade, porque a mineração é processada em conjuntos menores de dados;

Hall [Hall et al., 1999] descreve que cada regra criada pode estar associada a uma medida de qualidade, baseada na precisão e no número de exemplos que ela cobre. Utiliza-se uma versão normalizada do fator de certeza desenvolvido por Quinlan [Quinlan, 1987] para determinar a precisão da regra R sobre um exemplo E, desta forma:

$$acc(R,E) = \frac{(TP - 0.5)}{(TP + \rho FP)}$$

• onde TP é o número de exemplos verdadeiros positivos cobertos por R quando aplicado a E;

• FP é o número exemplos positivos falsos causados por R quando aplicado a E;

• ρ é a relação de exemplos positivos aos exemplos negativos para a classe ou a regra contida no conjunto de treinamento.

A regra R, deve ter $acc(R,E) \geq t$ para algum ponto inicial t a fim de ser considerada aceitável sobre um conjunto de exemplos E.

Quando a regra é construída com um único subconjunto de dados, talvez sua precisão possa mudar quando é aplicada a um outro subconjunto de dados. A regra pode ser descartada sempre que a precisão é menor que t ou somente depois que for aplicada a todos os exemplos e ter uma precisão abaixo do ponto inicial.

De acordo com Freitas [Freitas & Lavington, 1998] um método para aumentar a velocidade de processamento de algoritmos de indução de regras foi proposto por John e Langley. A técnica é realizada através de amostragem iterativa e dinâmica, em que cada iteração consiste em escolher uma pequena amostra de dados para ser minerados, é aplicado o algoritmo de mineração de dados à amostra e avalia-se o resultado da classificação. Em cada iteração, o tamanho da amostra é aumentado por um número constante de tuplas até que a diferença entre a precisão da classificação sobre a amostra e a precisão de classificação estimada sobre o conjunto inteiro de dados minerados, seja menor do que o valor especificado pelo usuário. Uma característica interessante do método é que a precisão da classificação sobre o conjunto inteiro dos dados pode ser estimada, sem minerá-lo inteiramente, por considerar o histórico da precisão das classificações obtidas em amostras de vários tamanhos em iterações procedentes do método.

Algoritmos de mineração baseados em indução de regras procuram por regras em um grande espaço. Essencialmente, o espaço de regras está associado com o algoritmo de mineração de dados e é o conjunto de todas as regras que podem ser expressas na linguagem de representação usada pelo algoritmo. O espaço de regras pode ser muito grande para ser pesquisado exaustivamente, e os algoritmos de mineração usam algumas heurísticas para procurar somente em partes do espaço.

Lawrence O. Hall entre outros autores [Hall et al., 1999] construíram um modelo único de aprendizagem para um conjunto de dados, os quais são distribuídos em uma rede de computadores e generaliza regras em paralelo. O objetivo é: dado N subconjuntos haverá N conjuntos de regras geradas. Cada subconjunto de dados reside em um processador distinto. Todos os conjuntos distribuídos de regras devem ser unidos em um único conjunto. Regras em conflito e com baixa performance são removidas: um exemplo de conflito ocorre quando duas regras são geradas para a mesma classe e foram criadas em diferentes subconjuntos disjuntos, mas têm sobreposição na cobertura. Nestes casos, a regra mais geral prevalece e é utilizada. A união em um conjunto de regras deve estar livre de conflitos e a precisão deve ser equivalente a um conjunto de regras de um conjunto inteiro de dados utilizados no treinamento.

Em outra técnica desenvolvida por Provost e Hennessy [Provost & Hennessy, 1996], o primeiro passo é a construção de um conjunto de regras (modelo) para cada conjunto de dados originais, os conjuntos de regras foram unidos até ser obtido finalmente um modelo dos dados. A técnica constrói um único e preciso conjunto de regras para N conjuntos de regras geradas pelos subconjuntos, em que a regra obtida por um subconjunto utiliza a propriedade de cooperação, para assegurar que em um único conjunto seja composto por regras que satisfazem os outros subconjuntos. A propriedade de cooperação é definida como o compartilhamento de regras e pode aumentar a qualidade do conhecimento obtido ou reduzi-lo diretamente. A precisão pode diminuir se o grau de paralelismo aumentar.

Kurgan e Cios [Kurgan & Cios, 2003] descrevem os problemas ocorridos pela introdução de um novo sistema de mineração chamado MetaSqueezer. O sistema faz uso do conceito de meta-mineração para gerar modelos de dados para meta-dados. Como vantagem, produz modelos de conhecimento compactos, escaláveis e convenientes para paralelização. O sistema extrai um conjunto de regras que descrevem conceitos-alvo para dados supervisionados.

Algumas características obtidas pelo sistema Meta Squeezer:

• Gera modelos compactos, na forma de regras de produção. Com menor número de regras sendo estas mais compactas, permite fácil compreensão e avaliação;

• Tem complexidade linear em relação ao número de exemplos de entrada de dados, e pode ser utilizado em grandes bases de dados;

• A nova arquitetura do sistema suporta paralelização, produzindo uma melhora de performance. O sistema é naturalmente capaz de adaptar-se a ambientes distribuídos. A arquitetura do sistema é baseada no conceito de meta-mineração, descrita nas seções posteriores.

O modelo de meta-mineração descrito por Kurgan [Kurgan & Cios, 2003], aplica o mesmo algoritmo base de aprendizagem aos dados para produção de hipóteses em dois passos, onde o resultado final é gerado a partir dos resultados obtidos pelos classificadores base. A meta-aprendizagem tem como objetivo descobrir a melhor estratégia de aprendizagem através da adaptação contínua dos algoritmos em diferentes níveis de abstração, como por exemplo, através de seleção dinâmica de tendências. Meta-mineração tem sido usada em regras de associação e é capaz de descrever mudanças nos dados. Kurgan [2004] em outra pesquisa relata a redução da complexidade de regras baseada em modelos. O trabalho também é baseado no conceito de meta-aprendizagem. O autor argumenta que a redução da complexidade da aprendizagem precisa levar em conta a redução do tamanho e o número de regras geradas, sem reduzir a precisão do conjunto de regras como um todo.

Muitas técnicas de meta-aprendizagem têm sido desenvolvidas, com o objetivo de selecionar um classificador apropriado para estimar o valor de um atributo desconhecido de uma nova instância.

Tsymbal [Tsymbal et al., 1999] entre outros, consideram a técnica usada na aprendizagem de classificadores em um número de subconjuntos em paralelo e seleciona cada instância para o melhor classificador, dinamicamente. A técnica limita a quantidade de dados pesquisados em um único processo de aprendizagem e, fornece dinamicamente a seleção de um classificador, sem perda significativa ou melhora da precisão. A estrutura proposta de meta-classificação consiste em dois níveis. O primeiro nível contém os classificadores base, enquanto que o segundo, contém a combinação dos erros preditos para cada classificador base. No treinamento, é obtido o desempenho dos classificadores base para cada instância. O desempenho do classificador base é armazenado, e posteriormente é usado junto com o treinamento do conjunto inicial como meta-nível de conhecimento para estimar o erro de classificação para a nova instância. O desempenho do classificador base é obtido utilizando a técnica de validação cruzada.

Chan e Stolfo [Chan & Stolfo, 1995], comparam a técnica do voto e meta-aprendizagem sobre dados particionados. A técnica proposta utiliza grandes quantidades de dados particionados em subconjuntos, em cada subconjunto é realizado a aprendizagem e posteriormente os resultados são combinados. Os autores concluem o trabalho afirmando que combinar resultados de classificadores separados pode não obter maior taxa de precisão do que os resultados obtidos em um conjunto inteiro dos dados.

Métodos de integração de conjuntos de regras foi interesse particular neste projeto, pois desenvolvemos um modelo de integração baseado em Lógica Paraconsistente. A maioria dos métodos estudados anteriormente necessita de um processo intenso de troca de mensagens entre processadores distribuídos, gerando normalmente um grande fluxo de informações e utilização de recursos de rede. Na técnica desenvolvida neste trabalho, nossa pretensão foi amenizar este

problema gerando um processo de tomada de decisão que é local, onde todas as regras geradas pelos classificadores estão disponíveis. As seções posteriores introduzem alguns conceitos que utilizamos para desenvolver nosso método.

Capítulo 4

Técnicas de Gerenciamento de Incerteza

Pesquisas em Inteligência Artificial necessitam que o conhecimento, bem como quaisquer outras informações pertinentes estejam disponíveis. No entanto, em muitos domínios reais, a informação nem sempre está presente de forma completa e consistente. Nestas situações, é necessário raciocinar quando há falta de informações e fazer uso de formalismos que possam servir de apoio como soluções.

"Conhecimento incerto é o conhecimento que não é discutível, mas ao qual está associada alguma medida de incerteza que descreve crenças para as quais existem certas evidências de apoio" [Rich & Knight, 1994]

Em Ladeira [Ladeira & Viccari, 1996], é descrito como o conhecimento pode ser deficiente em uma ou mais das seguintes formas:

• *o conhecimento é parcial:* quando informações importantes não estão disponíveis.

"Ayrton Senna morreu em um acidente automobilístico, ocasionado por uma falha mecânica, durante o Grand Prix de Ímola, em 1994" – não está explícito a falha que ocorreu.

• *conhecimento não é completamente confiável:* diversos motivos podem conduzir para que o conhecimento não possa ser considerado confiável.

"A morte de Senna foi provocada por uma falha no sistema de direção do seu Fórmula 1". – não esclarece o tipo de falha.

• *linguagem de representação imprecisa:* não existe clareza na linguagem utilizada e interpretações equivocadas podem levar a inferências erradas.

"Ao bater no muro de proteção, Senna sofreu desaceleração de cerca de 314 km/h para zero, em frações de milésimos de segundos" – linguagem imprecisa.

• *conhecimento é conflitante:* informações conflitantes são comuns em bases de dados, principalmente quando estão descentralizadas, podem existir dados contraditórios.

"Algumas fotos de amadores, feitas no dia da corrida, aparentemente mostraram um defeito na suspensão. Alguns pilotos e ex-pilotos acham que foi a barra de direção." – informações conflitantes de fontes diversas.

Em Inteligência Artificial, segundo Parsons [Parsons, 1996], existe uma divisão simbólica para tratar problemas com informações incompletas. De um lado, estão as técnicas para tratamento de informações incompletas, que geralmente utilizam mecanismos de inferência, baseados em lógica clássica ou suas extensões, como lógicas não monotônicas, a partir de suposições sobre dados faltantes. Do outro, estão os métodos numéricos, que combinam lógica com quantificadores numéricos através do uso de técnicas numéricas. No entanto, há possibilidade de gerar novos problemas em decorrência desta combinação, o que conduz o desenvolvimento de técnicas puramente numéricas.

O conceito de informação imperfeita, segundo Sandra Sandri [Sandri 1996], normalmente é conhecido como incerteza. De acordo com Sandri, o termo incerteza é muito restritivo e que se aplica a um tipo específico de imperfeição. Vamos supor que se deseja descobrir o horário de início de uma sessão no cinema. As informações fornecidas podem ser classificadas desta forma:

• "A sessão começa às 20:15 h." – informação perfeita
• "A sessão começa entre 20:00 e 21:00 h." – informação imprecisa
• "A sessão começa próximo de 20:30 h." – informação vaga
• "Imagino que a sessão começa às 20:15 h." – informação incerta

• “É provável que a sessão comece às 20:00 h.” – informação probabilística

• “É possível que a sessão comece às 20:00 h.” – informação possível

• “Maria disse que a sessão começa às 20:00 h, mas João disse que começa às 21:00 h.” – informação inconsistente.

• “Eu não sei que horas a sessão inicia, mas normalmente iniciam às 20 h” – informação incompleta.

Nas próximas seções são apresentadas algumas das principais técnicas utilizadas no raciocínio sobre informações incertas, entre as quais: Lógica Fuzzy e Lógica Paraconsistente.

4.1. Lógica Fuzzy

Os primeiros trabalhos sobre Lógica Fuzzy foram desenvolvidos por Zadeh em 1965 [Zadeh, 1965]. A Lógica Fuzzy tem como objetivo modelar o modo aproximado de raciocínio, inspirada na habilidade humana de tomar decisões em ambientes de incerteza e imprecisão.

A aplicação das propriedades dos conjuntos fuzzy permitem um tratamento mais adequado onde à teoria clássica de conjuntos é pouco eficiente. Por exemplo, dado um determinado problema que utiliza o conceito de “homem alto”. A definição que o conjunto “homem alto” é formado por todos os homens que possuem altura maior ou igual a 1.80 m. Mas visualmente, não poderia ser sido que um homem que possui altura de 1.795 é alto? De acordo com a teoria clássica dos conjuntos essa afirmação não seria possível.

Antes de apresentar os principais conceitos de Lógica Fuzzy, é importante recordar de alguns conceitos de teoria clássica de conjuntos.

Dado que X denota um conjunto de elementos, x denota um elemento individual de X. A representa um subconjunto de X e μ_{A} é uma função característica – ou uma função de pertinência (conforme Tabela 5) – de A se e somente se:

$$\mu_{\mathrm{A}}(X)=\begin{cases}1 & se \quad \mathrm{X}\in \mathrm{A}\\ 0 & caso \quad contrário\end{cases}$$

Geralmente, o conjunto de $\{0,1\}$ é chamado de conjunto de valoração e o valor associado pela função μ, para um dado x, é chamado de grau de pertinência. Portanto os valores possíveis para um função de pertinência, são 1 e 0. Por exemplo, assumindo X ser o conjunto de números inteiros e C o conjunto de números múltiplos de 5, então:

$\mu_C(x)=1$ para $x=35$ e

$\mu_C(x)=0$ para $x=27$

Na teoria dos conjuntos Fuzzy, o conjunto de valoração é estendido de $\{0,1\}$ para o intervalo $[0,1]$, ou seja, o grau de pertinência de um determinado elemento x pode assumir qualquer valor numérico pertencente a $[0,1]$. Intuitivamente, $\mu_C(x)=1$ é equivalente a $x\in C$. Por exemplo, seja X o conjunto de todas as pessoas e T o conjunto de todas as pessoas altas, a Tabela 5 descreve a função $\mu_T(x)$.

x	altura (m)	$\mu_{\mathrm{T}}(x)$	Significado
João	1.80	1.0	João é alto
Pedro	1.70	0.5	Pedro é meio alto
Paulo	1.55	0.0	Paulo não é alto

Tabela 5 – Função de Pertinência

A utilização de técnicas dos conjuntos Fuzzy depende da definição de uma função característica para cada conjunto empregado. Por exemplo, a função

$\mu_T(x)$, descrita anteriormente, poderia ser definida como demonstrada na Tabela 6.

$\mu_T(x)$	
0	para $altura(x) \leq 1.60$
$((altura(x)) - 1.60)/2$	para $1.60 \leq altura(x) \leq 1.80$
1	para $altura(x) \geq 1.80$

Tabela 6 - Definição da Função de Pertinência

Uma característica importante, inerente às funções de pertinência, é a sensibilidade ao contexto do problema. Por exemplo, se o universo de pessoas do conjunto X representasse jóqueis de turf, então, talvez o valor de $\mu_T(Pedro)$ seria mais adequado se fosse igual a 1.

Lógica Fuzzy não está relacionada à probabilidades, portanto os valores obtidos através de uma função $\mu_D(x)$ não representam a probabilidade de x pertencer ao conjunto D, e sim o grau de pertinência de x em relação a D.

As duas operações principais definidas por Zadeh são a interseção $\cap$ e a união $\cup$. Ambas são definidas da seguinte forma:

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)) \text{ e}$$
$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x))$$

Muitas outras definições foram propostas para as operações de interseção e união, geralmente com o objetivo de satisfazer certas proposições algébricas como monotonicidade, comutatividade e associatividade [Henkind & Harrison, 1988]. Porém foram definidas normas triangulares ou normas T para a formulação da operação de interseção e co-normas T para formulação da união, pois nem

todos os trabalhos utilizam a formulação de $\min$ e $\max$ para interseção e união, respectivamente.

Pode-se relacionar a teoria de Lógica Fuzzy com lógicas multi-valoradas – em que o valor-verdade de uma proposição pode assumir infinitos valores [Ávila, 1996] [Subrahmanian, 1987a] [Subrahmanian, 1987b]. Além disso, é possível utilizar qualificadores lingüísticos para aumentar o conjunto de valores de uma função de pertinência. Porém, a utilização de qualificadores implica na definição explícita de cada um deles para uma determinada variável.

A Figura 6 representa graficamente a definição de uma variável chamada temperatura que possui três qualificadores lingüísticos: baixa, média e alta.

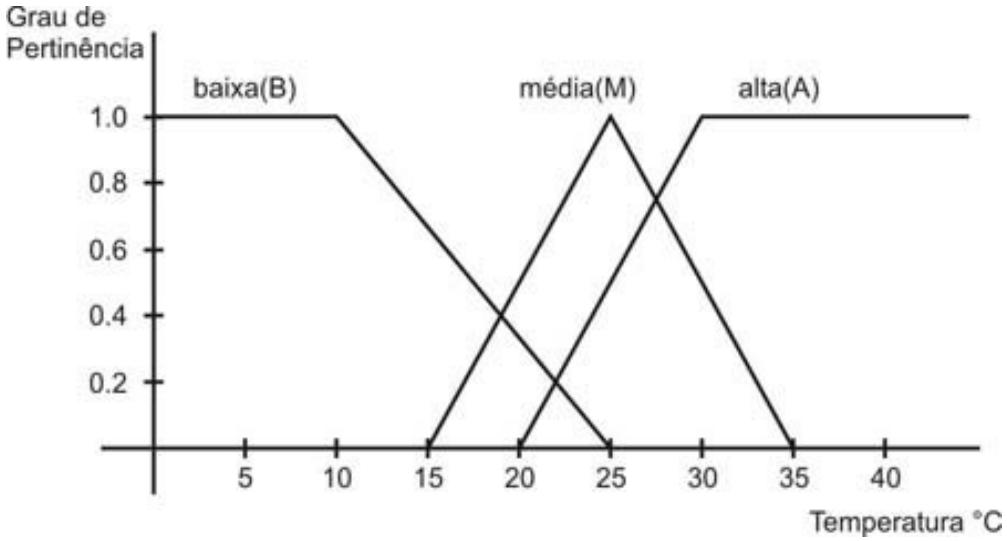


Figura 6 - Definição da Variável Temperatura [Enembreck, 1999]

De acordo com a definição dos conjuntos Fuzzy, as operações de união e interseção entre os conjuntos B e M ilustrados na Figura 6 são representadas nas Figura 7 e Figura 8, respectivamente.

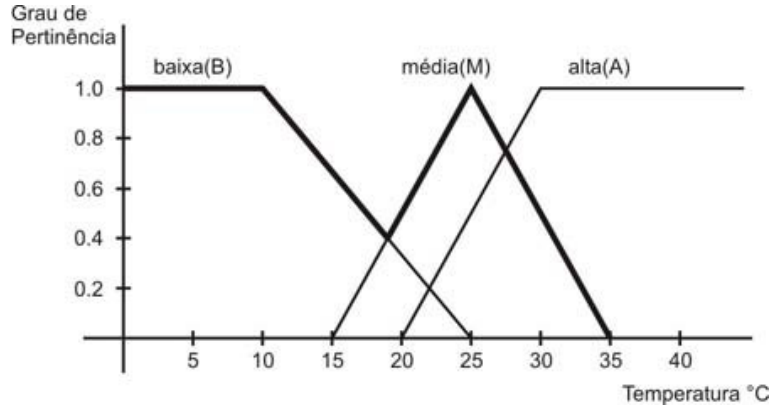


Figura 7 - União entre os Conjuntos B e M

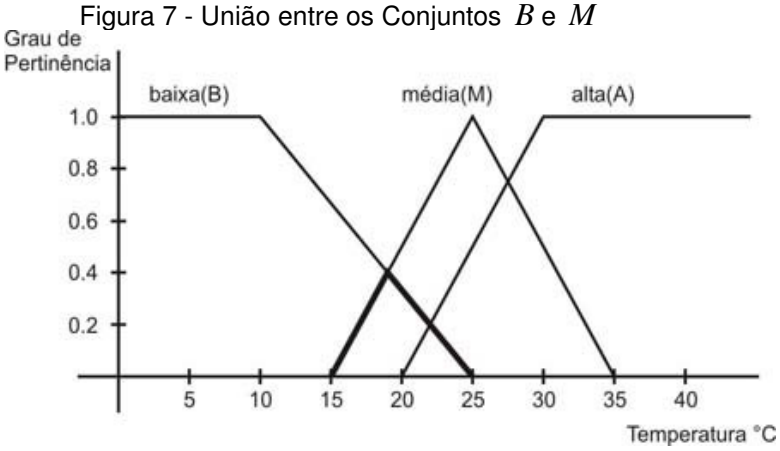


Figura 8 - Interseção entre os Conjuntos B e M

4.2. Lógica Paraconsistente

A inconsistência é um fenômeno natural ao mundo real e a todos que fazem parte dele. O ser humano consegue tratar a inconsistência de maneira satisfatória; quando ocorre uma inconsistência, sabiamente sempre tenta obter mais informações que auxiliem a sair desse estado.

Um exemplo na literatura é quando uma pessoa está atravessando uma região pantanosa e recebe uma informação visual [Hasegawa, 2004] – ocorrência de uma vegetação rasteira – de que o solo à sua frente é firme, mas, para confirmar essa informação utiliza um pequeno galho de árvore, e então descobre que o solo não é tão firme quanto parecia ser. Se avançar ela poderá ficar presa, então, ao ocorrer esse conflito de informações a pessoa poderá buscar novas

evidências através de novos testes. Pode verificar a dureza do solo com um galho maior, ou então jogar uma pedra. Após obter informações suficientes para sair do estado de inconsistência, ela poderá tomar uma decisão adequada, seja avançar com cautela ou procurar novo caminho. Sendo assim, a eliminação da inconsistência – como é feita em diversos sistemas computacionais – pode não ser a melhor alternativa, uma vez que informações importantes poderiam ser eliminadas.

A proposta deste projeto foi desenvolver um método de integração de modelos capaz de processar o grau de crença e descrença para cada regra de cada subconjunto de treinamento, identificar e tratar dois ou mais conjuntos de regras com o mesmo conseqüente, porém com atributos contraditórios, considerados inconsistentes.

4.2.1. Conceitos

Na Lógica Paraconsistente (LP) existe o princípio da contradição – de duas proposições contraditórias, uma é obrigatoriamente falsa. Ao contrário da Lógica Clássica, na LP é possível representar e realizar inferências sobre informações contraditórias, e também distinguir as situações onde uma determinada proposição é realmente falsa de uma em que não se tem conhecimento suficiente para se chegar a uma conclusão. A conclusão obtida pode ser muito útil em tomada de decisões em que não há informações suficientes, ou elas são contraditórias.

4.2.2. Mecanismo de Raciocínio

Seja *T* uma teoria fundamentada sobre uma lógica *L,* e supomos que a linguagem de *T* e de *L* contenha o símbolo para a negação. A teoria *L* é inconsistente se ela possuir teoremas contraditórios, isto é, um é negação do outro. Caso contrário, diz-se que *T* é consistente. Uma teoria *T* é trivial se todas as fórmulas de *L* forem teoremas de *T*, ou seja, tudo o que possa ser expresso na

linguagem *T* possa ser provado em *T;* caso contrário, *T* diz-se não trivial. Na maioria dos sistemas lógicos usuais é impossível distinguir a proposição verdadeira da falsa, pois qualquer proposição pode ser provada em *T.* Uma lógica *L* é chamada Paraconsistente se servir de base para teorias inconsistentes, mas não triviais. Na Lógica Evidencial Paraconsistente (*LEP*), uma proposição *p* está associada a dois fatores evidenciais $([c,d])$ que representam, respectivamente, a quantidade de *crença* e *descrença* da proposição. Os fatores evidencias pertencem ao intervalo $[0,1]$, ou seja, são infinitamente valorados. Os valores-verdade são compostos pelo fatores evidenciais e pertencem ao reticulado $\tau = \langle |\tau|, \leq \rangle$, onde:

$$|\tau| = \{c \in \Re | 0 \leq x \leq 1\} \times \{d \in \Re | 0 \leq x \leq 1\}$$

O reticulado τ pode ser representado através do diagrama de Hasse da Figura 9. No reticulado pode ser observado um ponto máximo $[1,1]$ que indica a inconsistência (T), um ponto mínimo $[0,0]$ que indica a indeterminação $(\perp)$, a verdade (v) é representada pelo ponto $[1,0]$ e o falso (f) é representada pelo ponto $[0,1]$.

O estado de inconsistência ocorre quando se acredita tanto na verdade quanto na falsidade de uma proposição em um determinado instante de tempo.

O estado de indeterminação ocorre quando não há informações sobre a verdade nem sobre a falsidade.

O estado de verdade ocorre quando se acredita totalmente na verdade e na há nenhuma informação que suporta a falsidade, por sua vez o estado de falsidade é o oposto.

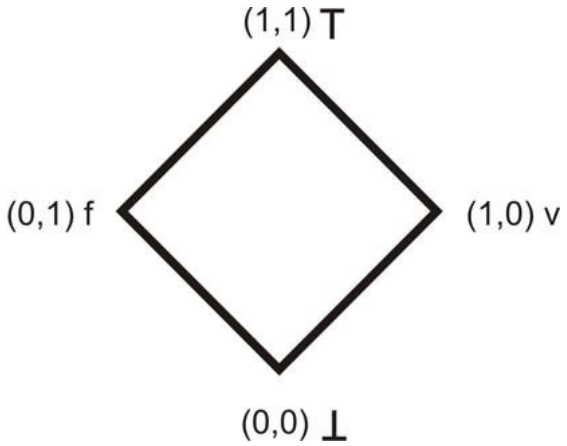


Figura 9 - Reticulado representado através o Diagrama de Hasse

Dada a proposição “João matou Pedro”, temos:

• Se anotarmos com $[1,1]$, existe uma inconsistência na proposição, alguém forneceu uma informação que João estava com Pedro no momento do crime, enquanto outra pessoa afirmou que naquele momento João estava no cinema;

• Se anotarmos com $[0,0]$, ninguém soube dizer onde João estava, ninguém viu com Pedro nem em outro lugar;

• Se anotarmos com $[1,0]$, crê-se totalmente que João matou Pedro, alguém afirmou que João estava com Pedro e viu o crime ocorrer;

• Se anotarmos com $[0,1]$, crê-se totalmente que João não matou Pedro, alguém informou que João não estava com Pedro no momento do crime;

O resultado de uma inferência no *Paralog_e* [3] [Ávila, 1996] é o fechamento dos fatores evidenciais do predicado que foi consultado na inferência. É possível obter o *grau de contradição* (G_{ct}) e o *grau de certeza* (G_c) a partir do *grau de crença* e do *grau de descrença* de uma proposição.

O G_{ct} é o valor que representa, no reticulado, a distância entre os dois estados extremos *inconsistente* e *indeterminado* – Figura 10 – e é obtido por:

[3] O ParaLog_e é um interpretador escrito na linguagem Prolog baseado no Paralog [da Costa et al., 1995]. O ParaLog_e utiliza conceitos de Programação Lógica Evidencial Paraconsistente baseada em Lógica Anotada com notações infinitamente valoradas.

G_{ct}= $c + d - 1$, para : $0 \leq c \leq 1$ e $0 \leq d \leq 1$

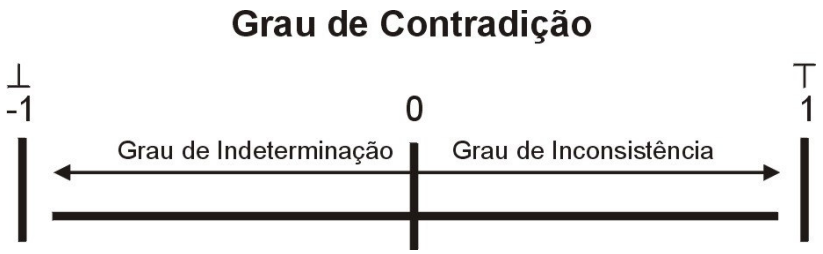


Figura 10 - Representação do grau de contradição

O G_c é o valor que representa, no reticulado, a distância entre dois estados extremos *verdadeiro* e *falso* – Figura 11 – e é obtido por:

G_c= $c - d$, para : $0 \leq c \leq 1$ e $0 \leq d \leq 1$

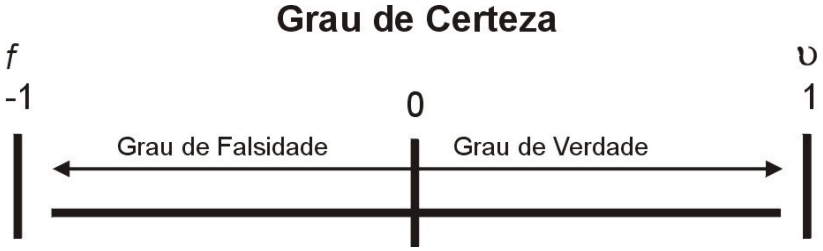


Figura 11 - Representação do grau de certeza

Ao inter-relacionarmos o G_{ct} e o G_c em dois eixos, obtém-se o reticulado τ representado com valores que podem ser quantificados e equacionados – Figura 12. Essa representação é útil para obter o estado lógico discretizado de uma inferência. O algoritmo *Para-Analisador* [da Costa et al., 1999] realiza uma discretização do G_{ct} e o G_c interpolando-os nesse reticulado e o ponto de encontro é o estado lógico resultante.

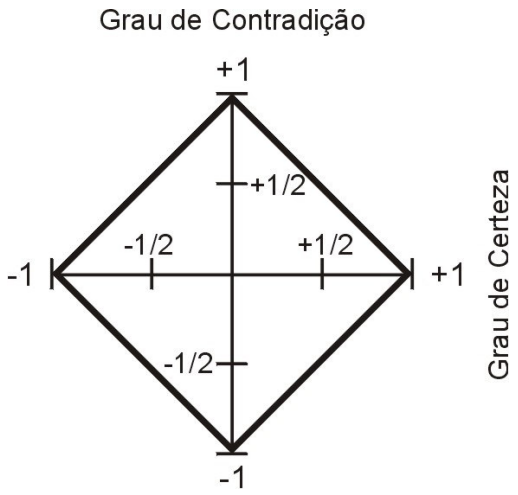


Figura 12 - Representação do G_{ct} e do G_c inter-relacionados

Existem quatro valores de controle:

• *Vscc (Valor Superior de Controle de Certeza):* limita o grau de certeza próximo ao verdadeiro;

• *Vicc (Valor Inferior de Controle de Certeza):* limita o grau de certeza próximo ao falso;

• *Vscct (Valor Superior de Controle de Contradição):* limita o grau de contradição próximo ao inconsistente;

• *Vicct (Valor Inferior de Controle de Contradição):* limita o grau de contradição próximo ao indeterminado.

A discretização é realizada pelo algoritmo Para-Analisador que se baseia no reticulado que é dividido em áreas que correspondem a um estado lógico. O G_{ct} e o G_c são interpolados no reticulado e o ponto de encontro está contido em uma área que corresponde a um estado lógico. Os limites que indicam os estados extremos podem ser regulados utilizando os valores de controle – Figura 13.

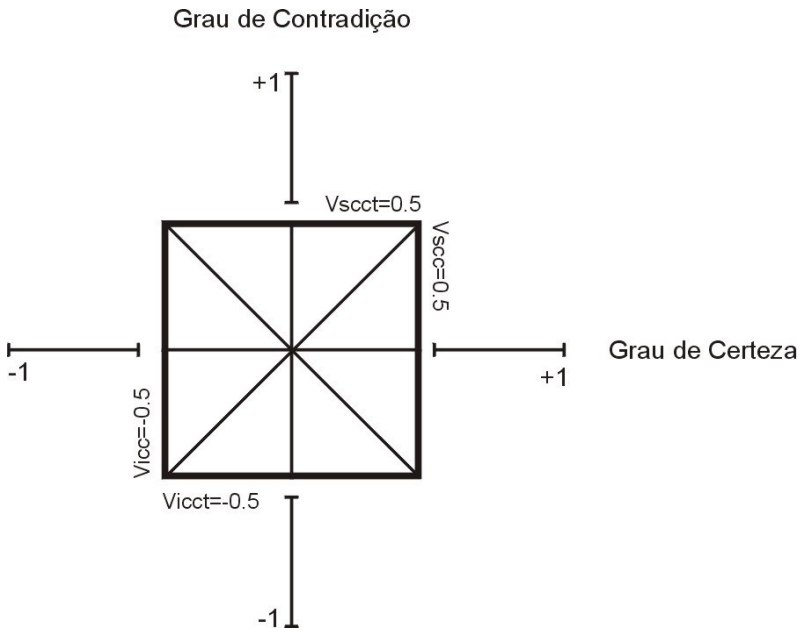


Figura 13 - Exemplo de controle de limites com limites configurados para 0.5 e -0.5 representado no gráfico de G_{ct} e G_c.

É possível definir um reticulado com mais dois estados lógicos de acordo com as necessidades de aplicação. Quanto mais estados lógicos maior será a precisão na análise do G_{ct} e do G_c. Exemplo de um reticulado com 12 estágios lógicos – Figura 14.

Onde:

- T: inconsistente;
- $T \rightarrow v$: inconsistente tendendo a verdade;
- $T \rightarrow$ f: inconsistente tendendo a falso;
- v : verdade;
- $Qv \rightarrow$ T: quase verdade tendendo a inconsistente;
- $Qv \rightarrow \perp$: quase verdade tendendo a indeterminado;
- f: falso;
- $Qf \rightarrow$ T: quase falso tendendo a inconsistente;
- $Qf \rightarrow \perp$: quase falso tendendo a indeterminado;
- $\perp$: indeterminado;

- $\perp \rightarrow \upsilon$: indeterminado tendendo a verdade;
- $\perp \rightarrow f$: indeterminado tendendo a falso;

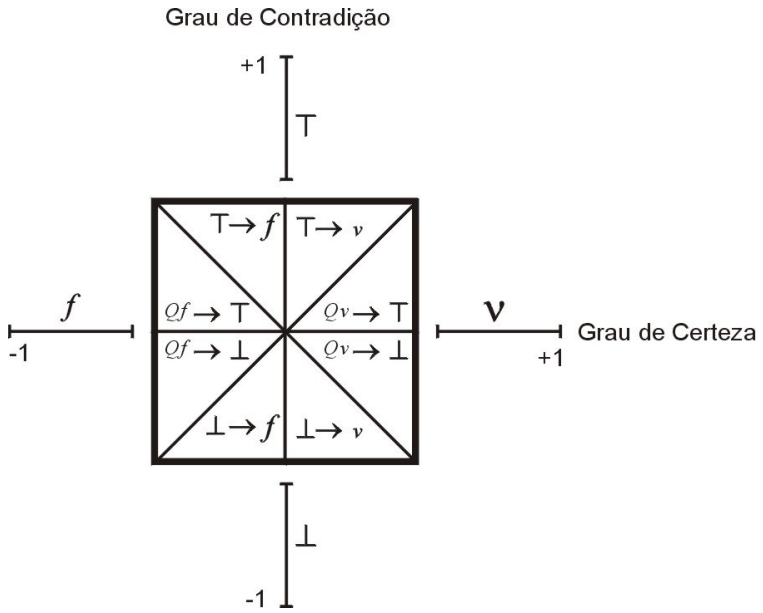


Figura 14 - Reticulado com 12 estados lógicos representados no gráfico de G_{ct} e G_c

4.2.3. Aplicações

A Lógica Paraconsistente obteve resultados significativos em aplicações de diversas áreas da computação, como por exemplo, na verificação automática de assinaturas manuscritas, desenvolvido por Enembreck [Enembreck, 1999], em que aprendizagem de máquina e programação lógica evidencial paraconsistente – PrLEP – foram utilizados conjuntamente para extrair informações úteis e confiáveis sobre dados imperfeitos.

Ávila [Ávila, 1996], propôs uma abordagem baseada em Lógica Paraconsistente para tratar exceções, tratar adequadamente questões como as exceções e inconsistências em sistemas de Frames com múltipla herança. Para atingir o objetivo, foi implementado em Paralog_e [Ávila, 1996] um raciocinador de herança paraconsistente, pelo fato de, até então, não existir uma semântica formal tanto para sistemas de frames paraconsistentes quanto para raciocinadores de

herança que tratam exceções e inconsistências em sistemas de frames com múltipla herança, e foi a motivação para a implementação desse tipo de raciocinador.

O sistema Pandora[4] [Angelotti, 2001] é um sistema multi-agente e baseia-se na arquitetura Multicheck [Scalabrin et al., 1998], e utiliza LP na implementação de mecanismos de raciocínio, que são utilizados para validar ou interpretar informações obtidas por algoritmos de reconhecimento de padrões. A interpretação dos campos lógicos *numérico* e *literal* do cheque podem ser efetuadas de maneira interativa e aproximada, onde os agentes numérico e literal interagem trocando crenças e raciocinando sobre elas.

4.3. Programação Lógica Paraconsistente

Os conceitos de raciocínio lógico evidencial representados por meio da Programação Lógica Evidencial – PLE, utiliza-se de valores verdade composto por dois fatores evidenciais pertencentes a um reticulado $\{x \in \Re \mid 0 \leq x \leq 1\} \times \{x \in \Re \mid 0 \leq x \leq 1\}$. A cada item de conhecimento de um sistema lógico evidencial são associados fatores evidenciais de crença e descrença. Crença representa a quantidade de crença que apóia a verdade da evidência. Descrença representa a quantidade de descrença associada à verdade da descrença, ou a crença associada a falsidade da evidência. Ambos os fatores pertencem ao intervalo [0,1]. Ou seja, infinitos valores podem estar associados às premissas do sistema.

Com os fatores evidenciais associados as premissas é possível obter interpretações sobre a veracidade ou não da informação.

Na seqüência são apresentadas algumas definições lógicas na construção e inferência de Programas Lógicos Evidenciais. [Ávila, 1996]

[4] Pandora, na mitologia grega, foi a primeira mulher criada por Júpiter. Como presente de casamento, Pandora recebeu uma caixa, denominada *Caixa de Pandora,* onde cada deus colocava um bem. “Pandora abriu a caixa, inadvertidamente, e todos os bens escaparam, exceto a esperança. Assim, mesmo que tudo nos escape a esperança não nos deixa inteiramente”.

Definição de Negação – O operador de negação $\sim:|T|$ é definido como $\sim([c,d])=[d,c]$

4.3.1. Sintaxe

Literal Evidencial – Se p é uma fórmula básica e $c,d \in \{x \in \Re \mid 0 \leq x \leq 1\}$ então, diz-se que $p:[c,d]$ é um literal bem anotado e que $[c,d]$ é a anotação de p.

Cláusula Evidencial – Se $p_0:[c_0,d_0],\ldots,p_n:[c_n,d_n]$ são literais bem anotados, então $p_0:[c_0,d_0] \Leftarrow p_1:[c_1,d_1] \wedge \ldots \wedge p_n:[c_n,d_n]$ chama-se de cláusula evidencial ou cláusula-e. $p_0:[c_0,d_0]$ é a cabeça da cláusula, enquanto $p_1:[c_1,d_1] \wedge \ldots \wedge p_n:[c_n,d_n]$ é o corpo da cláusula.

Unificação – Se $p:[c_0,d_0]$ e $q:[c_1,d_1]$ são literais, então diz-se que $p:[c_0,d_0]$ e $q:[c_1,d_1]$ são unificáveis se p e q são unificáveis.

Programa Lógico Evidencial (PLE) – Um PLE é qualquer conjunto finito não-vazio de cláusulas-e.

4.3.2. Semântica

A base Herbrand [Nicoletti & Monard, 1993] [Casanova et al., 1987] é o domínio de todas as interpretações. Uma interpretação é uma função $I:B_L \rightarrow T$, tal que B_L é a base Herbrand e T é o respectivo reticulado. $I(A)$ é o valor verdade associado por I ao literal A.

Definição 1 A interpretação I satisfaz um literal anotado $p:[c,d]$ se $I(p) \geq [c,d]$ (denotado $I \models p:[c,d]$).

Definição 2 A interpretação I satisfaz uma conjunção $p_1 : [c_1, d_1] \wedge \ldots \ldots \wedge p_k : [c_k, d_k]$ se $I \models p_i \forall (1 \leq i \leq k)$.

Definição 3 A interpretação I satisfaz uma cláusula-e $p : [c, d] \Leftarrow q_1 : [c_1, d_1] \wedge \ldots \wedge q_k : [c_k, d_k]$ se e somente se:

- $I \models p : [c, d] \wedge q_1 : [c_1, d_1] \wedge \ldots q_k : [c_k, d_k]$ ou
- $I \not\models q_1 : [c_1, d_1] \wedge \ldots \wedge q_k : [c_k, d_k]$

Definição 4 Diz-se que uma interpretação I satisfaz um PLE E se satisfaz todas as cláusulas-e de E. Portanto I é um modelo para E.

De maneira semelhante à ordenação $\leq$ estabelecida sobre as constantes anotacionais, é estabelecida a ordenação entre as interpretações:

$I_1 \leq I_2$ se e somente se $(\forall P \in B_E) I_1(p) \leq I_2(p)$ tal que B_E é a base Herbrand de E.

Definição 5 Se E é um PLE, define-se T_E como uma aplicação de interpretações Herbrand de E em interpretações Herbrand de E, tal que

$T_E = \sup\{[c, d] \mid p : [c, d] \Leftarrow q_1 : [c_1, d_1] \wedge \ldots \wedge q_k : [c_k, d_k]\}$ é a instância básica de uma cláusula-e em E e $I \models q_1 : [c_1, d_1] \wedge \ldots \wedge q_k : [c_k, d_k]$.

Definição 6 Um modelo I que atribui valores verdade $[c, d]$ ao átomo $[p]$, sendo $c + d > 1$ é dito sobre-determinado.

Definição 7 Um modelo I do PLE E é correto, se é correto em relação a todo átomo $p \in B_E$.

Definição 8 Um modelo I do PLE E é completo, se é completo em relação a todo átomo $p \in B_E$ se $I(p) = [c,d]$ e $c + d \geq 1$.

Definição 9 Um modelo I do PLE E é completo, se é completo em relação a todo átomo $p \in B_E$.

Definição 10 Diz-se que um PLE E é bem-comportado se as cláusulas-e de E satisfazem a seguinte condição:

Se C_1 e C_2 são cláusulas-e em E, sendo suas cabeças $p_1 : [c_1, d_1]$ e $p_2 : [c_2, d_2]$ e p_1 e p_2 são unificáveis, então

$$\max(c_1, c_2) + \max(d_1, d_2) < 1$$

4.3.3. Semântica Operacional dos Programas Lógicos Evidenciais

Definição 11 Se E é um PLE, p é um átomo na linguagem E e $[c,d]$ uma anotação, define-se uma árvore e/ou $T(E, p[c,d])$ da seguinte forma:

• a raiz de $T(E, p : [c,d])$ é um nó "ou" rotulado $P : [c,d]$;

• se N é um nó "ou", então ele é rotulado por um literal anotado simples;

• cada nó "e" é rotulado por uma cláusula-e em E e por uma substituição θ;

• descendentes de nós "ou" são nós "e" e descendentes de nós "e" são nós "ou";

• se N é um nó "ou" rotulado por $p : [c,d]$ e se $C\theta$ é um instância de uma cláusula-e C em E da seguinte forma: $p : [c_1, d_1] \Leftarrow q_i : [c_i, d_i] \wedge \ldots \wedge q_k : [c_k, d_k]$ tal que $[c_1, d_1] \geq [c,d]$, então existe um descendente N rotulado por C e θ. Um nó "ou" sem descendentes chama-se nó não-informativo;

• se N é um nó “e” rotulado por um cláusula-e C e a substituição θ, então para todo literal $p:[c,d]$ no corpo de C, existe um nó “ou” descendente rotulado $p\theta:[c,d]$. Um nó “e” sem descendentes chama-se nó sucesso.

Associado a cada nó N da árvore e/ou definida anteriormente, existe uma constante anotacional $v(N)$ chamada valor do nó, da seguinte forma:

• se N é um nó sucesso rotulado por $p:[c,d]$, então $v(N)=[c,d]$;

• se N é um nó não-informativo, então $v(N)=[0,0]$;

• se N é um nó “ou” que não é não-informativo e seus descendentes são $N_1,\ldots,N_m$, então $v(N)=\sup\{v(N_1),\ldots,v(N_m)\}$;

• se N é um nó “e” não-terminal rotulado pela cláusula-e $p:[c_1,d_1]\Leftarrow q_i:[c_i,d_i]\wedge\ldots\wedge q_k:[c_k,d_k]$, e se o valor $v(N_i)$ de cada um dos nós descendentes N_i rotulados q_i é tal que $v(N_i)\geq[c_i,d_i]$ para todo $1\leq i\leq m$, então $v(N)=[c_1,d_1]$, senão $v(N)=[0,0]$.

A seguir é apresentado um exemplo de um PLE e a árvore e/ou gerada. O exemplo 1, Figura 15,é uma extensão do exemplo apresentado em [Blair & Subrahmanian, 1988].

Exemplo 1 Questionamento $p(b):[1.0,0.0]$ sobre a base de conhecimento a seguir.

$p(a):[1.0,0.0]$.

$p(X):[1.0,0.0]\Leftarrow q(X):[0.0,1.0]\wedge r(X):[1.0,0.0]$.

$r(a):[1.0,0.0]$.

$r(b):[1.0,0.0]$.

$q(a):[0.0,1.0]$.

$q(b):[0.0,1.0]$.

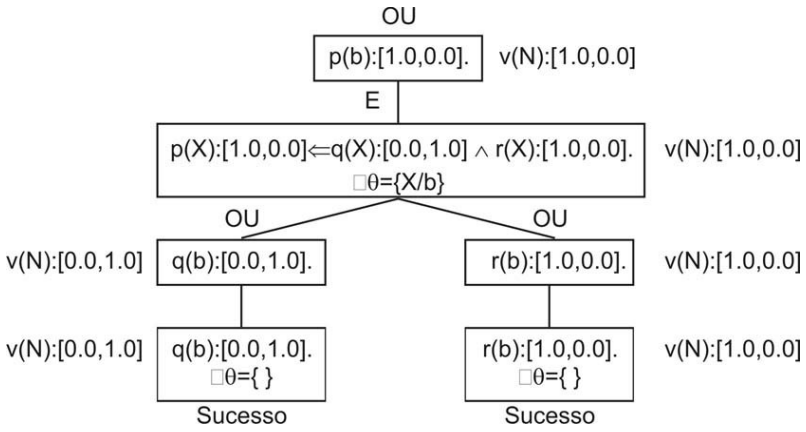


Figura 15 - Árvore do Exemplo 1

A árvore gerada para o exemplo 1 é trivial. Isso ocorre porque um PLE que possui apenas anotações perfeitamente definidas - $[1.0,0.0]$ e $[0.0,1.0]$- e não possui informações conflitantes, produz uma árvore e/ou equivalente gerada pela lógica de primeira ordem clássica.

Não é possível aplicar diretamente um procedimento de resolução padrão em PLE [Ávila, 1996]. Dessa forma, foi proposto em [Subrahmanian, 1987b] um procedimento de resolução chamado resolução-SLDe. Porém, este procedimento não pode ser aplicado diretamente a um PLE que não seja bem comportado [Ávila, 1996]. Devido à essa restrição, em [Blair & Subrahmanian, 1988] é proposto um outro procedimento chamado *fechamento.*

Definição 12 Um PLE é dito fechado se para quaisquer duas cláusulas-e C_1 e C_2 pertencentes a E da forma

$$p_1 : [c_1, d_1] \Leftarrow q_1 : [e_1, f_1] \wedge \ldots \wedge q_k : [e_k, f_k] \;\; k \geq 0$$
$$p_2 : [c_2, d_2] \Leftarrow q_1 : [g_1, h_1] \wedge \ldots \wedge q_l : [g_l, h_l] \;\; l \geq 0$$

tais que p_1 e p_2 são unificáveis – através de uma unificação mais geral – e $[c_1, d_1]$ e $[c_2, d_2]$ são não-comparáveis, há uma cláusula-e

$$p_1\theta : \sup\{[c_1, d_1], [c_2, d_2] \Leftarrow q_1 : [e_1, f_1] \wedge \ldots \wedge qk : [e_k, f_k] \wedge q_1 : [g_1, h_1] \wedge \ldots q_l : [g_l, h_l]\}$$

Definição 13 Seja E um PLE e

$A_1(E) = E$

$A_{n+1}(E) = A_n(E) \cup \{\lambda(C_1, C_2) | C_1, C_2 \in A_n(E)\} (n \geq 1)$

Para todo PLE E há um inteiro n tal que $A_n(E) = A_{n+1}(E)$. $A_n(E)$ chama-se fechamento de E e é denotado $CL(E)$.

Teorema 1 Se E é um PLE fechado, então:

1. I é um modelo de E se e somente se I é um modelo de $CL(E)$;
2. $T_E = T_{CL}(E)$.

Definição 14 Um PLE sobre T denomina-se inconsistente por negação se há alguma cláusula-e $p : [c, d]$, $p \in B_E$ tal que:

1. $T_E \uparrow w \models p : [c, d]$
2. $T_E \uparrow w \models p : [d, c]$

Definição 15 Um PLE é dito não-trivial se existe alguma cláusula-e $p : [c, d]$ tal que $T_E \uparrow w \not\models p : [c, d]$.

Definição 16 Um PLE E é dito *paraconsistente* se E for inconsistente por negação e não-trivial.

4.4. Técnicas de Gerenciamento de Incerteza e Mineração de Dados

Mesmo com o crescente uso e a diversidade de aplicações de algoritmos de aprendizagem de máquina, principalmente em mineração de dados, existe certa dificuldade no tratamento de informações incertas, o que conduz a utilização de técnicas que permitam interpretações adequadas onde são encontradas situações de incerteza, ou neste caso, inconsistência. Ferreira [Ferreira et al., 2004], propõem um método para tratar do problema de regras inconsistentes em mineração distribuída. O método considera que *n* subconjuntos de regras geradas são independentes para *n* subconjuntos de dados, o que pode resultar em regras inconsistentes, ou seja, regras com mesmo antecedente, porém com previsão de classes diferentes. As regras possuem o seguinte formato:

IF <antecedentes> **THEN** <conseqüente>

Onde:

<antecedentes>: formado por expressões condicionais envolvendo atributos do domínio da aplicação;

<conseqüente>: formado por uma expressão que indica a previsão do valor para o atributo meta, obtido em função dos valores encontrados nos atributos que compõem o antecedente.

A metodologia desenvolvida por Ferreira [Ferreira et al., 2004] utiliza Lógica Paraconsistente para determinar a regra mais adequada para classificar um novo exemplo. Cada regra pertencente ao primeiro subconjunto é submetida à comparação aos *n* subconjuntos de regras existentes.

O grau de crença da regra é obtido através da divisão do número de exemplos corretamente cobertos pela regra (onde os valores dos atributos do exemplo satisfazem às condições das *n* regras e possui o mesmo conseqüente),

pelo número total de exemplos cobertos (em que os valores dos atributos do exemplo satisfazem a condição da regra).

Por outro lado, o grau de descrença é obtido através da divisão dos exemplos incorretamente cobertos pela regra (onde os valores dos atributos do exemplo satisfazem às condições das *n* regras, mas possui conseqüente diferente), pelo número total de exemplos cobertos (em que os valores dos atributos do exemplo satisfazem a condição da regra).

Após estabelecer o grau de crença e descrença para cada regra, os subconjuntos são comparados para verificar quais regras possuem o mesmo antecedente. Quando o antecedente e o conseqüente das regras forem os mesmos, o algoritmo retorna o supremo[5], caso contrário, se as regras tiverem o mesmo antecedente, mas conseqüentes diferentes é escolhida a regra com maior valor resultante da multiplicação entre o grau de verdade e o grau de determinação obtida em cada regra.

Neste trabalho nós desenvolvemos um método capaz de integrar classificadores a base de regras utilizando princípios de Lógica Paraconsistente. Nós representamos regras como cláusulas Horn, tornando a interpretação da regras mais fiel ao modelo de lógica de predicados do que aquele utilizado por [Ferreira et al., 2004] utilizando raciocínio Lógico Evidencial para a tomada de decisão. A próxima seção descreve nossa metodologia.

[5] Supremo: máximo valor obtido na comparação entre os graus de crença e descrença de duas regras que são unidas se, e somente se, os conseqüentes das regras forem os mesmos.

Capítulo 5

Metodologia

Neste projeto foi desenvolvido um método de integração de modelos, baseado nos conceitos de Lógica Paraconsistente, que foi aplicado sobre os diversos *n* conjuntos de regras geradas sobre *n* subconjuntos de dados utilizados no treinamento. Nossa principal motivação foi tratar a inconsistência gerada com a união dos vários conjuntos de regras.

Para resolver o problema das regras inconsistentes foi utilizada a linguagem Paralog_e [Ávila, 1996] no desenvolvimento do método proposto. A linguagem Paralog_e é uma extensão da Linguagem de Programação Lógica ParaLog, desenvolvida por N. C. A. da Costa [da Costa et al., 1995]. A linguagem ParaLog permite associar somente uma anotação a uma proposição p. No Paralog_e [Ávila, 1996], é permitida a anotação de duas evidências, uma evidência favorável a *p* e uma evidência contrária a *p*, o que aumenta o poder expressivo da linguagem.

O motor de inferência do Paralog_e oferece uma semântica operacional para a linguagem implementada e sua execução está baseada no método de resolução-SLDe. Dessa forma, faz com que o motor de inferência simule uma *busca em profundidade* na árvore de refutação para as cláusulas do programa, ou seja, dado um programa *P* e um questionamento *Q,* o motor de inferência do Paralog_e fornece evidências como resposta aos questionamentos.

O método desenvolvido é capaz inferir sobre uma base de conhecimento, formada pela união de n subconjuntos de regras, em que as regras obtidas podem ser consideradas inconsistentes.

Duas ou mais regras são ditas inconsistentes se:

• Possuem os mesmos valores para os antecedentes (ou condições), mas os conseqüentes (ou conclusões) são diferentes;

• Possuem ao menos um antecedente com valores distintos, considerados contraditórios entre si, e que prevêem mesmo conseqüente.

Segundo Ávila [Ávila, 1996], diz-se que uma teoria é *consistente* se não possuir teoremas contraditórios, um dos quais, a negação do outro. Caso contrário, a teoria diz-se *inconsistente (ou contraditória).* Inconsistências, portanto, surgem naturalmente na descrição do mundo real e podem ocorrer em diversos contextos, como por exemplo, durante a integração de bases de conhecimento geradas de maneira distribuída.

A execução de algoritmos de aprendizagem de máquina sobre bases de dados inconsistentes tende a gerar conceitos inconsistentes. Dessa maneira os conceitos obtidos a partir de fontes distribuídas de dados, podem conter conceitos que quando avaliados em um único conjunto podem ser considerados inconsistentes.

Basicamente existem duas formas para tratar a existência da inconsistência:

• atribuir ao algoritmo de aprendizado a habilidade de manipular adequadamente as informações contraditórias durante o processo de aprendizado, com a finalidade de gerar conceitos consistentes e confiáveis;

• aplicar sobre o conhecimento obtido, através de um algoritmo de aprendizado qualquer, um método de raciocínio que possibilite a inferência confiável de informações: transformação da base de conhecimento de modo a torná-la consistente, gerar outra base de conhecimento a partir dos dados consistentes de conhecimento existente na base inicial, aplicar técnicas de gerenciamento de incerteza que possibilitem o raciocínio sobre conceitos inconsistentes, entre outras.

Técnicas de gerenciamento de incerteza e Lógica Paraconsistente podem ser utilizadas com o objetivo de medir a inconsistência de cada regra, associando a elas fatores evidenciais que permitem avaliar o seu grau de inconsistência a partir do subconjunto de dados utilizado para a geração da regra e, posteriormente, o grau de inconsistência obtido com a união dos subconjuntos em um único modelo.

A utilização de sistemas lógicos que permitam a manipulação de informações inconsistentes é uma área de importância crescente em ciência da computação, teoria de banco de dados e inteligência artificial.

Uma representação gráfica da metodologia a ser utilizada pode ser observada na Figura 16, a seguir. Em resumo, as etapas a serem aplicadas serão:

• extrair os *n* conjuntos de regras de *n* base de dados;

• converter as regras para o formato de entrada do Paralog_e;

• atualizar os graus de crença (c) e descrença (d) das regras, de acordo com a importância da regra no subconjunto;

• combinar os *n* conjuntos de regras em um único conjunto;

• ordenar as regras contidas no conjunto único em função da verdade;

• submeter novos exemplos a serem classificados.

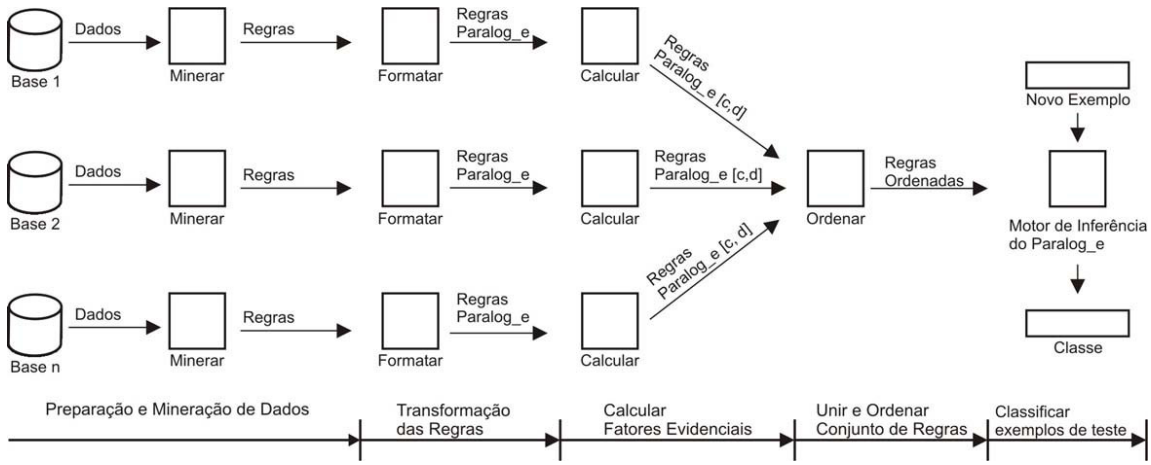


Figura 16 -Representação Gráfica da Metodologia a ser utilizada

Desta forma, a motivação para este projeto foi o desenvolvimento de um método capaz de tratar regras inconsistentes, com mesmo conseqüente, porém com atributos antecedentes contraditórios. Isso pode ocorrer porque as regras são geradas a partir de bases de dados distribuídas que contém exemplos diferentes e variações de distribuição. Diferentemente dos mecanismos discutidos na seção 3.2.7, o método desenvolvido neste trabalho não calcula os fatores de qualidade de uma regra a partir da avaliação em vários (possivelmente todos) os

subconjuntos de dados utilizados no treinamento colocando todas as regras obtidas em um único conjunto. As medidas utilizadas para o cálculo dos fatores evidenciais, são, portanto, locais ao conjunto de treinamento que originou a regra. Este diferencial, evita o alto fluxo de comunicação de mensagens entre os subconjuntos e reduz significativamente o tempo necessário para processamento em relação a outros métodos de mineração distribuída de dados.

Como mencionado anteriormente, os dados distribuídos foram submetidos a estas etapas:

i) dividir a base de dados em conjunto de treinamento e conjunto de testes;

ii) gerar o conjunto de regras para cada subconjunto de dados local. Para obter os conjuntos de regras foi utilizado o algoritmo de indução RIPPER [Cohen, 1995], que se encontra implementado e disponível para aplicação na plataforma WEKA [Frank et al, 2000], ambiente desenvolvido para aplicação de algoritmos de aprendizagem de máquina e descoberta de conhecimento.

O interesse em optar por algoritmos de indução de regras é que estes algoritmos de aprendizagem tendem a formar modelos altamente competitivos além de que a linguagem de representação simbólica do tipo *if* <antecedentes> *then* <conseqüente> permitir fácil compreensão. A escolha do algoritmo RIPPER é justificada pela necessidade de assegurar que o algoritmo de indução fosse capaz de produzir conjuntos de regras ordenadas com sobreposição, ou seja, mais de uma regra pode cobrir o mesmo exemplo. RIPPER é menos sensível a variabilidade dos dados de entrada do que o C4.5 [Quinlan, 1993], tem bom desempenho em domínios ruidosos, faz uso da redução incremental do erro através da poda e da análise do conjunto inicial das regras formadas, preservando as regras que possuem a medida com menor tamanho de descrição e eliminando do conjunto as demais.

iii) combinar os múltiplos conjuntos de regras locais em um único conjunto de regras global. O método proposto é aplicado e então avaliado, permitindo classificar exemplos utilizando o conjunto de regras que foram unidas. Essas regras foram anotadas com os fatores evidenciais (grau de crença e descrença)

calculados como sendo a distribuição linear como medida de crença, e o complemento da distribuição linear como descrença. O método de combinação de regras proposto deverá ser realizado em outras etapas: transformação das regras para o formato Paralog_e, atualização dos fatores evidenciais das regras em Paralog_e, união dos subconjuntos de regras, ordenação das regras e utilização do mecanismo de inferência existente no Paralog_e para a classificação de novos exemplos, devidamente transformados em fatos. Todas essas etapas são descritas em detalhes nas próximas seções.

5.1. Preparação e Mineração dos Dados

Com o objetivo de simular um ambiente distribuído, cada base de dados B foi dividida distribuindo-se aleatoriamente os exemplos para compor o conjunto de treinamento e o conjunto de testes.

- 20% dos exemplos existentes na base compõem o conjunto de testes;
- 80% dos exemplos existentes na base compõem o conjunto de treinamento;

Na Figura 17 é ilustrado em detalhes o processo de divisão e mineração.

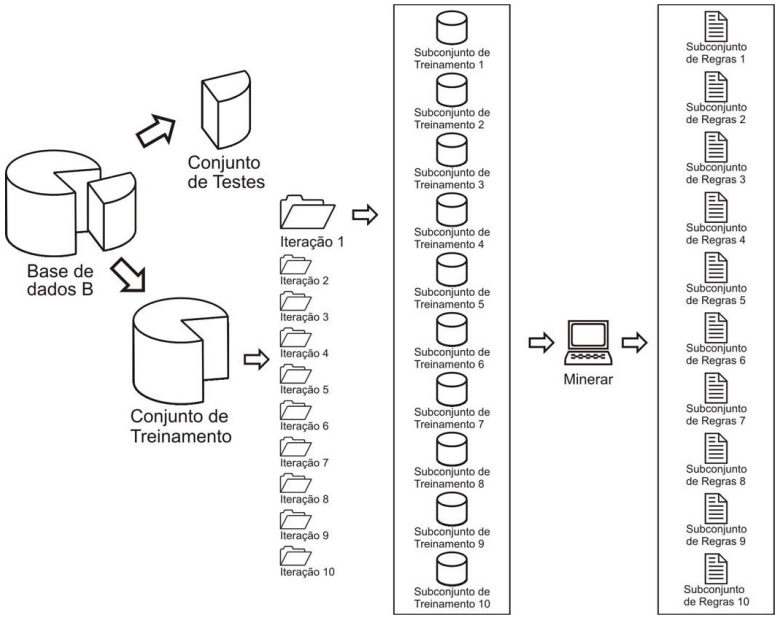


Figura 17 - Preparação e Mineração da base de dados

Foram geradas dez partições para a base B sendo que em cada partição foi formada por dez subconjuntos de treinamento. Cada subconjunto de treinamento utiliza 8% dos exemplos existentes no conjunto de treinamento, selecionados a partir de um processo de validação cruzada. Esses percentuais foram escolhidos porque acreditamos que os modelos gerados a partir de pequenas partições do conjunto têm maior chance de gerar regras inconsistentes, pois o espaço de tuplas utilizado é reduzido, diminuindo a possibilidade de interseção entre os modelos.

Os subconjuntos foram submetidos ao treinamento utilizando como base o algoritmo minerador RIPPER [Cohen, 1995], que está implementado e disponível no ambiente WEKA [Frank et al, 2000]. O conjunto de testes foi utilizado para medir o desempenho do algoritmo sobre cada um dos subconjuntos de treinamento.

O método desenvolvido utiliza os dez subconjuntos de regras geradas em cada partição e os une em um único conjunto. Os exemplos do conjunto de teste são submetidos então à avaliação. O processo é iterativo e o mesmo procedimento é realizado repetidamente dez vezes para se obter uma certa significância estatística.

5.2. Transformação de regras de classificação em regras Paralog_e

As regras descobertas a partir dos subconjuntos de dados foram transformadas em premissas lógicas anotadas Paralog_e. O Paralog_e [Ávila, 1996] utiliza conceitos de programação lógica evidencial e permite inferir a partir de regras anotadas. Cada regra está associada a um grau de crença e outro de descrença.

As regras foram transformadas em cláusulas evidenciais, conforme demonstrado através do algoritmo presente na Figura 18.

```
Dado: conjunto de n Regras r
Regra r: (nome_do_atributo1 operador1 valor1) and ...
         (nome_do_atributon operadorn valorn) => class=nome_da_classe(c1/d1)
início
LER conjunto de n Regras r
ENQUANTO (arquivo < > ø) FAÇA
 PARA Regrai até Regran FAÇA
    CONSTRUA a Regrai no formato
          class('nome_da_classe'):[c1,d1]←
             avaliador(nome_do_atributo1,Variavel1):[1.0,0.0] &
             variavel1 operador1 valor1 &...
             ...avaliador(nome_do_atributon,Variaveln):[1.0,0.0] &
             variaveln operadorn valorn.
    LER próxima Regra
 FIM-PARA
FIM-ENQUANTO
fim
```

Figura 18 - Algoritmo para transformar regras em cláusulas evidenciais

Como citado anteriormente uma regra é composta por $Cabeça \leftarrow Corpo$ e é construída atribuindo-se um valor de classe à $Cabeça$ e heuristicamente é construído o $Corpo$ da regra.

As regras obtidas são transformadas em cláusulas evidenciais, em que a $Cabeça$ é formada por $\text{class}('\text{nome\_da\_classe}'):[c_1, d_1]$, onde:

$\text{class}('\text{nome\_da\_classe}'):[c_1, d_1]$ representa a conclusão, ou seja, a classe que está associada a regra em questão e $[c_1, d_1]$ denotam os fatores evidenciais de crença e descrença que estão associados à regra, respectivamente;

O $Corpo$ é composto por uma conjunção de condições. A conjunção é denotada pelo símbolo $\&$. O conjunto de conjunções formam as condições que compõem a regra. Cada condição da regra é formatada em um predicado $avaliador$ que possibilita posteriormente avaliar se as condições das regras são

verdadeiras quando exemplos de teste são submetidos à inferência. Cada condição da regra é transformada no seguinte formato:

$$avaliador(nome\_do\_atributo, \text{var}iavel) : [1.0,0.0] \& \ \text{var}iavel \ operador \ valor$$

Para ilustrar o procedimento de transformação de regras no formato Paralog_e, foi criado a seguinte regra no formato RIPPER [Cohen, 1995]:

Exemplo de Formatação de regras em regras anotadas

```
(producaolacrimal = normal) and (astigmatismo = sim) and
(espectropia = hipermetropia) => class=nenhuma (2.0/1.0)
```

Ao submeter a regra à transformação obtém-se a mesma regra na seguinte formatação:

```
class('nenhuma'):[1.000000,0.000000] <--
     avaliador(producaolacrimal,V_0):[1.0,0.0] &
     V_0 = normal &
     avaliador(astigmatismo,V_1):[1.0,0.0] &
     V_1 = sim &
     avaliador(espectropia,V_2):[1.0,0.0] &
     V_2 = hipermetropia.
```

Para exemplificar todas as etapas, é apresentado detalhadamente a seguir cada processo da metodologia utilizada.

Na Figura 19 ilustramos a extração das regras obtidas por um algoritmo de indução de regras.

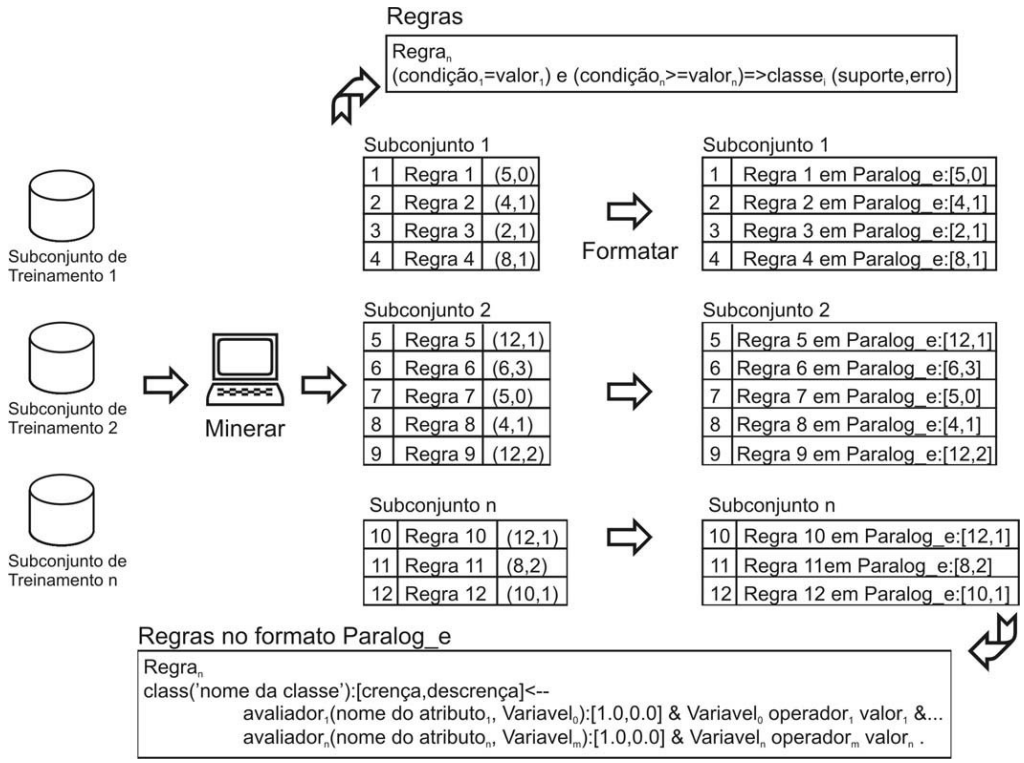


Figura 19 - Etapa de extração e transformação das regras em regras Paralog_e

Após a extração de regras, através do algoritmo de indução presente na plataforma WEKA [Frank et al, 2000], os *n* subconjuntos de regras obtidos são formatados em cláusulas evidenciais, que denominamos de *regras em Paralog_e*. Os valores dos fatores evidenciais anotados para cada regra, grau de crença e descrença, são atualizados na etapa seguinte da metodologia.

5.3. Atualização dos fatores evidenciais nas regras em Paralog_e

Os valores dos fatores evidenciais associados às cláusulas, que neste caso denominamos de regras em Paralog_e, são atualizados a partir da distribuição linear da regra na forma de uma progressão aritmética. Essa atualização é realizada localmente, ou seja, antes da integração dos conjuntos de regras. O fator evidencial favorável (c), ou seja, o grau de crença é atualizado com o valor obtido através da distribuição linear do subconjunto de regras. O fator evidencial

desfavorável (d), ou seja, o grau de descrença, é o complemento da distribuição linear do subconjunto de regras $(d = 1 - c)$. Para se obter a razão da progressão aritmética é necessário aplicar a seguinte divisão:

$$r = \frac{1}{numero\_de\_regras\_no\_subconjunto}$$

O valor de crença atribuídos à regra *i* corresponde à progressão aritmética de *i* por *r* iniciando da última regra para a primeira. Um exemplo pode ser visto na Figura 20. A última regra sempre possui *c = r* e a primeira regra sempre possui *c = 1,0*.

Diferentes estratégias heurísticas podem ser adotadas para a produção de regras, como visto em capítulos anteriores. Algumas categorias de algoritmos de aprendizado simbólico iniciam a construção pela regra mais geral e gradativamente a especializam. A especialização é finalizada quando a regra não cobre mais exemplos negativos. A tendência, neste caso, é que a regra mais geral (a primeira regra gerada) possua maior cobertura do que as demais regras que a sucedem, e que resultaram de um processo de refinamento, dessa forma definindo uma prioridade entre as regras.

O método desenvolvido leva em consideração o grau de importância da regra em relação ao subconjunto de regras em que está presente. A última regra do subconjunto é considerada a menos significativa, porque foi a regra mais especializada. Desta forma, em conjuntos ordenados de regras a regra mais importante é a primeira, depois a segunda e assim sucessivamente.

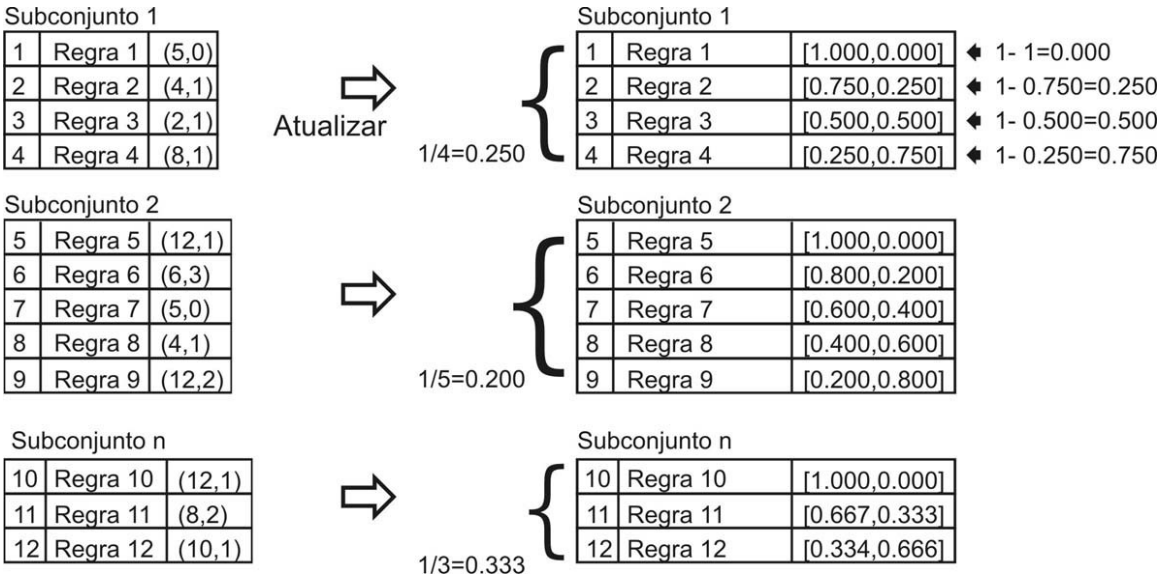


Figura 20 - Atualização dos fatores evidenciais nas regras em Paralog_e

5.4. União dos subconjuntos de regras

Após a atualização dos fatores evidenciais nas regras em Paralog_e, os subconjuntos são unificados em um único conjunto de regras em Paralog_e. Na Figura 21 é possível observar esta etapa de união das regras.

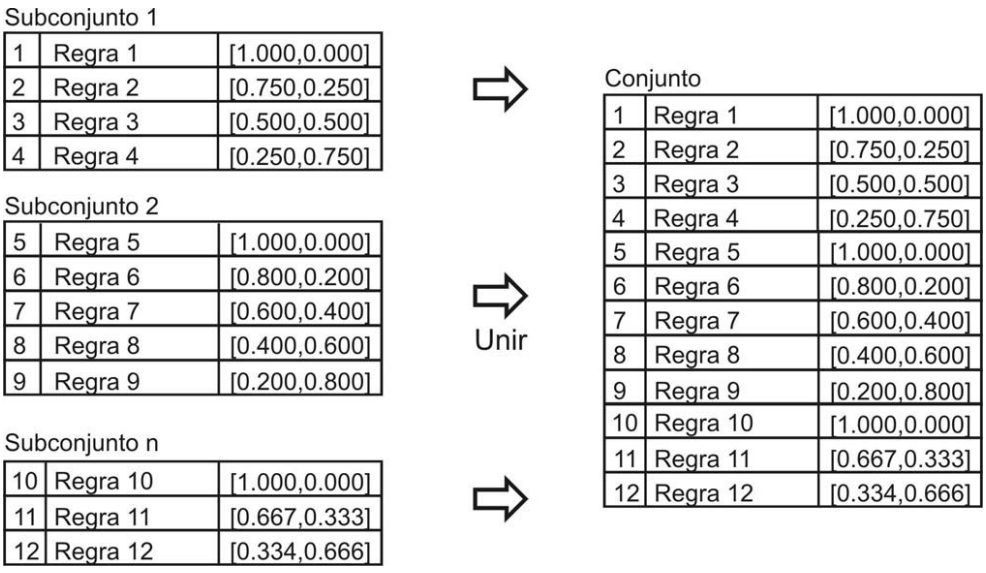


Figura 21 - União das regras em Paralog_e em um único conjunto

5.5. Ordenação das regras Paralog_e

Após a união dos subconjuntos em um único conjunto de regras, as regras são então ordenadas. As regras são ordenadas de acordo com a menor distância euclidiana em relação aos fatores evidenciais [1,0] que representam a verdade (v).

Os fatores evidenciais anotados que são associados às regras estão compreendidos no intervalo de valores entre 0 e 1. Estes valores indicam respectivamente o grau de crença e descrença associados à regra, e definem o estado lógico ao qual a regra está relacionada. Desta forma é possível representar todas as regras do conjunto através de um reticulado que é representado por meio do Diagrama de Hasse.

A medida de distância euclidiana é uma das medidas de similaridade mais freqüentemente utilizadas, e neste projeto significa que quanto menor o valor obtido entre a regra e o estado lógico que representa a verdade, mais próxima a regra está da verdade em termos quantitativos.

Com esta representação, é possível analisar todas as regras existentes do conjunto em relação às coordenadas que representam o valor máximo da verdade, ordenando as regras do conjunto de forma crescente em função da verdade.

Em Lógica Paraconsistente, os valores dos graus de crença e descrença são independentes entre si, ou seja, são valores não complementares que individualmente quantificam o quanto o conceito está apoiado na evidência verdadeira e paralelamente, quanto representa que o mesmo conceito pode ser considerado falso. Uma vez que os valores de crença e descrença calculados até então são complementares, nós não os utilizamos diretamente para se calcular a distância com a verdade. A partir da crença e descrença são obtidos $valor_1$ e $valor_2$ da seguinte forma:

- O valor$_1$ é denominado de Grau de Certeza, é calculado como o grau de crença diminuído do grau de descrença da regra, ou seja, $(valor_1 = c - d)$;
- O valor$_2$ é calculado pela multiplicação do grau de descrença da regra por 2. $(valor_2 = 2 \times d)$

Para cada regra existente no conjunto são calculados os valores $valor_1$ e $valor_2$. Estes valores são utilizados para calcular o quanto cada regra está distante do estado lógico da verdade. O grau de certeza (*valor$_1$*), como descrito na seção 4.2.2, é um fator importante e mede quanto de certeza está associado à verdade da proposição. Por outro lado, nós empiricamente definimos *valor$_2$* como o dobro da descrença associada à regra, por acreditar que toda regra produziria um erro maior (portanto, uma maior descrença) caso fosse avaliada sobre conjuntos de dados diferentes. Como não há indícios de quanto seria essa ampliação o dobro da descrença foi utilizado.

A ordem das regras será de forma crescente a partir da regra com a menor distância euclidiana obtida em relação ao par ordenado [1,0] – conforme Equação 1. Desta forma, a ordenação possibilita que o conjunto de regras esteja ordenado a partir da regra considerada mais próxima do estágio lógico da verdade.

Equação 1

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Conforme detalhado, a distância euclidiana é então aplicada a partir dos valores $valor_1$ e $valor_2$, da seguinte forma:

$$d(1,0) = \sqrt{((1 - valor_1)^2 + (0 - valor_2)^2)}$$

Na Figura 22 esta etapa é apresentada de forma detalhada.

Distância Euclidiana

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Conjunto

1	Regra 1	[1.000,0.000]
2	Regra 2	[0.750,0.250]
3	Regra 3	[0.500,0.500]
4	Regra 4	[0.250,0.750]
5	Regra 5	[1.000,0.000]
6	Regra 6	[0.800,0.200]
7	Regra 7	[0.600,0.400]
8	Regra 8	[0.400,0.600]
9	Regra 9	[0.200,0.800]
10	Regra 10	[1.000,0.000]
11	Regra 11	[0.667,0.333]
12	Regra 12	[0.334,0.666]

⇨ Ordenar

Conjunto

	Regra	Crença	Descrença	Certeza	2 x Descrença	Ordem
1	Regra 1	1,000	0,000	1,00	0,00	1
2	Regra 2	0,750	0,250	0,50	0,50	5
3	Regra 3	0,500	0,500	0,00	1,00	8
4	Regra 4	0,250	0,750	-0,50	1,50	11
5	Regra 5	1,000	0,000	1,00	0,00	2
6	Regra 6	0,800	0,200	0,60	0,40	4
7	Regra 7	0,600	0,400	0,20	0,80	7
8	Regra 8	0,400	0,600	-0,20	1,20	9
9	Regra 9	0,200	0,800	-0,60	1,60	12
10	Regra 10	1,000	0,000	1,00	0,00	3
11	Regra 11	0,667	0,333	0,33	0,67	6
12	Regra 12	0,334	0,666	-0,33	1,33	10

Exemplo do critério de ordenação

	Regra	Crença	Descrença	Certeza	2 x Descrença	Distância	Ordem
7	Regra 7	0,600	0,400	0,20	0,80	1,13137085	7

Distância Euclidiana em função do par ordenado [1,0]

$$d(1,0) = \sqrt{((1-0.20)^2 + (0-0.80)^2)} = 1.1313$$

Figura 22 - Ordenação das regras em Paralog_e

5.6. Classificação de exemplos de teste

Após ordenar as regras em ordem crescente em relação a menor distância euclidiana obtida, os exemplos de teste são submetidos ao conjunto de regras ordenadas para a classificação.

Os exemplos de teste que estão no formato *arff* definem o formato de entrada de dados no ambiente WEKA [Frank et al, 2000]. Todos os exemplos de teste contidos no conjunto de testes são formatados em fatos. Na seqüência é observado como esta etapa é realizada. Na Figura 23 é demonstrado como é o formato *arff* que define o formato de entrada de dados do ambiente WEKA.

Na Figura 23 foi definido um exemplo que compõe um conjunto de testes hipotético, criado para ilustrar a metodologia.

O exemplo foi transformado em fatos, conforme demonstração na Figura 24.

```
@RELATION Lentes

@ATTRIBUTE idade {jovem,prepresbiotico,presbiotico}
@ATTRIBUTE espectropia {miopia,hipermetropia}
@ATTRIBUTE astigmatismo {sim,nao}
@ATTRIBUTE producaolacrimal {reduzido, normal}
@ATTRIBUTE class {nenhuma,macia,dura}

@DATA
jovem,miopia,nao,reduzido,nenhuma
```

Figura 23 - Formato entrada de dados arff

```
valor(idade,jovem):[1.0,0.0].
valor(espectropia,miopia):[1.0,0.0].
valor(astigmatismo,nao):[1.0,0.0].
valor(producaolacrimal,reduzido):[1.0,0.0].
valor(class,nenhuma):[1.0,0.0].
```

Figura 24 - Formato entrada de dados Paralog_e

Cada exemplo de teste, denotado como um conjunto de fatos, é submetido à prova a partir da base de conhecimento formada pelo conjunto de regras, que anteriormente foram transformadas em cláusulas de Horn [Casanova et al., 1987], ou regras Paralog_e, assim definido neste projeto. A representação das regras em cláusulas de Horn permite verificar se as condições das regras são verdadeiras em relação ao exemplo de teste fornecido.

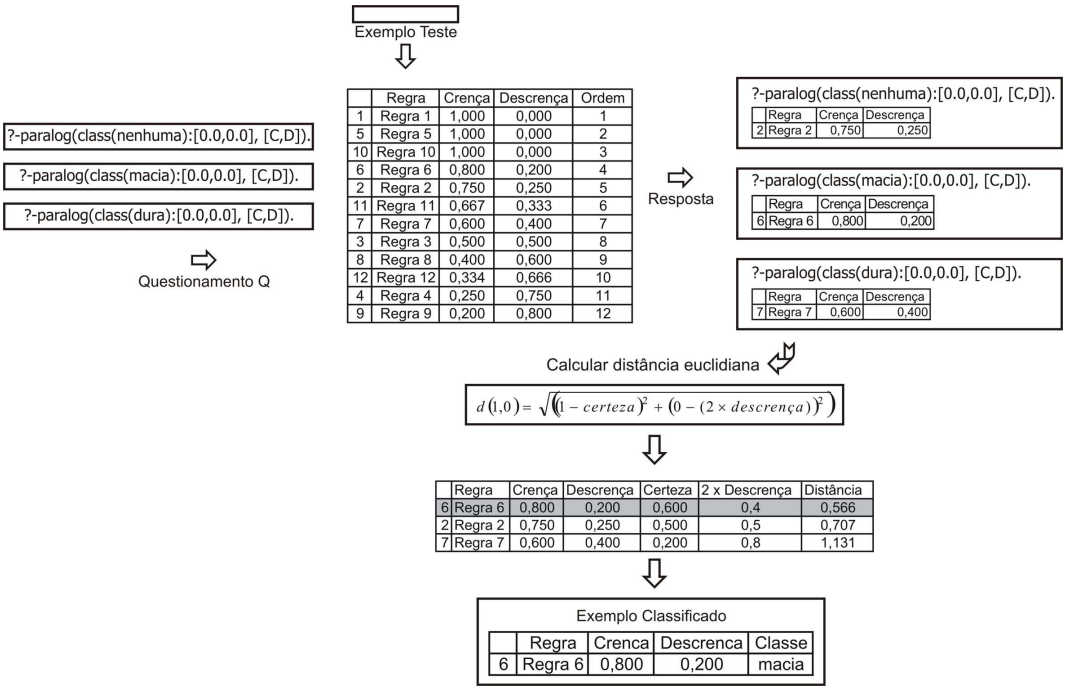
A etapa do questionamento é realizada fornecendo como entrada os valores de todos os atributos, inclusive o valor do atributo-alvo (classe associada ao exemplo) que é então avaliado sob a base de conhecimento formada pelo conjunto de regras em Paralog_e.

O questionamento é feito ao motor de inferência do Paralog_e que fornece evidências como respostas aos questionamentos.

Em seguida, é feito um questionamento Q para cada classe pertencente ao domínio da base de dados. A resposta do questionamento é dada por

$Q = C : \sup\{E\}$, tal que $C : \sup\{E\}$ representa o supremo obtido a partir do subconjunto de regras E que possuem a classe C. O supremo representa o questionamento Q, e retorna a *cabeça* da regra, cuja classe foi objeto de questionamento, com os fatores evidenciais (crença e descrença) máximos da regra em que as condições satisfazem os valores dos atributos do exemplo de teste (transformado em fatos).

Através dos valores supremo obtidos para cada classe pertencente ao domínio da base, é escolhido para classificar o exemplo de teste a classe cujos fatores evidenciais possuem a menor distância euclidiana em relação ao valor verdade, representado no reticulado pelos fatores evidenciais [1,0], ver Figura 25. A fórmula da distância euclidiana foi detalhada anteriormente na seção 5.5, através da Equação 1.



	Regra	Crença	Descrença	Ordem
1	Regra 1	1,000	0,000	1
5	Regra 5	1,000	0,000	2
10	Regra 10	1,000	0,000	3
6	Regra 6	0,800	0,200	4
2	Regra 2	0,750	0,250	5
11	Regra 11	0,667	0,333	6
7	Regra 7	0,600	0,400	7
3	Regra 3	0,500	0,500	8
8	Regra 8	0,400	0,600	9
12	Regra 12	0,334	0,666	10
4	Regra 4	0,250	0,750	11
9	Regra 9	0,200	0,800	12

	Regra	Crença	Descrença
2	Regra 2	0,750	0,250

	Regra	Crença	Descrença
6	Regra 6	0,800	0,200

	Regra	Crença	Descrença
7	Regra 7	0,600	0,400

	Regra	Crença	Descrença	Certeza	2 x Descrença	Distância
6	Regra 6	0,800	0,200	0,600	0,4	0,566
2	Regra 2	0,750	0,250	0,500	0,5	0,707
7	Regra 7	0,600	0,400	0,200	0,8	1,131

	Regra	Crenca	Descrenca	Classe
6	Regra 6	0,800	0,200	macia

Figura 25 - Classificação de exemplos de teste

Para avaliar se houve acerto na classificação, a classe eleita para classificar o exemplo de teste, ou seja, a resposta do questionamento para a

classe que possui a menor distância euclidiana em relação ao estado lógico da verdade, é comparada com a classe que rotula o exemplo de teste. Se a classe eleita for a mesma que rotula o exemplo de teste, houve acerto na classificação.

Capítulo 6

Experimentos e Resultados

Os experimentos foram aplicados em diversas bases de dados, com os mais diversos conteúdos de informação e quantidade de atributos existentes, bem como o número de classes presente.

As bases foram divididas, distribuindo-se aleatoriamente os exemplos para compor o conjunto de treinamento e o conjunto de testes.

- 20% dos exemplos existentes na base formaram o conjunto de testes;
- 80% dos exemplos da base compuseram o conjunto de treinamento;

Nestas proporções, cada base utilizada foi dividida em conjunto de treinamento e conjunto de testes. O conjunto de treinamento foi dividido em dez iterações, sendo que cada iteração foi formada por dez subconjuntos de treinamento, em que os exemplos foram selecionados de forma aleatória.

O método desenvolvido utiliza os dez subconjuntos de regras geradas em cada iteração e os une em um único conjunto. Os exemplos do conjunto de teste foram submetidos à avaliação do conjunto de regras anteriormente unificado, conforme descrito anteriormente. O processo é iterativo e o mesmo procedimento foi aplicado sob as dez iterações formadas, conforme detalhado nas seções anteriores.

6.1. Análise dos Resultados em Bases Específicas

Nos experimentos foram utilizadas bases de dados públicas, pertencentes ao diretório de bases para aprendizagem de máquina e mineração de dados, mantido pela Universidade da Califórnia [Blake et al., 1998]. As características de cada base utilizada, bem como o número de exemplos presente, a existência ou não de valores faltantes nos exemplos que compõem a base entre outras características foram detalhadas na Tabela 7.

	Base	Número de Exemplos	Valores Faltantes	Total de Atributos	Atributos Nominais	Atributos Numéricos	Total de Classes	Distribuição das Classes
1	Zoo	101	não	17	16	1	7	desbalanceada
2	Audiology	226	sim	69	69	–	24	desbalanceada
3	Monk 1	432	não	6	6	–	2	desbalanceada
4	Monk 2	432	não	6	6	–	2	desbalanceada
5	Soybean	683	sim	35	35	–	19	desbalanceada
6	Vehicle	846	não	18	–	18	4	desbalanceada
7	Tic-Tac-Toe	958	não	9	9	–	2	desbalanceada
8	Vowel	990	não	13	3	10	11	balanceada
9	Car	1728	não	6	6	–	4	desbalanceada
10	Segment	2310	não	19	–	19	7	balanceada

Tabela 7 - Características das bases de utilizadas nos experimentos [Blake et al., 1998]

Na Tabela 7, a coluna “Total de Atributos” não inclui o atributo-meta. Na mesma Tabela 7, a distribuição das classes dita desbalanceada caracteriza a distribuição não uniforme dos exemplos em função das classes.

O método desenvolvido neste trabalho, que faz uso da linguagem Paralog_e na implementação, é comparado com a taxa média de acerto obtida em cada iteração através dos dez subconjuntos, em que foi utilizado o algoritmo RIPPER [Cohen, 1995] para formar os classificadores. Os resultados obtidos são apresentados nas tabelas a seguir (Tabelas 8 a 17).

Nos resultados específicos, obtidos para cada base de dados, em cada iteração a taxa de acerto obtida pelo método Paralog_e é comparada com as taxas de acerto obtidas pelo algoritmo RIPPER (do primeiro ao décimo subconjunto de regras).

Também nas tabelas com os resultados específicos, tabelas de 8 a 17, a média geral dos resultados obtidos a partir do método Paralog_e é comparada com a média geral dos resultados obtidos entre as dez iterações.

Após a notação da média obtida, seja a partir de cada iteração ou o valor médio geral obtido em todas as iterações, é apresentado também o valor calculado do desvio padrão.

Na avaliação dos resultados é considerado que um resultado i é estatisticamente melhor que um resultado j, se a média e o desvio padrão de i tem valor superior a média e o desvio padrão de j. Os melhores resultados obtidos estão destacados em negrito.

Zoo		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	**57,14**	38,09	38,09	38,09	**57,14**	**57,14**	38,09	38,09	38,09	38,09	**57,14**	43,81±9,20
2	**61,90**	42,85	42,85	42,85	42,85	42,85	42,85	**61,90**	**61,90**	**61,90**	**61,90**	50,47±9,84
3	**61,90**	42,85	42,85	**61,90**	42,85	42,85	**61,90**	**61,90**	**61,90**	42,85	42,85	50,47±9,84
4	**57,14**	38,09	38,09	38,09	38,09	38,09	38,09	**57,14**	**57,14**	**57,14**	**57,14**	45,71±9,84
5	**57,14**	38,09	38,09	38,09	38,09	38,09	38,09	38,09	**57,14**	**57,14**	**57,14**	43,81±9,20
6	**57,14**	38,09	38,09	38,09	38,09	38,09	38,09	**57,14**	**57,14**	**57,14**	**57,14**	45,71±9,84
7	**61,90**	47,85	42,85	**61,90**	**61,90**	**61,90**	**61,90**	42,85	42,85	42,85	42,85	50,97±9,53
8	**61,90**	42,85	42,85	**61,90**	**61,90**	**61,90**	**61,90**	**61,90**	**61,90**	42,85	42,85	54,28±9,84
9	**71,42**	42,85	42,85	61,90	61,90	61,90	61,90	61,90	61,90	52,38	42,85	55,23±9,04
10	**66,67**	42,85	42,85	61,90	61,90	42,85	42,85	42,85	47,61	61,90	61,90	50,95±9,54
Média Geral	**61,43±4,73**											49,14±4,14

Tabela 8 - Resultados obtidos para a base Zoo

Nos resultados para a base de dados Zoo, contidos na Tabela 8, foi observado que a média geral de 61,43% e desvio padrão 4,73 obtida com o método Paralog_e foi superior aos valores obtidos pelo algoritmo RIPPER nas dez iterações (média geral de 49,14% e desvio padrão de 4,14).

Ao analisar os resultados obtidos em cada iteração da base de dados Zoo, é possível notar que nas primeiras oito iterações, os resultados alcançados com o método Paralog_e têm a mesma taxa de acerto que o melhor resultado conseguido pelo algoritmo RIPPER entre os dez subconjuntos avaliados. Em todas as iterações, os resultados do método Paralog_e foram mais competitivos que a média dos resultados dos subconjuntos.

Na Tabela 9, são apresentados os resultados para a base de dados Audiology.

Ao avaliar a média geral, o método Paralog_e obteve resultado superior (Paralog_e com média de taxa de acerto de 48,89%) a média geral obtida entre as iterações (RIPPER com média de taxa de acerto de 32,65%).

Entretanto se considerar os resultados individualmente em cada iteração, em seis das dez iterações analisadas, a maior taxa de acerto entre os dez subconjuntos foi superior ao resultado no método Paralog_e. Por exemplo, na iteração quatro, o resultado obtido no subconjunto dez, obteve 60,86% de acerto utilizando o algoritmo RIPPER, e que foi superior a taxa de acerto do método Paralog_e, 47,82%. Mas se considerar os valores obtidos pelo Paralog_e e a média dos resultados dos subconjuntos nas dez partições, o método Paralog_e em todas as iterações teve taxas de acerto superiores.

Audiology		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	**50,00**	26,08	26,08	32,60	17,39	26,08	36,95	43,47	32,60	43,47	34,78	31,95±8,26
2	47,82	28,26	43,47	43,47	**56,52**	32,60	45,65	26,08	26,08	30,43	36,95	36,95±10,09
3	**52,17**	45,65	34,78	45,65	26,08	36,95	30,43	23,91	30,43	23,91	41,90	33,97±8,40
4	47,82	45,65	54,34	26,08	32,60	26,08	43,47	56,52	28,26	41,30	**60,86**	41,52±12,98
5	45,65	23,91	23,91	23,91	23,91	23,91	28,26	45,65	26,08	45,65	**56,52**	32,17±12,25
6	47,82	30,43	32,60	23,91	23,91	23,91	23,91	21,73	43,47	**58,69**	39,13	32,17±11,85
7	47,82	**52,17**	23,91	23,91	23,91	23,91	23,91	23,91	45,65	28,26	23,91	29,35±10,51
8	**52,17**	36,95	23,91	23,91	23,91	23,91	28,26	43,47	30,43	28,26	32,60	29,56±6,58
9	**52,17**	30,43	34,78	45,65	23,91	34,78	34,78	23,91	23,91	28,26	41,30	32,17±7,52
10	43,47	41,30	23,91	13,04	34,78	**45,65**	28,26	23,91	30,43	23,91	21,91	28,71±9,70
Média Geral	**48,69±2,94**											32,65±3,88

Tabela 9 - Resultados obtidos para a base Audiology

Na base de dados Monk1, conforme pode ser observado na Tabela 10, ao compararmos a média geral o método Paralog_e obteve taxa de acerto inferior (55,23%) a média geral obtida entre as iterações (55,50%) utilizando o algoritmo de indução de regras RIPPER.

Monk1		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	66,67	50,00	**75,00**	58,33	58,33	**75,00**	61,11	61,11	50,00	50,00	**75,00**	61,39±10,35
2	56,15	50,00	50,00	**75,00**	66,67	**75,00**	**75,00**	**75,00**	50,00	50,00	50,00	61,67±12,55
3	50,00	50,00	50,00	50,00	**58,33**	50,00	50,00	50,00	50,00	50,00	50,00	50,83±2,63
4	59,17	50,00	50,00	50,00	50,00	50,00	50,00	**75,00**	55,50	50,00	50,00	53,05±7,90
5	50,00	50,00	50,00	50,00	50,00	50,00	50,00	50,00	**58,33**	50,00	50,00	50,83±2,63
6	50,00	50,00	50,00	50,00	50,00	**75,00**	**75,00**	50,00	50,00	**75,00**	50,00	57,50±12,08
7	52,78	50,00	50,00	58,33	50,00	50,00	50,00	50,00	50,00	50,00	**66,67**	52,50±5,63
8	59,24	**75,00**	50,00	61,11	55,55	50,00	50,00	50,00	50,00	50,00	**75,00**	56,67±10,33
9	50,00	**58,33**	50,00	50,00	50,00	50,00	**58,33**	50,00	50,00	50,00	50,00	51,67±3,51
10	58,33	**75,00**	50,00	**75,00**	50,00	50,00	50,00	50,00	**75,00**	55,55	58,33	58,81±11,48
Média Geral	55,23±5,66											**55,50±4,25**

Tabela 10 - Resultados obtidos para a base Monk1

Ao avaliar os resultados em cada iteração, o melhor resultado obtido entre os subconjuntos é superior a taxa de acerto do método Paralog_e. Em seis das dez iterações a média obtida através dos subconjuntos utilizando o algoritmo RIPPER foi superior a taxa de acerto no método Paralog_e. Por exemplo, na iteração dois, a média dos resultados obtidos com o algoritmo RIPPER nos subconjuntos foi de 61,67% de acerto, enquanto que a taxa de acerto com o método Paralog_e foi de 56,15%.

Monk 2		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	**67,12**	63,88	51,38	**67,12**	63,42	63,42	63,88	63,88	59,72	57,40	**67,12**	62,12±4,78
2	**67,12**	55,09	57,40	**67,12**	**67,12**	51,39	**67,12**	59,72	**67,12**	**67,12**	**67,12**	62,63±6,15
3	**67,12**	48,61	56,94	**67,12**	**67,12**	59,72	**67,12**	59,72	**67,12**	**67,12**	51,38	61,20±7,09
4	**67,12**	**67,12**	63,42	64,81	59,72	57,40	42,59	51,38	47,91	57,40	62,50	57,43±7,91
5	**67,12**	54,16	57,40	**67,12**	61,57	62,50	51,38	57,40	**67,12**	**67,12**	51,38	59,72±6,29
6	**67,12**	57,40	48,61	63,42	57,40	51,38	**67,12**	59,72	**67,12**	59,72	61,11	59,30±6,03
7	**67,12**	57,40	**67,12**	56,48	**67,12**	55,09	**67,12**	**67,12**	**67,12**	55,09	**67,12**	62,68±5,77
8	32,87	48,61	**63,89**	53,70	56,48	63,42	57,40	59,72	63,42	51,39	52,31	57,03±5,50
9	**67,12**	59,72	**67,12**	**67,12**	**67,12**	59,72	57,40	**67,12**	**67,12**	57,40	63,89	63,37±4,33
10	**67,12**	**67,12**	63,42	52,31	63,42	57,40	52,31	51,39	**67,12**	57,40	61,11	59,30±6,03
Média Geral	**63,70±10,83**											60,48±2,25

Tabela 11 - Resultados obtidos para a base Monk2

Na análise dos resultados para a base Monk2, demonstrados na Tabela 11 a média geral dos resultados do método Paralog_e (63,70%) foi superior a média dos resultados em todas as iterações com o uso do algoritmo RIPPER (60,18%). Observando as soluções individualmente em cada iteração, em nove das dez iterações o melhor resultado do algoritmo RIPPER nos subconjuntos foi idêntico ao resultado do método Paralog_e.

Apenas na iteração oito, o melhor resultado conseguido com o RIPPER no subconjunto dois (63,89% de acerto) foi superior a taxa de acerto do método Paralog_e (32,87% de acerto). Somente na iteração oito a média dos resultados nos subconjuntos (57,03%) foi superior ao método desenvolvido (32,87%).

Soybean		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	**75,18**	55,47	53,18	57,66	54,01	48,17	50,36	60,58	56,20	52,55	55,47	54,37±3,57
2	58,39	54,01	51,09	45,98	51,82	**65,69**	61,31	49,63	41,60	62,77	55,47	53,94±7,60
3	**60,58**	45,25	56,20	53,28	**60,58**	50,36	51,09	53,28	48,17	56,93	56,20	53,13±4,56
4	**57,66**	55,47	45,25	46,71	44,52	50,36	32,11	52,55	56,20	48,17	48,90	48,02±6,86
5	**70,80**	59,85	47,44	58,39	52,55	41,60	37,95	54,74	51,09	45,98	48,90	49,85±6,97
6	**66,42**	54,74	59,85	46,71	61,31	50,36	44,52	56,93	55,47	49,63	52,55	53,21±5,48
7	52,55	45,98	45,25	45,25	54,74	47,44	50,36	51,09	45,98	**56,93**	37,95	48,10±5,42
8	56,93	44,52	56,93	50,36	56,93	**57,66**	51,82	40,14	43,78	50,36	51,82	50,43±6,02
9	**59,85**	50,36	55,47	51,82	51,09	45,98	45,98	52,55	54,01	48,90	48,17	50,43±3,20
10	**72,26**	43,79	45,25	47,44	38,68	48,90	55,47	57,66	51,09	62,77	54,74	50,58±7,22
Média Geral	**63,06±7,59**											51,21±2,32

Tabela 12 - Resultados obtidos para a base Soybean

Nas soluções encontradas para a base Soybean, descritas na Tabela 12, a média geral do método Paralog_e (63,06%) foi estatisticamente melhor do que a média obtida nas iterações pelo RIPPER (51,21%).

Ao avaliar os resultados individualmente em cada iteração, em todas as dez iterações o método Paralog_e conseguiu taxa de acerto superior a média dos resultados obtidos nos subconjuntos através do algoritmo RIPPER. Apenas em quatro iterações a maior taxa de acerto obtida entre os subconjuntos foi igual ou superior a taxa de acerto do método proposto. Por exemplo, na iteração sete, o

subconjunto nove apresentou taxa de 59,63% de acerto, superior ao resultado obtido com o Paralog_e com 52,55%.

Na análise dos resultados para a base Vehicle, descritos na Tabela 13, a média obtida com o método Paralog_e (52,35%) foi equivalente à média dos resultados obtidos nas iterações com o algoritmo RIPPER (52,24%).

Na avaliação dos resultados em cada iteração, em sete das dez iterações existentes o método Paralog_e se mostrou inferior a média dos resultados obtidos nos subconjuntos com o algoritmo RIPPER. Por exemplo, na iteração dois a taxa de acerto com o método desenvolvido foi de 49,41% enquanto que a média dos resultados nos subconjuntos foi de 52,35%.

Vehicle		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	58,23	51,17	47,64	52,94	43,52	**59,41**	49,41	44,70	51,17	41,76	54,11	49,58±5,36
2	49,41	42,35	54,70	49,41	**63,52**	48,82	58,23	58,82	51,17	45,29	51,17	52,35±6,50
3	58,82	53,52	**65,29**	51,17	58,23	48,23	51,17	42,94	58,82	60,00	54,70	54,14±6,46
4	48,23	47,05	47,64	55,29	42,94	47,64	59,41	**61,76**	51,76	47,05	48,23	50,88±6,07
5	50,00	50,00	55,29	**63,52**	57,05	51,17	50,00	51,17	48,82	56,47	48,23	53,17±4,82
6	47,64	51,17	57,64	53,32	41,76	47,05	55,29	**61,17**	49,41	43,52	45,29	50,56±6,34
7	55,29	47,05	56,47	57,64	51,17	55,29	**68,82**	63,52	57,64	64,11	47,05	56,88±7,22
8	51,76	55,29	50,00	36,47	54,11	57,94	44,11	47,64	57,64	**61,76**	52,94	51,79±7,49
9	54,70	52,94	44,70	51,76	53,32	58,23	50,00	45,29	48,23	**60,58**	58,82	52,39±5,54
10	49,41	51,17	44,70	54,70	54,11	46,47	51,76	41,76	49,41	**58,23**	51,17	50,35±4,95
Média	**52,35±4,12**											52,24±2,18

Tabela 13 - Resultados obtidos para a base Vehicle

Ainda considerando a Tabela 13, em todos os casos, nas dez iterações a maior taxa de acerto conseguida nos subconjuntos se comparada com o método desenvolvido tem resultados superiores.

Para os experimentos realizados na base Tic-Tac-Toe, os resultados foram apresentados na Tabela 14. Conforme observação, a média geral para o método Paralog_e, com 65,22% de acerto, foi inferior a média geral do algoritmo RIPPER, com 67,76% de taxa de acerto.

Tic-Tac-Toe		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	62,63	66,15	71,88	67,71	**72,40**	65,63	56,77	64,58	66,67	64,58	68,23	66,46±4,36
2	65,10	**73,96**	71,88	71,75	67,71	68,23	68,23	67,19	61,46	68,23	69,79	68,84±3,41
3	65,63	61,98	70,31	55,21	60,94	66,67	59,38	72,40	70,31	**77,08**	73,96	66,82±7,15
4	65,63	60,42	65,10	**69,79**	65,10	59,90	65,53	66,67	63,02	62,50	68,23	64,63±3,20
5	65,63	67,71	64,58	67,71	65,63	53,65	62,50	**70,83**	65,10	65,10	65,10	64,79±4,52
6	65,63	**73,44**	60,94	67,71	67,71	67,71	60,94	65,63	67,71	68,75	67,71	66,82±3,68
7	65,63	63,02	**72,92**	64,58	67,19	65,10	70,83	67,71	**72,92**	63,02	**72,92**	68,02±4,10
8	65,63	69,79	**77,60**	77,08	75,00	62,50	63,54	66,67	73,44	69,79	65,10	70,05±5,56
9	65,63	71,88	73,44	67,71	70,83	72,92	73,44	73,96	61,46	73,44	**75,52**	71,46±4,10
10	65,10	**73,96**	58,33	**73,96**	67,71	70,83	64,58	67,71	**73,96**	72,92	73,44	69,74±5,20
Média Geral	65,22±0,94											**67,76±2,27**

Tabela 14 - Resultados obtidos para a base Tic-Tac-Toe

Ao analisar a média do RIPPER em cada iteração e o método Paralog_e, em oito iterações a média do algoritmo RIPPER foi superior a taxa de acerto obtida pelo Paralog_e. Por exemplo, na iteração nove, a média do algoritmo de indução de regras RIPPER foi 71,46% enquanto que no método Paralog_e a taxa de acerto foi de 65,10%.

Vowel		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	40,40	28,28	26,76	35,85	37,87	32,82	37,37	28,28	26,26	**44,44**	38,88	33,68±6,15
2	**37,37**	33,33	35,85	29,29	35,35	28,78	36,86	24,74	27,27	31,81	31,81	31,51±3,97
3	**41,41**	31,31	37,87	33,83	26,26	35,85	33,83	26,76	33,83	27,77	31,31	31,86±3,92
4	**48,48**	31,81	31,81	27,77	40,40	26,76	39,89	25,25	31,31	32,82	25,25	31,31±5,43
5	**51,01**	42,92	37,87	29,29	29,79	31,81	35,35	35,85	25,75	27,27	36,86	33,28±5,38
6	40,91	36,86	23,73	28,78	37,87	28,28	31,31	35,85	27,77	**47,97**	33,83	33,23±6,89
7	43,43	25,75	31,31	25,75	30,30	14,74	37,87	**44,94**	34,94	33,83	25,25	30,47±8,27
8	**41,41**	28,28	33,83	37,37	29,79	35,85	24,74	18,68	29,29	35,35	39,39	31,26±6,36
9	**47,47**	39,39	34,34	33,33	36,36	37,87	26,26	23,73	32,32	37,37	27,27	32,82±5,39
10	**46,97**	26,76	35,35	31,81	38,38	26,26	36,36	40,40	28,28	32,32	38,88	33,48±5,18
Média Geral	**43,89±4,35**											33,29±1,14

Tabela 15 - Resultados obtidos para a base Vowel

Novamente na análise individual dos resultados em cada iteração, o maior resultado obtido nos subconjuntos foi superior ao resultado do método desenvolvido. Por exemplo, na iteração três, no subconjunto nove foi obtido uma

taxa de acerto de 77,08% para o algoritmo RIPPER enquanto que na mesma iteração o desempenho do método Paralog_e foi de 65,63%.

Nos resultados para a base Vowel, demonstrados na Tabela 15, a média geral no método Paralog_e, com 43,89% de acerto foi significativamente superior a média geral do algoritmo RIPPER, que apresentou 33,29% de acerto.

Na análise dos resultados parciais, ou seja, analisando os resultados individualmente em cada iteração, a taxa de acerto do método Paralog_e, nas dez iterações, foi superior a média do algoritmo RIPPER nos subconjuntos.

Ao considerar o maior resultado conseguido entre os subconjuntos o Paralog_e mostrou-se mais competitivo em sete das dez partições. Por exemplo, na iteração cinco, o método Paralog_e obteve 51,01% de acerto enquanto que o melhor resultado, na mesma iteração, foi de 42,92% de acerto no subconjunto um.

Car		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	54,05	74,28	**78,03**	77,75	61,85	73,41	69,36	65,32	75,14	65,32	67,05	70,75±8,72
2	56,36	73,70	64,74	66,47	62,72	67,34	**79,77**	68,79	75,43	71,39	67,92	69,83±5,23
3	52,31	70,52	**82,37**	70,81	70,23	73,12	69,65	71,68	69,94	67,34	74,57	72,02±4,13
4	55,20	74,43	78,32	72,83	**78,90**	72,83	75,14	67,63	70,81	68,79	73,99	73,37±3,67
5	65,03	75,43	74,86	73,99	65,90	70,23	68,79	**77,75**	69,65	74,28	77,46	72,83±3,96
6	58,38	71,39	72,54	67,63	71,39	**77,17**	67,34	72,25	74,28	71,39	69,94	71,53±2,91
7	55,78	69,36	68,50	68,21	73,70	69,36	71,97	67,05	**76,01**	66,19	73,12	70,35±3,19
8	57,80	75,14	76,01	**78,61**	76,01	73,70	73,12	72,25	72,54	64,16	70,81	73,24±3,91
9	63,29	73,41	**76,59**	75,72	74,57	72,54	75,43	73,70	72,25	70,23	73,99	73,84±1,87
10	64,16	71,68	67,92	72,83	68,50	73,70	65,32	**76,59**	75,43	73,99	73,99	71,99±3,62
Média Geral	58,24±4,45											**71,98±1,36**

Tabela 16 - Resultados obtidos para a base Car

Ao observar os resultados para a base Car, presentes na Tabela 16, percebe-se que na média geral o método Paralog_e obteve resultados inferiores a média geral dos resultados do algoritmo RIPPER, nas dez iterações. O método Paralog_e teve taxa de 58,24% de acerto, enquanto que a média geral do RIPPER atingiu 71,98% de acerto.

Nos resultados individuais, o maior resultado entre os subconjuntos e a média obtida nos subconjuntos, em cada iteração, foram superiores a taxa de acerto do método Paralog_e, em todas as iterações.

Para a base de dados Segment, a média geral do método Paralog_e obteve 87,87% de acerto e foi superior a média geral do algoritmo RIPPER, com 82,83%.- conforme Tabela 17.

Na análise dos resultados para a base Segment, de acordo com a Tabela 17, em seis das dez iterações o método Paralog_e obteve taxas de acerto superiores ao maior resultado entre os subconjuntos, para a mesma iteração. Por exemplo, na iteração sete, o método Paralog_e obteve taxa de acerto de 90,69% enquanto que, na mesma iteração, o maior resultado obtido entre os subconjuntos, está presente no subconjunto seis, em que o resultado foi de 87,01%.

Segment		Subconjuntos										
Iteração	Paralog_e	1	2	3	4	5	6	7	8	9	10	Média
1	**90,04**	78,35	78,35	88,96	87,66	83,76	86,79	83,55	79,00	78,35	86,14	83,09±4,26
2	84,41	85,71	**86,58**	85,06	77,70	78,35	73,37	77,27	86,14	84,41	83,98	81,86±4,71
3	**89,39**	81,81	82,03	80,30	83,11	88,74	83,33	82,25	85,06	84,19	85,28	83,61±2,37
4	**90,25**	85,06	81,16	84,84	88,31	86,14	88,09	84,63	80,51	83,98	86,36	84,91±2,58
5	**91,12**	85,49	83,54	86,44	88,96	75,75	82,90	83,76	75,75	74,67	75,10	81,24±5,38
6	**91,99**	81,16	87,44	82,68	85,49	77,70	76,83	82,68	85,06	84,63	85,49	82,92±3,48
7	**90,69**	86,36	81,38	84,63	80,95	84,84	87,01	77,70	83,76	79,22	85,71	83,16±3,17
8	83,98	79,87	78,35	76,19	77,92	79,22	80,51	85,93	81,38	**86,79**	85,49	81,17±3,68
9	86,36	85,49	88,52	79,87	82,90	**90,25**	88,74	83,76	85,28	86,79	78,78	85,04±3,78
10	80,51	76,40	82,25	**88,09**	86,58	84,84	82,90	78,35	78,13	82,68	73,16	81,34±4,72
Média Geral	**87,87±3,83**											82,83±1,43

Tabela 17 - Resultados obtidos para a base Segment

Em apenas uma das dez iterações, a média dos subconjuntos utilizando o algoritmo RIPPER foi superior ao método desenvolvido. Este caso é apresentado na iteração dez em que a média dos resultados dos subconjuntos foi de 81,34%, enquanto que no Paralog_e foi de 80,51%.

6.2. Análise dos Resultados Globais

Na comparação dos resultados globais, entre a média do método Paralog_e e a média do algoritmo RIPPER nas dez iterações das bases, ficou evidente que o método Paralog_e teve média de resultados estatisticamente[6] melhores que a média do algoritmo RIPPER em sete das dez bases em que os experimentos foram aplicados, conforme demonstrado na Tabela 18. A coluna "Diferença Percentual" indica o valor obtido pela divisão da "Média Geral no Paralog_e" pela "Média Geral no Ripper".

Nas bases: Monk1, Tic-Tac-Toe e Car, em que o método Paralog_e teve desempenho inferior, foi constatado que o número total de classes que determinam o domínio destas bases é menor se comparado com a quantidade total de classes das demais bases em que o resultado do método desenvolvido foi melhor. Nos experimentos realizados ficou evidente que o desempenho do método Paralog_e foi melhor para as bases com maior número de atributos.

Base	Média Geral no Paralog_e	Média Geral no Ripper	Diferença Percentual entre os Métodos	Relação Total de Atributos/ Total de Exemplos
Audiology	**48,69 ± 2,94**	32,65 ± 3,88	1,49	0,305
Zoo	**61,43 ± 4,73**	49,14 ± 4,14	1,25	0,168
Soybean	**63,06 ± 7,59**	51,21 ± 2,32	1,23	0,051
Vowel	**43,89 ± 4,35**	33,29 ± 1,14	1,32	0,013
Segment	**87,87 ± 3,83**	82,83 ± 1,43	1,06	0,008
Monk2	**63,70 ±10,83**	60,48 ± 2,25	1,05	0,014
Vehicle	**52,35 ± 4,12**	52,24 ± 2,18	1,00	0,021
Monk1	55,23 ± 5,66	**55,50 ± 4,25**	1,00	0,014
Tic-Tac-Toe	65,22 ± 0,94	**67,76 ± 2,27**	0,96	0,009
Car	58,24 ± 4,45	**71,98 ± 1,36**	0,81	0,003

Tabela 18 - Comparação dos Resultados

Outra característica verificada nas bases em que o desempenho ficou a desejar, foi que o número total de atributos é menor se comparado com o total de atributos presente no domínio das demais bases, em que os experimentos foram

[6] Alguns desses resultados não são estatisticamente significantes.

aplicados (ver coluna Relação Total de Atributos/ Total de exemplos, presente na Tabela 18). Por exemplo, a base Car que obteve desempenho inferior no método Paralog_e teve a proporção de apenas 0,003 atributos pelo total de exemplos existentes na base.

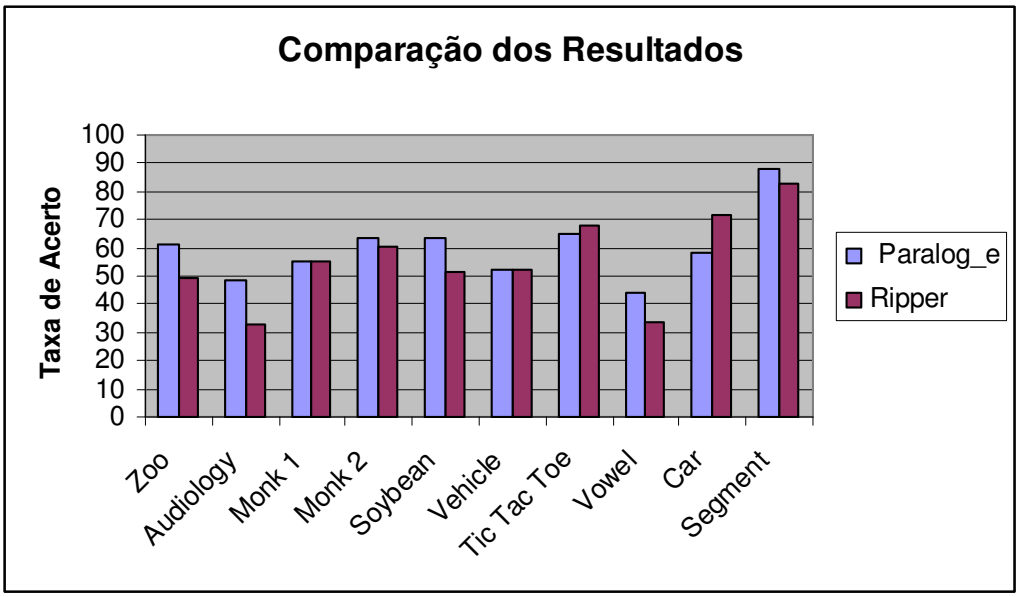


Figura 26 - Comparação Gráfica dos Resultados

Os resultados das médias obtidas presentes na Tabela 18, são apresentados na Figura 26, que demonstra graficamente o desempenho médio do método Paralog_e e a média dos resultados do algoritmo RIPPER.

A base Audiology teve a diferença mais significativa entre as médias dos dois métodos aplicados, em comparação as demais bases. Uma hipótese para a razão deste desempenho pode estar relacionada ao número total de atributos presente nesta base (com uma proporção de 0,0305 entre o número total de atributos e o número total de exemplos contidos na base, conforme Tabela 18), capaz de gerar maior quantidade de regras nos subconjuntos, e conseqüentemente, uma melhor representação das características da base.

Nas bases Audiology, Zoo e Soybean, ocorreram as maiores diferenças entre os dois métodos aplicados. As diferenças percentuais foram de 1,49%,

1,25% e 1,23%, respectivamente para as bases, e foram favoráveis ao método Paralog_e. Neste caso, para as bases Audiology e Soybean o número de total de atributos foram os maiores em relação aos demais experimentos.

Outra observação interessante foi que nas bases: Audiology, Zoo e Soybean, o número total de classes é o maior em relação às demais bases nos experimentos. Esta característica pode permitir uma melhor distinção entre as regras geradas.

O número de atributos e o número de valores que estes atributos podem assumir foram variáveis importantes que colaboraram no bom desempenho do método. Apesar disso, segundo Freitas [Freitas & Lavington, 1998], em algoritmos de indução de regras, o produto cartesiano dessas variáveis aumenta exponencialmente o tamanho do espaço de tuplas e, conseqüentemente, o espaço de regras a ser pesquisado.

Capítulo 7

Conclusões

A mineração distribuída de dados viabiliza a aplicação de técnicas de mineração de dados em ambientes distribuídos. As pesquisas em mineração distribuída têm como interesse principal o projeto de algoritmos, técnicas e arquiteturas que possibilitem a análise de dados fisicamente distribuídos e que permitem aliar ao desenvolvimento, soluções mais viáveis financeiramente e computacionalmente.

Técnicas em mineração distribuída viabilizam a descoberta de conhecimento em grandes volumes de dados através da combinação de diversos classificadores.

A divisão dos dados em subconjuntos menores permite treinar simultaneamente classificadores para posteriormente combiná-los em um único conjunto, porém, a existência de dados inconsistentes nos subconjuntos pode comprometer o desempenho e formar classificadores com opiniões contraditórias.

A ocorrência de dados inconsistentes pode estar relacionada a diversos motivos, como por exemplo, a união de bases de dados geradas a partir e fontes distribuídas, ruídos nos dados ou mesmo erro na coleta dos dados.

A utilização de técnicas de gerenciamento de incerteza permite um raciocínio mais adequado nas situações em que os dados ou conceitos obtidos nem sempre estão completos ou consistentes.

Neste trabalho foram empregados conceitos de Lógica Paraconsistente no desenvolvimento de um método capaz de tomar decisões confiáveis a partir de um conjunto de classificadores, mesmo quando opiniões contraditórias estivessem presentes.

A utilização dos conceitos em Lógica Paraconsistente permitiu atribuir às regras, fatores evidenciais de crença e descrença, quantificando o grau de

importância da regra em relação aos subconjuntos. Com a ordenação do conjunto de regras foi possível mensurar as regras em relação ao estado lógico que representa a verdade, e determinar um critério de decisão na escolha entre as regras candidatas.

O critério de decisão elege a regra que cobre adequadamente o exemplo de teste e possui a menor distância em relação ao conceito verdade, conforme o mecanismo de raciocínio adotado em Lógica Paraconsistente.

Com objetivo de avaliar o desempenho do método desenvolvido, os resultados foram comparados às médias do algoritmo RIPPER. Nos experimentos realizados em diversas bases de dados, o método desenvolvido, denominado método Paralog_e, teve desempenho competitivo em relação às médias do algoritmo RIPPER

Não existe uma limitação do método em relação ao número de atributos presente na base. Foi observado, porém, que os resultados mais expressivos do método Paralog_e foram identificados nas bases formadas por um número maior de atributos. Uma hipótese para a razão deste desempenho é que um número maior de atributos pode gerar conjuntos de regras mais expressivos e, conseqüentemente, uma melhor representação das características da base.

Na análise dos resultados, implicitamente foi possível constatar que a relação entre o número de atributos e os valores que estes atributos podem assumir pode implicar na construção de regras mais distintas entre si.

A principal contribuição deste trabalho foi o desenvolvimento de uma nova metodologia, que pode ser aplicada em ambientes distribuídos para a detecção de inconsistências em algoritmos de indução de regras. O diferencial nesta metodologia é evitar o alto fluxo de comunicação das mensagens entre os classificadores, o que reduz significativamente o tempo necessário para processamento se comparado a outros métodos de mineração distribuída de dados.

O método Paralog_e tem como objetivo tratar possíveis inconsistências entre as regras obtidas a partir de diversos classificadores. O método é aplicado

para integrar classificadores, possivelmente inconsistentes, obtidos a partir de métodos de mineração distribuída orientada a dados.

Desta forma, a contribuição deste trabalho incide em pelos menos três áreas de interesse da comunidade científica: mineração distribuída, algoritmos de aprendizagem simbólicos e aplicações de técnicas de gerenciamento de incerteza.

7.1. Trabalhos Futuros

Diversas extensões deste trabalho podem ser exploradas. Uma das propostas é aplicar o mesmo método baseado em outros algoritmos de indução, avaliar o desempenho sob diferentes distribuições do conjunto de treinamento e teste.

Uma outra extensão está relacionada à aplicação de técnicas de redução de dimensionalidade de dados, verificar o comportamento dessas abordagens em bases de dados mais volumosas e possivelmente propor novas soluções para o problema de escalabilidade.

O desempenho do modelo proposto pode ser avaliado utilizando outras medidas de avaliação de regras, identificando quais são as melhores regras sustentadas pelos dados, ou ainda, selecionando as que possam trazer algum conhecimento surpreendente ou inesperado.

Uma nova abordagem pode ser desenvolvida para atribuir fatores evidenciais às regras, levando em consideração a importância da regra dentro de seu subconjunto. A nova abordagem tem como objetivo a diferenciação das regras em um mesmo nível, mas obtidas em diferentes subconjuntos.

Finalmente, outro trabalho futuro consiste em propor alternativas para casos de inconsistência em dados heterogêneos, ou seja, bases de dados que podem conter características diferentes.

Referências

[Angelotti, 2001] Angelotti, E. S. *Utilização da Lógica Paraconsistente na Implementação de um Sistema Multi-Agente*. Dissertação de Mestrado, Pontifícia Universidade Católica do Paraná, Curitiba, 2001.

[Aronis et al., 1997] Aronis, J. M.; Provost, F. J.; Kolluri, V.; Buchanan, B. G. *The WoRLD: Knowledge Discovery from Multiple Distributed Databases*. In Proceedings of 10 th International Florida AI Research Symposium, pag. 337-341. 1997

[Ávila, 1996] Ávila, B. C. *Uma Abordagem Paraconsistente Baseada em Lógica Evidencial para Tratar de Exceções em Sistemas de Frames com Múltipla Herança.* Tese de Doutorado, Escola Politécnica da Universidade de São Paulo. São Paulo, 1996.

[Blair & Subrahmanian, 1988] Blair, H.A.; Subrahmanian, V. S. *Paraconsistent Foundations for Logic Programming*. Journal of Non-Classical Logic, 5, 2, pag. 45-73, 1988

[Blake et al., 1998] Blake, C.L.; Newman, D.J.; Hettich, S.; Merz, C.J. UCI Repository of machine learning databases. [http://www.ics.uci.edu/~mlearn/ MLRepository.html]. University of California, Irvine, Dept. of Information and Computer Sciences, 1998.

[Breiman, 1996] Breiman, L. *Bagging Predictors*. Machine Learning, n. 2, vol. 24, p.p. 123-140, 1996.

[Casanova et al., 1987] Casanova, M. A; Giorno, F. A.C.; Furtado, A.L.F. Programação em Lógica e a Linguagem Prolog. Ed. Edgard Blücher Ltda. São Paulo, 1987.

[Chan & Stolfo, 1995] Chan, P. K.; Stolfo, S. J. *A Comparative Evaluation of Voting and Meta-Learning on Partitioned Data.* In Proceedings of the Twelfth International Conference on Machine Learning, pages 90-98. 1995

[Chan & Stolfo, 1998] Chan, P. K.; Stolfo, S. J. *Toward Scalable Learning with Non-Uniform Distribution: Effects and a Multi-Classifier Approach*. Knowledge Discovery and Data Mining. Pages 164-168. 1998.

[Clark & Niblett, 1989] Clark, P.; Niblett, T. *The CN2 Induction algorithm*. Machine Learning, 3 (4): 261-283. 1989

[Clark & Boswell, 1991] Clark, P. ; Boswell, R. *Rule Induction with CN2: Some recent improvements*. In Proceedings of the 5th European Conf. On Machine Learning, vol 482 of Lecture Notes in Artificial Intelligence, pag 151-163. Ed. Springer. 1991.

[Cohen, 1995] Cohen, W. W. *Fast Effective Rule Induction*. In Proceedings of the 12th International Conference in Machine Learning (ICML '95). Pages 115-123. 1995

[da Costa et al., 1995] da Costa; N. C. A.; Prado, J. P. A.; Abe, J. M., Ávila, B. C.; Rillo, M. *Paralog: Um Prolog Paraconsistente baseado em Lógica Anotada*. Em: Coleção Documentos, n 18, série: Lógica e Teoria da Ciência. Instituto de Estudos Avançados, Universidade de São Paulo, Abril 1995.

[da Costa et al., 1999] da Costa, N. C. A.; Murolo, A. C.; Abe, J. M.; da Silva Filho, J. I.; Leite, C.F.S. *Lógica Paraconsistente Aplicada.* Ed. Atlas, São Paulo, 1999.

[da Costa et al., 2000] da Costa, N. C. A; Abe, J. M. *Paraconsistência em Informática e Inteligência Artificia*l. Revista Estudos Avançados n 14, vol 39 - USP. São Paulo, Maio/Agosto 2000.

[Dayal et al., 1999] Dayal, U.; Chen,Q.; Hsu, M. *Large Scale Data Mining Applications: Requirements and Architectures*. Hewlitt-Packard Corp. In: Workshop on Large-Scale Parallel KDD Systems. August 15th, 1999, San Diego, CA, USA in conjunction with ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD99)

[Enembreck, 1999] Enembreck, F. *Um Sistema Paraconsistente para Verificação Automática de Assinaturas Manuscritas*. Dissertação de Mestrado PPGIA – PUCPR. Curitiba, 1999.

[Fayyad et al., 1996] Fayyad, U. M; Piatetsky-Shapiro, G.; Smyth, P.; Uthurusamy, R. *Automating the Analysis and Cataloging of Sky Surveys*. In: *Advances in Knowledge Discovery and Data Mining*. AAAI Press/ The MIT Press. ISBN 0-262-56097-6.Pages 471-493. 1996

[Ferreira et al., 2004] Ferreira, S. M. N.; Ávila, B. C.; Freitas, A. A. *Handling Inconsistency in Distributed Data Mining with Paraconsistent Logic*. In: Turkish Symposium on Artificial Intelligence and Neural, Tainn. 2004.

[Flach & Lavrac, 2003] Flach, P.; Lavrac, N. Rule Induction. In Berthold, M. & Hand, D. editors. Intelligent Data Analysis. Springer-verlag. 2003.

[Frank et al, 2000] Frank, E. WEKA Plataforma para Experimentos em Aprendizagem de Máquina. [http://www.cs.waikato.ac.nz/ml/weka]

[Freitas & Lavington, 1998] Freitas, A.; Lavington, S. H. Approaches to Speed Up Data Mining. *Mining Very Large Databases with Parallel Processing*. ISBN 0-7923-8048-7. Pages 89-108. Kluwer Academic Publishers, The Netherlands, 1998.

[Hall et al., 1999] Hall, O. L.; Chawla, N.; Bowyer, K. W.; Kegelmeyer, P. *Learning Rules from Distributed Data*. Department of Computer Science and Engineering, University of South Florida. USA. ISBN 3-540-67194-3. 1999

[Hasegawa, 2004] Hasegawa, F. M. *Uma Abordagem baseada em Lógica Paraconsistente para Avaliação de Ofertas em Negociações entre Organizações Artificiais*. Dissertação de Mestrado PPGIA – PUCPR. Curitiba, 2004.

[Henkind & Harrison, 1988] Henkind, S. J.; Harrison, M. C. *An Analysis of Four Uncertainty Calculi*. IEEE Transactions n Systems, Man and Cybernetics, vol. 18, n. 5, September/ October, 1988.

[Köpf et al., 2000] Köpf, C.; Taylor, C., Keller, J. *Meta-Analysis: From Data Characterisation for Meta-Learning to Meta-Regression*. Pavel Brazdil and Alipio Jorge, editors, In Proceedings of the PKDD-2000 Workshop on Data Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions. Lyon, France, 2000.

[Kowalsky, 1979] Kowalsky, R. *Logic for Problem Solving*. Ed.North-Holland

[Kurgan & Cios, 2003] Kurgan, L.; Cios, K. J. *Meta Mining Architecture for Supervised Learning.* Ph.D dissertation, University of Colorado, Departament of Computer Science, 2003.

Formatado:

[Kurgan, 2004] Kurgan, L. *Reducing Complexity of Rule Based Models via Meta Mining*. The 2004 Int. Conference on Machine Learning and Applications - ICMLA 2004, Louisville, KY, EUA.

[Ladeira & Viccari, 1996] Ladeira, M.; Viccari, R. M. *Representação do Conhecimento Incerto*. XIII Symposium on Artificial Intelligence SBIA'96, Curitiba, October, 1996.

[Lavrac et al.,1999] Lavrac, N.; Flach, P.; Zupan, Blaz. *Rule evaluation measures: A unifying view*. In Dzeroski, S. & Flathc, P. Editors. In 9th International Workshop on Inductive Logic Programming (ILP '99), vol. 1634 of Lecture Notes in Computer Science, pag. 174-185. Ed. Springer. 1999.

[Lavrac et al., 2004] Lavrac, N.; Kavsek, B.; Flach, P.; Todorovski, L. *Subgroup Discovery with CN2-SD*. Journal of Machine Learning Research, 5:153-188. 2004.

[Littlestone & Warmuth, 1992] Littlestone, N.; Warmuth, M. *The weighted majority Algorithm.* Technical IEEE Symposium on Foundations of Computer Science. Pages 256-261. 1992

[Mitchell, 1997] Mitchell, T. M. *Machine Learning*. McGraw-Hill Series in Computer Science, McGraw-Hill Companies, USA, 1997.

[Motro, 1990] Motro, A. *Accomodating Imprecision in Database Systems: Issues and Solutions*. SIGMOD Record, vol.19 , n.4, pages 69-74, 1990.

[Motro, 1994] Motro, A. *Management of Uncertainty in Database Systems*. In *Modern Database Systems: the Object Model, Interoperability and Beyond* (W. Kim, Editor), Addison-Wesley/ACM Press, 1994, pp. 457–476.

[Nicoletti & Monard, 1993] Nicoletti, M. C.; Monard, M. C. Herbrand Interpretation, Model and Least Model within the Framework of Logic Programming. Notas do Instituto de Ciências Matemáticas de São Carlos. ISSN 0103-2577, pag. 30. São Carlos, Junho, 1993.

[Parsons, 1996] Parsons, S. *Current Approaches to Handling Imperfect Information in Data and Knowledge Bases*. IEEE Transactions on Knowledge and Data Engineering, Vol 8, Issue 3, page 353-372. June, 1996. ISSN:1041-4347

[Prati et al., 2002] Prati, R. C.; Baranauskas, J. A.; Monard, M. C. *Padronização da sintaxe e informações sobre regras induzidas a partir de algoritmos de aprendizado de máquina simbólico.* Revista Eletrônica de Iniciação Científica, 2 (3). [http://www.sbc.org.br/reic/edicoes/2002e3/] acessado em 10/06/2006. Publicação Setembro,2002

[Prati, 2006] Prati, R. C. *Novas Abordagens em Aprendizado de Máquina para Geração de Regras, Classes Desbalanceadas e Ordenação de Casos*. Tese apresentada ao Instituto de Ciências Matemáticas e de Computação ICMC-USP. Maio, 2006.

[Prodomidis et al., 1999a] Prodomidis, A. L.; Stolfo, S. J.; Chan, P. K. *Effective and Efficient Pruning of Meta-Classifiers in a Distributed Data Mining System.* Technical Report, Columbia University, 1999.

[Prodomidis et al., 1999b] Prodomidis, A. L.; Chan, P. K.; Stolfo, S. J. *Distributed Data Mining in Credit Card Fraud Detection*. IEEE Nov/ Dez1999 (Vol. 14, No. 6)

[Prodomidis et al., 2000] Prodomidis, A. L., Stolfo, S. J.; Stolfo, P. K. *Meta-Learning in Distributed Data Mining Systems: Issues and Approaches.* In Advances in Distributed and Parallel Knowledge Discovery, Chapter 3. 2000.

[Provost & Hennessy, 1996] Provost, F. J.; Hennessy, D. N. *Scaling Up: Distributed Machine Learning with Cooperation*. In Proceeding AAAI '96, pages 74-79, 1996.

[Provost & Kolluri, 1999] Provost, F.; Kolluri, V. *A Survey of Methods for Scaling Up Inductive Algorithms.* Kluwer Academic Publishers, Boston, USA – 1999.

[Quinlan, 1986] Quinlan, J. R. *Induction of decisión trees*. Machine Learning vol. 1, Kluwer Academic Publishers, pag. 81-106, Netherlands, 1986.

[Quinlan, 1987] Quinlan, J. R. *Generating Production Rules from Decision Trees.* In Proceedings of IJCAI 87, pages 304-307, 1987.

[Quinlan, 1993] Quinlan, J. R. *C4.5: Programs for Machine Learning*. Chapter 5 – From Trees to Rules. Morgan Kaufmann, 1st edition, 1993.

[Rich & Knight, 1994] Rich, E. ; Knight, K. *Inteligência Artificial*, 2 ed. Editora Makron Books do Brasil Ltda, São Paulo, 1994.

[Sandri, 1996] Sandri, S. *Numerical Models for the Treatment of Imperfect Information in Knowledge-Bases Systems*. XIII Brazilian Symposium on Artificial Intelligence, Curitiba, from 23-25 October, 1996.

[Scalabrin et al., 1998] Scalabrin, E.; Bortolozzi, F.; Kaestner, C.; Sabourin, R. *Multicheck: Une Arquiteture d'Agents Cognitifs Independants pour le Traitement*

Automatique des Chèsques Bancaires Brésiliens. In: CIFED's 1998, Canadá, Maio.

[Schapire & Freund, 2001] Schapire, R. E.; Freund, Y. *The Boosting Approach to Machine Learning: An Overview.* In MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA. 2001.

[Subrahmanian, 1987a] Subrahmanian, V. S. *On the Semantics of Quantitative Logic Programs*. Proceedings of 4th IEEE Symposium on Logic Programming, San Francisco, September, pp. 173-182, 1987.

[Subrahmanian, 1987b] Subrahmanian, V. S. *Towards a Theory of Evidential Reasoning in Logic Programming.* Logic Colloquium '87, The European Summer Meeting of the Association for Symbolic Logic, Spain, July, 1987.

[Tsymbal et al., 1999] Tsymbal, A.; Puuronen, S.; Terziyan, V. *Arbiter Meta-Learning with Dynamic Selection of Classifiers and its Experimental Investigation.* In: J.Eder, I.Rozman, T.Welzer (eds.), Advances in Databases and Information Systems: 3rd East European Conference ADBIS'99, LNCS, Vol. 1691, Springer-Verlag, Maribor (1999) 205-217

[vam Rijsbergen, 1979] van Rijsbergen, C. J*. Information Retrieval*. University of Glasgow. [http://www.dcs.gla.ac.uk/Keith/Preface.html] acessado em 10/06/2006. 2nd edition,1979.

[Zadeh, 1965] Zadeh, L.A. *Fuzzy Sets, Information and Control*, 8, pp. 338-353, 1965.

[Zaki, 1998] Zaki, M. J. *Scalable Data Mining for Rules.* Requirements for the Degree Doctor of Philosofy. Department of Computer Science, University of Rochester. New York, 1998.

[Williams et al., 1999] Williams, G. J.; Altas, I; Bakin, S; Christen, P.; Hegland, M.; Marquez, A.; Milne, P.; Nagappan, R.; Roberts, S. *The Integrated Delivery of Large-Scale Data Mining*. The ACSys Data Mining Project. Large-Scale Parallel Data Mining 1999: 24-55

[Witten et al., 2000] Witten, I. ;Frank, E. *Data Mining: Pratical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufman, 2000.

[Wolpert, 1992] Wolpert, D. H. *Stacked Generalization*. Complex Systems Group, Theorical Division for Non-Linear Studies. LA-UR-90-3460. Los Alamos. 1992