Márcio Fuckner

Combinação de Classificadores usando IAD

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

Curitiba
2008

Márcio Fuckner

Combinação de Classificadores usando IAD

Dissertação apresentada ao Programa de Pós-Graduação em Informática 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

Curitiba
2008

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

F951c
2008

Fuckner, Márcio
Combinação de classificadores usando IAD / Márcio Fuckner ; orientador, Fabrício Enembreck. – 2008.
xii, 79 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2008
Bibliografia: f. 73-79

1. Inteligência artificial distribuída. 2. Software – Medição. 3. Algoritmos de computador. I. Enembreck, Fabrício. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD 20. ed. – 006.3

Pontifícia Universidade Católica do Paraná
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Informática

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DEFESA DE DISSERTAÇÃO Nº 03/2008

Aos 28 dias do mês de fevereiro de 2008 realizou-se a sessão pública de Defesa da Dissertação **"Combinação de Classificadores Usando IAD"**, apresentada pelo aluno **Márcio Fuckner** como requisito parcial para a obtenção do título de **Mestre em Informática**, perante uma Banca Examinadora composta pelos seguintes membros:

Prof. Dr. Fabrício Enembreck PUCPR (Orientador)	_____ *(assinatura)*	APROVADO *(aprov/reprov.)*
Prof. Dr. Edson Emílio Scalabrin PUCPR	_____	Aprov.
Prof. Dr. Gustavo Alberto Giménez Lugo UTFPR	_____	APROVADO

Conforme as normas regimentais do PPGIa e da PUCPR, o trabalho apresentado foi considerado APROVADO (*aprovado/reprovado*), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora registradas no Livro de Defesas do programa.

Prof. Dr. Alessandro Lameiras Koerich
Diretor do Programa de Pós-Graduação em Informática

Programa de Pós-Graduação em Informática PPGIa PUCPR

Para Ana Maria, com amor.

Agradecimentos

À minha mulher Ana Maria, por ser minha força e fonte de inspiração diária e por ter compreendido a minha ausência em tantas ocasiões em que precisei me dedicar integralmente ao trabalho.

Ao meu orientador Prof. Dr. Fabrício Enembreck por ser uma fonte de conhecimento e incentivo ao longo da minha pesquisa, o que faz dele um modelo a ser seguido.

À Pontifícia Universidade Católica do Paraná, aos professores do curso de Pós-Graduação em Informática, em especial aos professores Dr. Edson Emílio Scalabrin e Dr. Bráulio Coelho Ávila pelas valiosas discussões e sugestões dadas ao longo deste trabalho.

Ao meu grande amigo Daniel Pavelec, pela amizade duradoura e constante incentivo na caminhada acadêmica.

Ao grande amigo Lian pelo carinho e força que sempre me passou.

Ao parceiro acadêmico e amigo Emerson Romanhuki pelo espírito de equipe e companheirismo que se manteve até os últimos momentos de nossos trabalhos.

Ao meu irmão Rodrigo, que acompanhou grande parte das etapas do meu trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviações

PKD	*Descoberta de Conhecimentos em Paralelo / Parallel Knowledge Discovery*
DKD	*Descoberta Distribuída de Conhecimentos / Distributed Knowledge Discovery*
KDD	*Descoberta de Conhecimentos a partir de Dados / Knowledge Data Discovery*
MDL	*Menor Tamanho de uma Descrição / Minimum Description Length*
DDM	*Mineração Distribuída de Dados / Distributed Data Mining*
IAD	*Inteligência Artificial Distribuída*
IA	*Inteligência Artificial*
RDP	*Resolução Distribuída de Problemas*
SMA	*Sistemas Multi-Agente*
FIPA	*Organização para Agentes Inteligentes Físicos / Foundation for Intelligent Physical Agents*
JADE	*Ambiente de Desenvolvimento de Agentes em Java / Java Agent Development Environment*
PGP	*Planejamento Global Parcial / Partial Global Planning*
GPGP	*Planejamento Global Parcial Generalizado / Generalized Partial Global Planning*
DCOP	*Satisfação Distribuída de Restrições / Distributed Constraint Optimization Problems*
SBB	*Divisão e Conquista Síncrona / Synchronous Branch and Bound*

WEKA	*Ambiente para Análise de Conhecimentos Waikato / Waikato Environment for Knowledge Analysis*
DF	*Facilitador de Diretórios / Directory Facilitator*
MTS	*Serviço de Transporte de Mensagens / Message Transport Service*
UML	*Linguagem Unificada de Modelagem / Unified Modeling Language*
ROC	*Receiver Operating Characteristic*

Resumo

A execução de algoritmos de aprendizagem de máquina para identificação de padrões ou realização de predição é uma etapa importante da mineração de dados. A fusão das técnicas de mineração de dados e computação distribuída viabiliza a execução de algoritmos de aprendizagem de máquina em grandes bases de dados através da combinação ou integração de classificadores, sem que isso comprometa a confiabilidade das predições. Além disso, esta abordagem pode favorecer a descoberta de visões alternativas, quando diferentes algoritmos são utilizados na mesma amostra. Algumas abordagens de combinação de classificadores possuem baixa capacidade de explicação das inferências, uma vez que as decisões dos combinadores em geral ocorrem em tempo de inferência, funcionando como uma caixa preta ou prejudicando a representação final do modelo devido a quantidade, complexidade ou grau de inconsistência entre as regras. Este trabalho apresenta uma proposta de geração de um modelo unificado de regras geradas por agentes de software a partir de bases de dados distribuídas. Neste processo, uma base de dados é dividida em N partes, que são utilizadas por agentes de software que geram N conjuntos de hipóteses, sob a forma de regras de classificação ordenadas. Agentes reativos interativos alcançam seus objetivos solicitando a validação de seus conceitos a outros agentes que avaliam as regras enviadas através de uma heurística proposta. A interação entre os agentes é capaz de produzir um conjunto de regras unificado e livre de conflitos. Os indicadores de desempenho deste modelo distribuído de busca e as técnicas utilizadas para mensurar o conhecimento se mostram adequados quando comparados com outras abordagens distribuídas sobre bases de dados conhecidas da literatura.

Palavras-chave: Mineração Distribuída de Dados, Combinação de Regras, Aprendizagem Simbólica, Resolução Distribuída de Problemas

Abstract

The execution of machine learning algorithms in order to identify trends or prediction purposes is an important step in the data mining context. The fusion of data mining techniques and distributed computing leverages the execution of machine learning algorithms in large datasets through the classifier combination or integration causing positive effects on prediction reliability. Moreover, these approaches allow the discovery of alternative visions, when different algorithms are used in the same dataset. Some classifier combination approaches produces low level inference explanatory capabilities thanks to the decision process made during the inference process, working as a "black box" or generating poor quality models in terms of representation (quantity, complexity or inconsistency level of rules). This work presents an unified ruleset generation process through software agents using distributed datasets. In this process a dataset is splitted into N subsets, which are used by software agents. They generate N hipothesys, in the form of an ordered classification ruleset. Interactive reactive agents reach their objectives validating their concepts with others, which will use a proposed heuristic. The agent interaction approach is capable of producing an unified ruleset free of conflicts. The performance metrics of the distributed search model and the techniques used to evaluate the knowledge in well known literature databases are reasonable when compared with other approaches.

Keywords: Distributed Data Mining, Ruleset Combination, Symbolic Learning, Distributed Problem Solving

Capítulo 1

Introdução

Muitas das técnicas de mineração de dados foram criadas para serem aplicadas em bases de dados centralizadas e precisam de alterações para que sejam viáveis em um ambiente distribuído. A demanda crescente para permitir a mineração de dados massivos e distribuídos em redes com limitações de banda e recursos computacionais motivaram o desenvolvimento de métodos de PKD (*Descoberta de Conhecimentos em Paralelo / Parallel Knowledge Discovery*) e DKD (*Descoberta Distribuída de Conhecimentos / Distributed Knowledge Discovery*) [KLM03] [KPHJ00].

Certas características tornam inviável a execução de algoritmos centralizados para descoberta de conhecimentos. A quantidade e volume das bases de dados crescem a cada dia e mais rapidamente do que as melhorias aplicadas em recursos computacionais e técnicas de aprendizagem indutiva [PC00]. Os motivos para criar modelos distribuídos de mineração de dados são diversos: privacidade, problemas inerentes à localização física, custo de transmissão de dados, limitação de algoritmos de aprendizagem de máquina ou desempenho.

Analisar bases de dados com milhares de registros pode demandar muito tempo e poder computacional se o algoritmo for aplicado de forma seqüencial. Este trabalho propõe um método que viabiliza a mineração distribuída e descentralizada de dados, gerando regras de classificação consistentes sem que haja perda da confiabilidade. Este trabalho mostra que o uso de agentes, mecanismos de cooperação e decomposição de tarefas contribui para a construção de um modelo unificado de conhecimento extraído de conjuntos particionados de dados.

1.1 Objetivo

O objetivo deste trabalho é a criação de um *framework* baseado em agentes para permitir que a atividade de descoberta de conhecimentos ocorra de forma distribuída. A solução é composta por quatro etapas básicas:

- aquisição de dados distribuídos em N partições;
- execução de N agentes dotados de algoritmos de aprendizagem simbólica sobre as bases locais;
- interação entre os agentes para combinar os classificadores gerados na etapa anterior por meio de uma técnica de resolução de conflitos e busca em espaço de estados;
- classificação de novos exemplos usando o conhecimento unificado e consensual.

Como resultado esperado, deve-se gerar um único conjunto de regras reduzido, porém consistente e livre de conflitos. As regras deverão apresentar um bom desempenho sobre as partições utilizadas para treinamento. A concentração dos esforços é na obtenção de um modelo compreensível e com boa taxa de acerto. Uma vez comprovada a eficiência do *framework*, ele poderá ser utilizado para integrar os conhecimentos existentes em aplicações distribuídas, nos casos onde não é permitida a tomada de decisões contraditórias, além do desenvolvimento de aplicações de mineração distribuída de dados.

1.2 Motivação

A utilização de agentes de software nas atividades de mineração de dados está sendo amplamente aplicada e discutida [SPT+97] [SB02] [GA04] [SB05] [PASE06]. Os agentes possuem características intrínsecas de processamento paralelo, colaboração e negociação. Neste cenário cada agente tem embutido em seu comportamento um algoritmo de aprendizagem de máquina. Cada um está associado a um conjunto de treinamento e validação em alguns casos. Os agentes são dotados de características que permitem o planejamento e a colaboração com outros agentes, tendo como objetivo coletivo a criação de um classificador comum.

Uma questão importante e que vem sendo alvo desses estudos é como esses classificadores devem ser integrados. Existem diversas iniciativas explorando técnicas de mineração distribuída de dados e compartilhamento de conhecimento. O trabalho apresentado por [PC00] aplica o conceito de Meta-aprendizagem, no qual classificadores são

combinados para aumentar o poder de predição. Em [SB05] é apresentado um modelo de negociação cooperativa entre agentes. Podemos perceber que a integração de conhecimento entre agentes é um assunto recente e que não possui uma solução universal. Este trabalho vem portanto ajudar a preencher esta lacuna deixada pelas técnicas existentes na literatura.

Outro aspecto motivador que deve ser levado em consideração são os ganhos obtidos ao usar agentes para a resolução de um problema: robustez, escalabilidade e na adoção de um modelo de decomposição de tarefas. A distribuição auxilia na decomposição do problema, o que influencia positivamente na tratabilidade do problema. Por outro lado, sob a ótica de desenvolvimento, a complexidade é substancialmente maior, uma vez que é necessário prover uma arquitetura que permita a coordenação, cooperação e interoperabilidade semântica entre os agentes.

1.3 Organização

Este trabalho está organizado em 7 capítulos. O primeiro capítulo contém uma breve introdução. Os capítulos 2 a 4 contém a fundamentação teórica, necessária para o desenvolvimento da pesquisa. No capítulo 5 é apresentada a metodologia para a solução do problema. O capítulo 6 apresenta os experimentos realizados e resultados obtidos. O trabalho é concluído com o capítulo 7.

Capítulo 2

Mineração de Dados

Nos últimos anos tem se observado um crescimento acentuado das bases de dados em diversas áreas de aplicação [KLM03]. Estes dados podem ser transformados em conhecimentos úteis, seja para explicar padrões de comportamento ou para realizar predições.

A análise tradicional dos dados aplicada por seres humanos é uma atividade inviável, sendo imprescindível a criação de métodos computacionais para a realização desta tarefa. A KDD (*Descoberta de Conhecimentos a partir de Dados / Knowledge Data Discovery*) é uma área de pesquisa focada no desenvolvimento de métodos e técnicas para transformar dados em informações úteis. A *Mineração de Dados* é uma etapa importante do processo de KDD, que faz uso de algoritmos de aprendizagem de máquina e tem como objetivo identificar padrões em bases de dados, seja para descobrir comportamentos relevantes, seja para a realização de predições.

Neste capítulo são apresentados alguns conceitos sobre mineração de dados necessários para a compreensão das principais dificuldades envolvidas nesta área de pesquisa. O foco da pesquisa está voltado para aprendizagem indutiva, mais especificamente sobre regras de classificação. Os demais métodos de mineração são discutidos superficialmente.

2.1 Definições

A mineração de dados em um contexto de KDD utiliza técnicas conhecidas de aprendizagem de máquina, reconhecimento de padrões e métodos estatísticos para extrair padrões nos dados [FPSS96]. Diversas áreas aplicam as técnicas de mineração de dados para extração de padrões relevantes. O tipo do problema e a natureza da informação desejada define qual técnica de mineração é a mais adequada. Dentre as técnicas existentes é possível citar:

- Classificação: é a descoberta de relacionamentos entre atributos de predição e um atributo meta, também conhecido como classe. O objetivo neste caso é criar uma função $f'(x)$ que se aproxime de $f(x)$, dado um conjunto de exemplos de treinamento contendo os atributos previsores e a classe alvo;

- Associação: consiste em identificar relacionamentos entre itens que ocorrem com determinada freqüência em uma base de dados. Uma de suas típicas utilizações é a análise de transações de compra (*Market Basket Analysis*);

- Regressão: similar a classificação, mas com o objetivo de encontrar uma função $f'(x)$ que se aproxime de $f(x)$ para um atributo meta contínuo;

- Agrupamento: Descoberta de grupos/classes formados por elementos com características comuns.

Como citado anteriormente, este trabalho concentra os esforços na descoberta do conhecimento compreensível e na forma de regras de classificação. Essa técnica de aprendizagem de máquina é conhecida como aprendizagem simbólica. A próxima seção aborda este tema.

2.2 Aprendizagem Simbólica

Um sistema de aprendizagem de máquina supervisionado é um programa (indutor) capaz de induzir uma descrição de conceitos (classificador) usando um conjunto de exemplos conhecidos e previamente rotulados com as suas respectivas classes [PBM02]. Um algoritmo recebe como entrada o valor correto da função desconhecida para entradas específicas e deve tentar descobrir a função desconhecida ou algo próximo disso.

Formalmente é possível definir que um exemplo é um par $(x, f(x))$, onde x é a entrada e $f(x)$ é a saída da função aplicada a x [RN04]. A variável x pode ser interpretada como um dado histórico ou tupla. Os atributos presentes em cada tupla podem ser do tipo numérico ou discreto. Baseado no conteúdo desses atributos, é possível criar um modelo para predição de outro. Por exemplo, a partir de informações sobre condições do mercado em diferentes períodos, é possível predizer como as ações de determinado mercado se comportarão. Quando o objetivo da predição é um dado discreto, a tarefa de predição é chamada de classificação. Por outro lado, quando o alvo é um dado contínuo, a técnica é chamada de regressão [AW97]. As soluções são consideradas funções de aproximação que descrevem resumidamente um espaço de atributos.

Sistemas de aprendizagem simbólica são utilizados em situações onde é necessário, além da capacidade de predição, apresentar um modelo compreensível para seres humanos. Árvores de decisão e regras de classificação são representações de aprendizagem simbólica e são usadas como mecanismos para descoberta de padrões e predição. Tanto árvores de decisão quanto regras de classificação têm se mostrado extremamente competitivas em termos de precisão se comparadas a outros métodos de predição como por exemplo, redes neurais [AW97].

No caso de árvores de decisão, o conhecimento é representado através de uma árvore, sendo que cada nó da árvore corresponde a um atributo e as arestas correspondem a condições sobre os atributos. A folha corresponde ao conceito alvo ou classe associada aos testes. Já as regras de classificação, são representações no formado $se < cond >$ então $< alvo >$. Elas podem ser entendidas analogamente a árvores de decisão como se fossem o caminho completo do nó raiz de uma árvore até um conceito alvo.

A avaliação do desempenho dos modelos gerados é um passo importante do processo de descoberta. Certos modelos apresentam alto desempenho na base de treinamento, porém podem ser menos precisos em dados não conhecidos. Ruídos e novos padrões diminuem a capacidade de predição do modelo. Uma avaliação em duas etapas se faz necessária: Divisão da base de exemplos em base de treinamento e base de teste. Na primeira fase, o modelo é gerado a partir da base de treinamento. Na segunda, os dados não conhecidos pelo modelo são apresentados para avaliação do desempenho. Técnicas como a validação cruzada podem ser usadas para avaliar o modelo através de sucessivas iterações. Na validação cruzada, os dados são divididos em k subconjuntos com tamanhos aproximadamente iguais. Os subconjuntos são treinados k vezes. Em cada iteração, um dos subconjuntos é deixado de lado, sendo usando posteriormente para a fase de testes.

2.3 Aprendizagem a partir de Árvores de Decisão

Árvores de decisão são utilizadas em diferentes áreas para realização de inferência indutiva, através da execução de algoritmos de aprendizagem de máquina. Uma árvore de decisão é gerada a partir de uma base de exemplos de treinamento, os quais são descritos utilizando um número finito de atributos. Esses atributos são utilizados para predição de um atributo meta. Uma árvore de decisão é composta pelos seguintes elementos:

- um nó de decisão, que representa um atributo a ser utilizado na predição;
- N arestas contendo testes relativos ao atributo, sendo que cada aresta é ligada a outro nó. O nó subseqüente pode ser um outro nó de decisão ou um nó folha;

- um nó folha, que é a representação do final do caminho da árvore e contém o valor do atributo meta.

A árvore de decisão pode ser usada para classificar um exemplo, iniciando no nó raiz da árvore e movendo-se através dos nós de decisão até chegar a uma folha. A Tabela 2.1 apresenta um conjunto de exemplos usado para a construção da árvore mostrada na Figura 2.1[1].

Tabela 2.1: Uma Base de Treinamento [QUI93]

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	FALSE	no
sunny	80	90	TRUE	no
overcast	83	86	FALSE	yes
rainy	70	96	FALSE	yes
rainy	68	80	FALSE	yes
rainy	65	70	TRUE	no
overcast	64	65	TRUE	yes
sunny	72	95	FALSE	no
sunny	69	70	FALSE	yes
rainy	75	80	FALSE	yes
sunny	75	70	TRUE	yes
overcast	72	90	TRUE	yes
overcast	81	75	FALSE	yes
rainy	71	91	TRUE	no

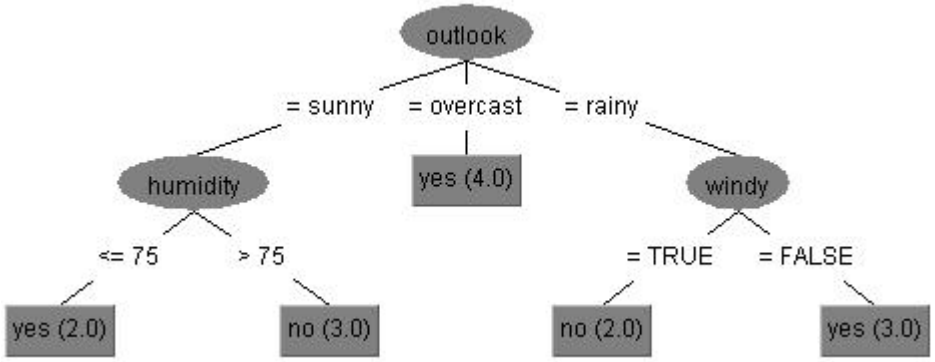

Figura 2.1: Uma Árvore de Decisão

Uma árvore de decisão é construída recursivamente através da estratégia *dividir-para-conquistar*, onde um problema complexo é dividido em problemas menores. Inicialmente, um atributo é selecionado como nó e uma divisão dos dados é criada para cada valor do atributo. O processo é repetido recursivamente para cada divisão de dados até

[1]Figura Extraída do Aplicativo Weka (Waikato Environment for Knowledge Analysis)

que todas as instâncias do subconjunto gerado tenham a mesma classe alvo. Devido a essa característica de particionamento, uma árvore é essencialmente uma conjunção de disjunções, onde cada caminho é uma conjunção dos atributos e seus testes e a árvore como um todo é uma disjunção de todas essas conjunções [SHA00]. O seguinte algoritmo descreve a sequência de passos para a criação de uma árvore de decisão:

1. Quando todos ou a maioria dos exemplos no conjunto pertencerem a uma única classe, declare o nó como sendo uma folha, representando a classe;

2. Selecione o melhor atributo usando alguma heurística e crie uma divisão do conjunto, gerando novos vértices, sendo que para cada vértice deve ser atribuído um valor. Para dados contínuos os valores devem ser atribuídos no formato $a \leq v$ e $a > v$, onde v é o valor do atributo a;

3. Continue o processo de crescimento dos nós até que todas as instâncias sejam direcionadas em alguma folha.

Para que o algoritmo gere uma árvore compacta, é necessário usar algum critério estatístico que determine qual será o atributo ideal para descrever o dado. O algoritmo C4.5 [QUI93], usa uma heurística baseada na teoria da informação para identificar o atributo que apresenta maior ganho. A escolha do atributo que será utilizado para divisão é baseado em uma propriedade estatística chamada *ganho de informação*, que estabelece o seguinte princípio: a informação presente em uma mensagem depende da probabilidade da sua ocorrência e pode ser mensurada em *bits*. Em outras palavras, antes de adicionar um nó e arestas que particionam o conjunto de exemplos, é necessário medir a entropia (ou desordem) dos dados antes e depois do particionamento. Se a quantidade de informação necessária para a classificação for menor após a ramificação, então o nível de entropia ou desordem foi reduzido, sendo o particionamento bem sucedido.

Em um primeiro momento é necessário calcular a entropia dos dados antes do particionamento. Portanto, dado um conjunto de K exemplos em um nó, C sendo a quantidade de classes e $p(K, j)$ a proporção de casos em K que pertencem a j-ésima classe, a informação é obtida a partir da fórmula apresentada em 2.1.

$$info(K) = \sum_{j=0}^{C-1} -p(K, j) * \log_2(p(K, j)) \tag{2.1}$$

O ganho da informação para cada atributo a ser testado (fórmula 2.2) é a diferença entre a quantidade de informação antes da divisão e a quantidade de informação após a divisão, sendo que m é a quantidade de subconjuntos gerados, $|Ti|$ é a quantidade de

exemplos positivos do subconjunto, $|T|$ é a quantidade de exemplos do subconjunto e K é a quantidade de exemplos da base de treinamento.

$$ganho(K) = info(K) - \sum_{i=1}^{m} \frac{|Ti|}{|T|} * Info(Ki) \tag{2.2}$$

Em certas situações a fórmula de ganho de informação é ineficiente. O problema ocorre nos casos onde o atributo possui valores únicos para os particionamentos possíveis, como por exemplo um código de produto ou a representação extraída da impressão digital de uma pessoa. O valor máximo do ganho de informação é gerado para esses atributos, fazendo com que a base seja dividida em vários segmentos contendo apenas um exemplo no pior caso. Para corrigir esse problema, o critério de taxa de ganho usa o grau de entropia gerado pela divisão da base de exemplos para normalizar o valor do ganho. Essa técnica é chamada de ganho médio. Primeiramente é calculado o ganho da divisão (fórmula 2.3)

$$ganho\ da\ divisao(X) = -\sum_{i=1}^{m} \frac{|Ti|}{|T|} * \log_2(\frac{|Ti|}{|T|}) \tag{2.3}$$

Finalmente, para obter o ganho médio é usada a fórmula 2.4.

$$ganho\ medio(X) = \frac{ganho(X)}{ganho\ da\ divisao(X)} \tag{2.4}$$

Para classificar perfeitamente os dados de treinamento, uma árvore pode produzir um efeito de superadaptação. A árvore pode ficar demasiadamente ajustada aos exemplos de treinamento e ineficiente na predição de dados desconhecidos. Faz-se necessário uma técnica de generalização para aumentar o poder de predição do modelo. A poda é uma técnica utilizada para minimizar esse problema. A poda consiste em transformar um nó candidato em uma folha, sendo que este corresponde à classe mais comum encontrada nos exemplos de treinamento cobertos pelo nó. Ela pode ocorrer basicamente de duas maneiras:

- Pré-poda: ocorre em tempo de construção do classificador e utiliza algum critério para evitar a subdivisão desnecessária do conjunto;
- Pós-poda: é aplicada após a criação da árvore completa. Consiste em simplificar a árvore, transformando nós de decisão em folhas, baseado na classe mais freqüente.

A pré-poda apresenta melhor desempenho já que evita o crescimento da árvore com ramos desnecessários. Porém a técnica de pós-poda é a mais utilizada e trabalhos mostram que ela apresenta melhores resultados [QUI93].

2.4 Aprendizagem a partir de Regras

Apesar das árvores de decisão se apresentarem compreensíveis por seres humanos como um conjunto de hipóteses, o tamanho da árvore é decisivo para a extração de informações relevantes sobre o modelo. Como o último nó de um caminho completo da árvore é uma folha e depende do caminhamento pelos diversos nós adjacentes, a leitura pode ficar prejudicada. Regras de classificação, tornam-se visivelmente mais atraentes por se apresentarem em um formato $se < cond > então < alvo >$, onde $< cond >$ é uma conjunção de atributos e testes e $< alvo >$ é a classe correspondente ao padrão. Além disso os sistemas de indução de regras tem alto valor para aplicações que necessitam de precisão e compreensibilidade dos modelos gerados [LS95]. A Figura 2.2 apresenta um conjunto de regras gerado a partir dos exemplos apresentados na Tabela 2.1.

$$R1 = \begin{cases} \text{se} & outlook = sunny \\ \text{e} & humidity \leq 75 \end{cases} \rightarrow play = yes$$

$$R2 = \begin{cases} \text{se} & outlook = sunny \\ \text{e} & humidity > 75 \end{cases} \rightarrow play = no$$

$$R3 = \{ \text{se} \quad outlook = overcast \rightarrow play = yes$$

$$R4 = \begin{cases} \text{se} & outlook = rainy \\ \text{e} & windy = true \end{cases} \rightarrow play = no$$

$$R5 = \begin{cases} \text{se} & outlook = rainy \\ \text{e} & windy = false \end{cases} \rightarrow play = yes$$

Figura 2.2: Regras de Classificação

Com o objetivo de padronização, será adotada a representação $Body \rightarrow Head$ ou resumidamente $B \rightarrow H$ para uma regra de classificação. Onde B é a conjunção de pares de atributos e valores e H é a classe associada. Quando as regras de classificação são geradas a partir de árvores de decisão, B é todo o caminho percorrido do nó raíz da árvore até a folha, onde os nós são convertidos em atributos, as arestas em testes e H é a folha (a classe correspondente ao padrão especificado nas condições). As regras obtidas diretamente a partir de uma árvore de decisão são mutuamente exclusivas, já que um caminho da árvore de decisão é uma disjunção de uma conjunção. A vantagem dessa técnica é que uma regra pode ser avaliada de forma independente e não haverá conflito entre regras. Entende-se por conflito, regiões de dados que são cobertas por regras que prevêem valores diferentes para o atributo meta. A desvantagem desta técnica é que pode ser gerado um conjunto extenso de regras dependendo do espaço de dados a ser

percorrido. Para reduzir esse efeito, algoritmos executam otimizações e podas nas regras com o objetivo de generalização.

A maneira pela qual um algoritmo avalia novos dados vai depender da maneira que as regras foram geradas. Se foram geradas diretamente a partir de uma árvore de decisão, a avaliação é análoga ao estilo *top-down da árvore*: as regras podem ser avaliadas em qualquer ordem, já que podem ser entendidas como um caminho completo da árvore. Certos tipos de regras devem obedecer um critério de ordenação pré-estabelecido. Essa heurística depende do algoritmo que gerou a regra. Com o objetivo de aprofundar o conhecimento sobre regras de classificação, a próxima seção apresenta algumas medidas de avaliação, critérios de seleção e ordenação de regras.

2.4.1 Medidas de Avaliação de Regras

Como o objetivo da pesquisa é a integração de conjuntos de regras, há uma necessidade de adotar técnicas para avaliar as regras. As medidas devem fornecer informações sobre precisão, qualidade e grau de interesse do conhecimento induzido. Várias medidas de avaliação de regras foram pesquisadas com o objetivo de auxiliar a compreensão e poder de predição dos modelos simbólicos. O trabalho de Nada Lavrac et. al. [LFZ99] apresenta uma compilação de diversas medidas de avaliação de regras, que será apresentado adiante. Antes, é importante apresentar a matriz de contingência, a qual produz as informações necessárias para obtenção das medidas de avaliação.

A matriz de contingência é uma generalização da matriz de confusão, que é a base padrão para calcular medidas de avaliação de hipóteses em problemas de classificação [PBM02]. A diferença fundamental é que a matriz de contingência tem uma relação de $1:1$ com as regras, enquanto que a matriz de confusão tem uma relação $1:N$, sendo que N é a quantidade de regras do classificador, funcionando como uma caixa preta para todo o modelo. A Tabela 2.2 apresenta uma matriz de contingência para uma regra R no formato B $\rightarrow$ H.

Tabela 2.2: Matriz de Contingência de uma Regra R: B $\rightarrow$ H

	B	¬B	
H	hb	h¬b	h
¬H	¬hb	¬h¬b	¬h
	b	¬b	n

Da matriz de contingência é possível extrair as seguintes informações:

- hb é a quantidade de exemplos onde tanto H quanto B são verdadeiros;

- h¬b é a quantidade de exemplos onde H é verdadeiro e B é falso;
- ¬hb é a quantidade de exemplos onde H é falso e B é verdadeiro;
- ¬h¬b é a quantidade de exemplos onde H e B são falsos;
- b é a quantidade de exemplos onde B é verdadeiro;
- ¬b é a quantidade de exemplos onde B é falso;
- h é a quantidade de exemplos onde H é verdadeiro;
- ¬h é a quantidade de exemplos onde H é falso;
- n é a quantidade total de exemplos.

Também é possível criar uma matriz de contingência com freqüências relativas, onde há uma normalização dos exemplos através de n, funcionando como uma probabilidade da ocorrência do evento $f(e) = \frac{e}{n}$ também denotado fe. Na Tabela 2.3 é apresentada a matriz de contingência com freqüências relativas. Com a matriz de contingência é possível obter os dados necessários para realizar a maioria das medidas propostas na literatura.

Tabela 2.3: Matriz de Contingência com Freqüências Relativas para uma Regra R: B → H

	B	¬B	
H	$f$hb	fh¬b	fh
¬H	f¬hb	f¬h¬b	f¬h
	b	¬b	1

Abaixo, estão relacionadas algumas medidas de avaliação de regras presentes em [LFZ99].

- Precisão (2.5): é a medida que indica a probabilidade de H e B serem verdadeiras.

$$Prec(R) = P(H|B) = \frac{P(HB)}{P(B)} = \frac{fhb}{fb} \tag{2.5}$$

- Erro (2.6): é obtido através de $1 - Prec(R)$. Quanto maior o erro, menos precisa é a regra.

$$Err(R) = 1 - Prec(r) = P(\neg H|B) = \frac{f\neg hb}{fb} \tag{2.6}$$

- Confiança Negativa (2.7): corresponde à medida de precisão para exemplos não cobertos pela regra.

$$NegRel(R) = P(\neg H|\neg B) = \frac{P(\neg H \neg B)}{P(\neg B)} = \frac{f\neg h\neg b}{f\neg h} \tag{2.7}$$

- Sensitividade (2.8): é a probabilidade condicional de B ser verdade dado que H é verdade. Quanto maior a sensitividade, mais exemplos são cobertos pela regra.

$$Sens(R) = P(B|H) = \frac{P(HB)}{P(H)} = \frac{fhb}{fh} \tag{2.8}$$

- Especificidade (2.9): é a probabilidade condicional de B ser falso dado que H é falso.

$$Spec(R) = P(\neg B|\neg H) = \frac{P(\neg H \neg B)}{P(\neg H)} = \frac{f\neg h\neg b}{f\neg h} \tag{2.9}$$

- Cobertura (2.10): é a probabilidade de B ser verdade. Quanto maior a cobertura, maior a quantidade de exemplos cobertos pela regra.

$$Cov(R) = P(B) = fb \tag{2.10}$$

- Suporte (2.11): é a probabilidade de H e B serem verdade. Quanto maior o suporte, maior a quantidade de exemplos da classe serão cobertas pela regra.

$$Sup(R) = P(HB) = fhb \tag{2.11}$$

- Novidade (2.12): é a probabilidade de B e H serem estatisticamente dependentes.

$$Nov(R) = P(HB) - P(H)P(B) = fhb - fh.fb \tag{2.12}$$

- Satisfação (2.13): é uma medida que indica se há aumento relativo na precisão entre a verdade da regra B e a regra B $\rightarrow$ H. É uma medida indicada para tarefas voltadas à descoberta de conhecimentos, sendo capaz de promover um equilíbrio entre regras com diferentes condições e conclusões.

$$Sat(R) = \frac{P(\neg H) - P(\neg H|B)}{P\neg H} = \frac{f\neg h - \frac{f\neg hb}{fb}}{f\neg h} \tag{2.13}$$

- Precisão Relativa (2.14): mede o ganho de precisão em relação a uma regra *default*.

$$RPrec(R) = P(H|B) - P(H) = \frac{fhb}{fb} - fh \tag{2.14}$$

- Confiança Negativa Relativa (2.15): é similar à precisão relativa, mas usa uma regra *default* falsa.

$$RNegRel(R) = P(\neg H|\neg B) - P(\neg H) = \frac{f\neg h\neg b}{f\neg b} - f\neg h \tag{2.15}$$

- Sensitividade Relativa (2.16): mede o ganho de sensitividade em relação a uma regra *default*.

$$RPrec(R) = P(B|H) - P(B) = \frac{fhb}{fh} - fb \tag{2.16}$$

- Especificidade Relativa (2.17): é similar à sensitividade relativa, mas usa uma regra *default* falsa.

$$RSpec(R) = P(\neg B|\neg H) - P(\neg B) = \frac{\neg h\neg b}{f\neg h} - f\neg b \tag{2.17}$$

2.4.2 Interpretação de Regras

Árvores de decisão dividem o espaço de exemplos em regiões disjuntas. Cada exemplo é classificado por apenas um único ramo da árvore [PBM02]. Transformar uma árvore em regras sem que ocorra um processo de generalização ou otimização tem o mesmo efeito. A ordem das regras é irrelevante. A Figura 2.3 apresenta uma árvore de decisão de duas classes (+ e −) e dois atributos ($x1$ e $x2$) e a interpretação geométrica correspondente.

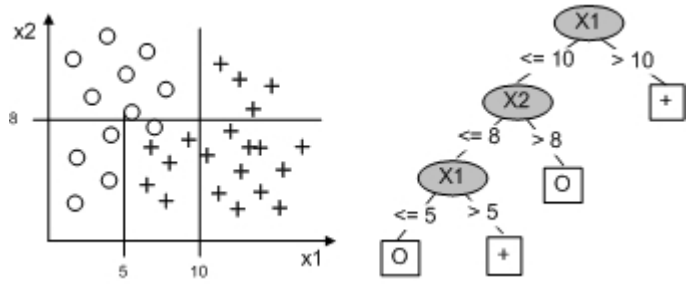

Figura 2.3: Interpretação Geométrica para uma Árvore de Decisão [PBM02]

Já em regras de classificação, a forma de aplicação das regras vai depender da maneira com que as mesmas foram construídas. É possível classificar as regras em dois tipos:

- Regras não ordenadas: Regras mutuamente exclusivas são chamadas de regras não ordenadas, porque apenas uma delas cobrirá um dado exemplo. Todas as regras podem ser utilizadas na avaliação de um exemplo. Regras geradas diretamente a partir de uma árvore de decisão se enquadram nessa classificação;

- Regras ordenadas: Regras que não são mutuamente exclusivas e podem apresentar conflitos, ou seja, mais de uma regra pode cobrir o exemplo. Neste caso a ordem de avaliação das regras é fundamental, já que algum critério de ordenação, baseado na especificidade ou desempenho, por exemplo, pode ter sido utilizado durante a geração das regras.

Na Figura 2.4 é apresentada uma interpretação geométrica, para um conjunto de três regras não ordenadas e duas classes, + e ∘.

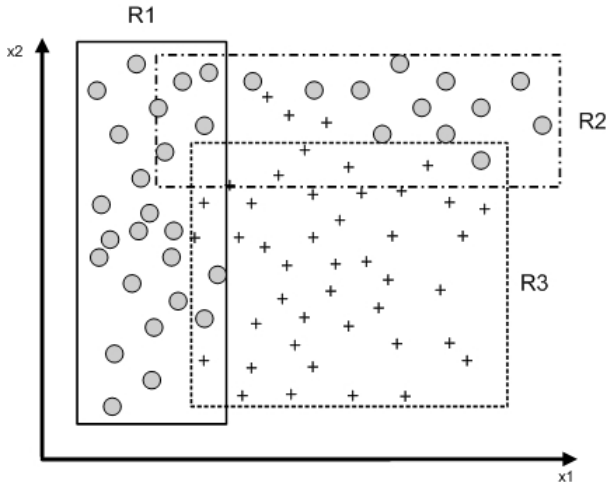

Figura 2.4: Interpretação Geométrica para um Conjunto de Regras não Ordenadas [PBM02]

2.4.3 Classificação e Seleção de Regras

Regras ordenadas dependem diretamente do mecanismo de ordenação e seleção, já que não são mutuamente exclusivas. Coenen e Leng [CL04] apresentam 5 critérios de ordenação de regras, a saber:

- CSA (Confiança, Suporte e Antecedentes): As regras são ordenadas de acordo com os indicativos de confiança, suporte e quantidade de antecedentes da regra. Baseado na matriz de contingência, a medida de confiança pode ser obtida a partir da fórmula $\frac{fhb}{b}$. A medida de suporte é dada pelo indicador fhb da matriz de contingência e finalmente, a medida do antecedente é obtida a partir da quantidade de condições

presentes em B ou as condições da regra. Coenen e Leng [CL04] mencionam que a probabilidade do fator de confiança ser igual para duas regras é muito baixo. Portanto, os indicativos de suporte e quantidade de antecedentes raramente são utilizados;

- WRA (Precisão Relativa com Peso): é um mecanismo unificado para medir regras. O termo "relativo" indica que a expectativa do suporte é utilizada para normalizar o valor desta medida. Esta medida foi criada especificamente para mecanismos de ordenação de regras que refletem várias medidas de avaliação em uma só. A idéia é que WRA seja uma síntese da "interessabilidade" de uma regra. Usando a matriz de contingência, é possível obter WRA com a fórmula $fb(\frac{fhb}{fb} - fh)$;

- Medida de precisão de Laplace: é utilizada por alguns algoritmos para ordenação de regras. Esta medida leva em consideração os exemplos, corretos, incorretos e a quantidade de classes presentes no problema. Quanto maior é a precisão de Laplace, melhor é a regra. Utilizando a matriz de contingência, a fórmula para a obtenção da medida de Laplace é $\frac{fbh+1}{fbh+fb\neg h+k}$, onde k é o número de classes;

- X^2 ou Chi quadrado: é uma medida estatística usada para determinar se duas variáveis são independentes através da comparação de valores observados e esperados. A fórmula é obtida a partir de $x^2 = \sum_{i=1}^{n} \frac{(Oi-Ei)^2}{Ei}$, onde n é o número de valores observados/esperados. Se o valor resultante for menor do que um limiar estipulado, então pode se afirmar que há uma relação entre os valores esperados e observados, caso contrário não há. Quanto maior o valor, maior o grau de independência da regra;

- ACS (Antecedentes, Confiança e Suporte): A proposta é uma alternativa a CSA, uma vez que dá prioridade a regras mais específicas.

Adicionalmente, os autores [CL04] apresentam 3 formas de seleção de regras (*firing*):

- Melhor regra: Parte do princípio de que apenas uma regra pode classificar um exemplo. Dado um exemplo, o algoritmo percorrerá o conjunto de regras e aquela que se ajustar aos atributos do exemplo será utilizada e as demais regras serão desprezadas;

- Melhores K regras: O algoritmo selecionará as K "melhores" regras do classificador que satisfaçam a condição apresentada, onde K é um parâmetro estipulado. Através de um critério de votação, a classe vencedora será selecionada;

- Todas as regras: O algoritmo selecionará todas as regras do classificador que satisfaçam a condição apresentada e um critério de desempate baseado em X^2 é utilizado.

Os testes realizados no trabalho de Coenen e Leng [CL04] utilizando combinações dos diferentes critérios de ordenação e seleção indicaram que não houve um critério de ordenação que se sobressai em relação aos outros. Quanto à seleção, o critério "melhor regra" apresentou melhor desempenho.

O algoritmo C4.5rules [QUI93], que gera regras de classificação a partir de uma árvore de decisão usa um método de ordenação de regras baseado no princípio MDL (*Menor Tamanho de uma Descrição / Minimum Description Length*). Este princípio será descrito na próxima seção.

2.4.4 Codificação de Regras Usando MDL

O princípio MDL pode ser explicado como um modelo de comunicação no qual um processo emissor transmite para um receptor a descrição de uma teoria T e o dado D do qual ele é derivado [QR89]. A descrição do tamanho da mensagem obtida consiste no custo necessário para descrever um dado. O princípio MDL diz que a melhor teoria derivada de um conjunto de exemplos vai minimizar a quantidade de *bits* necessários para codificar a mensagem completa, que consiste na teoria juntamente com suas exceções [QUI95]. Trazendo o conceito MDL para o problema de avaliação do conjunto de regras, uma teoria é representada pelo conjunto de regras, o dado é representado pela base de treinamento e a mensagem é a regra propriamente dita. Da mesma forma que o algoritmo C4.5 [QUI93], as regras são agrupadas pela cabeça da regra (H) ou a classe, sendo criado k subconjuntos de regras. A quantidade de informação então é calculada para cada regra pertencente à classe. Ao final é possível calcular a quantidade de informação para o subconjunto todo. Nos parágrafos a seguir são apresentadas as etapas para realizar o cálculo da quantidade da informação do subconjunto.

- Para codificar uma regra, é necessário especificar cada condição presente no corpo da regra (B). A cabeça da regra não precisa ser codificada já que todas as regras no subconjunto pertencem a mesma classe. A quantidade de 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, subtraindo um crédito similar para a ordenação das regras;

- As exceções são codificadas indicando quais dos casos cobertos pela regra S 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 dada pela fórmula 2.18.

$$excecao = log_2(\frac{r}{fp}) + log_2(\frac{n-r}{fn}) \tag{2.18}$$

  O primeiro termo refere-se à quantidade de *bits* necessários para indicar os falsos positivos entre os casos cobertos pela regra e o segundo termo é uma expressão similar, indicando os falsos negativos entre os casos não cobertos;

- O custo de um subconjunto é mensurado pela soma da codificação das regras e suas exceções. Quanto menor a soma, melhor a teoria representada por S.

2.5 Considerações Finais

Neste capítulo foram apresentados alguns conceitos de mineração de dados e aprendizagem simbólica, com ênfase nos métodos de avaliação, classificação e seleção de regras. A maioria dos conceitos pode ser aplicada em qualquer conjunto de regras, desde que se obedeça a característica principal dos classificadores. É importante questionar: o conjunto de regras é ordenado ou não? Qual é o mecanismo de classificação (*best-first*, voto, *best-k-first*)? Uma vez que o trabalho proposto não é voltado para nenhum algoritmo em especial, estes conceitos formaram uma base importante para o desenvolvimento do trabalho.

Capítulo 3

Mineração Distribuída de Dados

Muitos algoritmos estatísticos de mineração de dados, reconhecimento de padrões e aprendizagem de máquina requerem que todos os dados a serem analisados estejam em memória, podendo falhar já que nem sempre o espaço de memória é suficiente. A mineração distribuída de dados é uma fusão das técnicas de mineração de dados e sistemas distribuídos que viabiliza a aplicação de mineração em diversas áreas. Aplicações das mais diversas naturezas usam técnicas de mineração distribuídas, tais como fusão de informação, mineração de dados, detecção de intrusão e descoberta de padrões. Especialmente a atividade de classificação tem recebido diversas contribuições [DIE97] [TD00] [PC00] [HK00] [OCA01] [PASE06] [BMP06] [BGB07].

Neste capítulo será apresentada uma definição sobre mineração distribuída de dados e mecanismos de combinação de classificadores simbólicos são discutidos.

3.1 Definição

DDM (*Mineração Distribuída de Dados / Distributed Data Mining*) é a associação de técnicas de mineração de dados com sistemas distribuídos que permite o uso do paralelismo e distribuição de tarefas entre diversos processadores. Segundo Freitas e Lavington [FL98], a atividade de mineração distribuída de dados pode ser dividida em três fases:

1. Dividir a base de exemplos em p subconjuntos, onde p é o número de processadores. Cada subconjunto é enviado para um processador;

2. Executar em cada processador um algoritmo de mineração de dados em seu ambiente local; os processadores podem executar o mesmo algoritmo de mineração de dados ou diferentes algoritmos;

3. Combinar o conhecimento local descoberto por cada algoritmo de mineração de dados em um modelo global e consistente.

Os autores ainda discutem que esse conhecimento global é diferente do conhecimento descoberto se o algoritmo for aplicado no conjunto inteiro formado pela união dos subconjuntos de dados individuais. O classificador tende a ficar menos preciso a medida que a quantidade de classificadores aumenta e a quantidade de dados em cada subconjunto diminui.

A Figura 3.1 ilustra o processo, onde os dados são representados por retângulos, os algoritmos por elipses e o conhecimento obtido por triângulos.

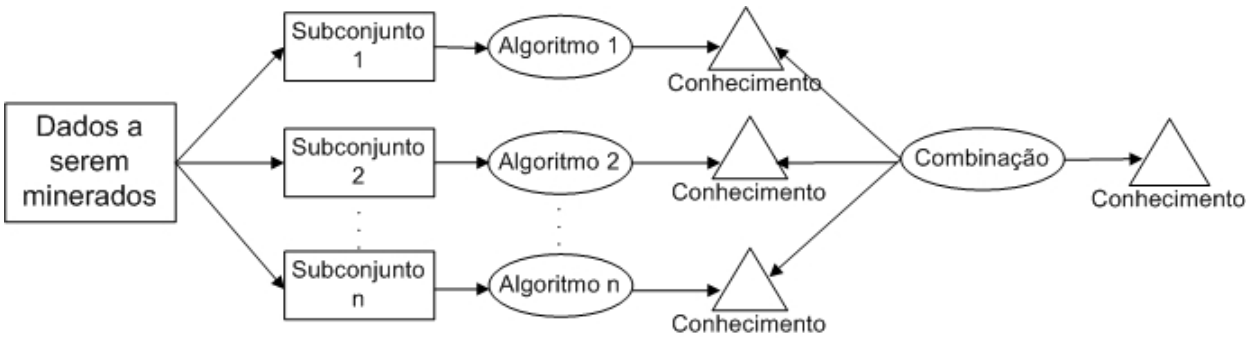

Figura 3.1: A Estrutura Básica de uma Mineração Distribuída de Dados [FL98]

Na terceira fase do DDM, citada acima, o conhecimento local descoberto pelos processadores pode ser combinado de diferentes maneiras. Uma das maneiras mais triviais é através da técnica de votação simples. Esse processo pode ser utilizado para resolver diferentes tipos de problemas, como por exemplo:

- Extensibilidade: Suportam a inclusão de novas tecnologias de mineração;
- Portabilidade: Capacidade de operar em diferentes ambientes ou plataformas;
- Escalabilidade: Eficiência em minerar grandes volumes de dados sem perda de qualidade;
- Eficiência: Determina a capacidade de utilizar corretamente os recursos disponíveis;
- Compatibilidade: Integração de informações para bases de dados similares, mas com diferentes esquemas, para gerar modelos de dados mais precisos.

Muitas pesquisas têm sido feitas no desenvolvimento de algoritmos com o objetivo de se produzir técnicas escaláveis e computacionalmente mais eficientes de mineração de grandes conjuntos de dados. A seguir são apresentados alguns algoritmos de mineração distribuída de dados desenvolvidos com esse objetivo.

3.2 Voto

Um método dinâmico de combinação e resolução de conflitos é a estratégia de voto. Esta técnica consiste em aplicar um exemplo desconhecido em diferentes classificadores. A classe associada a esse exemplo será aquela apontada pela maioria dos classificadores em tempo de execução [PC00].

De acordo com Chan e Stolfo [CS95a], uma das variações da técnica de voto é o voto ponderado, que associa um peso determinado pela precisão do classificador sobre a base de treinamento. Os pesos dos votos são somados e uma instância de teste é classificada de acordo com a classe com maior peso obtido.

Shawla [SHA00] apresenta alguns experimentos usando a técnica de voto simples e ponderado aplicado diretamente a regras de classificação. Duas medidas foram associadas às regras: O fator de certeza e a precisão. Os resultados apresentaram melhora sensível na predição, utilizando o voto ponderado em relação ao voto simples.

3.3 Multi-esquema

Segundo Enembreck e Ávila [EV06], uma estratégia simples de combinação é a seleção de um classificador dentre diversos. O desempenho de um classificador pode ser medido no conjunto de dados a partir do percentual de classificação considerada correta. Esta técnica consiste na escolha de um classificador que possui a melhor qualidade dentre um conjunto de classificadores que usam amostras do conjunto completo. Essa técnica é demasiadamente suscetível a resultados de baixa qualidade, pois a construção de classificadores simbólicos é fortemente influenciada pelo método de amostragem, mesmo que as distribuições sejam semelhantes. Isso favorece a superadaptação do modelo a uma amostra específica.

3.4 Meta-aprendizagem

O conceito de meta-aprendizagem é inspirado na estratégia de *stacking*, criada por Wolpert em [WOL90]. A idéia geral é minimizar a taxa de erro de classificadores, transformando as predições dos classificadores em instâncias de treinamento, utilizadas para a geração de um novo classificador. Chan e Stolfo [CS95a] [CS95b] propuseram técnicas de meta-aprendizagem inspiradas na estratégia de *stacking*. As técnicas consistem em combinar N classificadores e utilizá-los como elemento de entrada para outro classificador. Cada algoritmo de aprendizagem local é tratado como uma caixa preta sendo possível

combinar diferentes algoritmos de aprendizagem. Os classificadores locais são chamados de classificadores base. A tarefa de combinação consiste em utilizar as predições dos classificadores base e transformá-las em uma base de treinamento meta-nível. Esses dados são utilizados para a geração de um meta-classificador, que representa um modelo unificado.

A construção de meta-classificadores pode ocorrer diversas vezes sendo possível criar uma hierarquia de classificadores. A meta-aprendizagem pode ser resumida em 4 etapas:

- classificadores base são treinados usando bases de exemplos locais;
- as predições são geradas pelos classificadores em uma base de validação separada;
- uma base de meta-treinamento é montada a partir das predições dos classificadores base;
- um meta-classificador é treinado usando a base de meta-treinamento.

Chan e Stolfo [CS95a] apresentaram duas técnicas para combinar as predições dos classificadores base, chamadas árbitro e combinador.

Na técnica do árbitro, é criado um classificador especial, chamado árbitro, que irá decidir a classe final de predições para uma dada entrada. O árbitro gera um classificador executando um algoritmo de aprendizagem sobre instâncias difíceis de classificar com os classificadores base. A classificação é feita sobre a predição dos algoritmos de aprendizagem locais e a predição do árbitro. Estas predições são combinadas, retornando a maioria de ocorrências, com preferência dada às predições do árbitro em caso de empate.

Já no caso do combinador, o meta-classificador recebe como entrada as predições realizadas por cada um dos classificadores base juntamente com as predições corretas presentes em cada base de treinamento local. Outras informações tais como os valores dos atributos, também podem ser adicionadas ao conjunto de dados a ser utilizado na meta-classificação, dependendo da estratégia adotada na implementação da meta-aprendizagem. A meta-aprendizagem usa estes dados para descobrir a relação entre as predições feitas pelos algoritmos locais e as predições corretas.

Enembreck e Ávila [EV06] apresentam um modelo de meta-aprendizagem para integração de classificadores simbólicos chamado KNOMA. O processo consiste em usar como entrada N classificadores base gerados a partir de bases distribuídas. Os conjuntos de regras são unificados gerando um único conjunto de regras. Baseado nesse conjunto, é gerada uma meta-base de treinamento em uma etapa preliminar, sendo que cada meta-instância é a representação de uma regra e os atributos deste conjunto são os antecedentes

(B) da regra. A classe da meta-instância é dada pela classe ou cabeça (H) da regra. Um meta-classificador definitivo é gerado usando essa base.

Segundo Chan e Stolfo [CS95a], o uso de classificadores com diferentes tendências criam um classificador mais preciso. 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 e Lavington [FL98] afirmam que a precisão da predição conseguida com técnicas de meta-aprendizagem tende a diminuir quando o número de subconjuntos de dados aumentar, a menos que ocorra um aumento na quantidade de dados contidos em cada subconjunto. Além disso, técnicas de meta-aprendizagem podem reduzir a compreensibilidade do conhecimento descoberto.

3.5 Combinação de Classificadores Simbólicos

A combinação de classificadores consiste na unificação de classificadores aprendidos a partir de bases de dados disjuntas, gerando um modelo consistente. Muitas pesquisas tem sido realizadas, com o objetivo de criar técnicas de unificação de classificadores.

Provost e Hennessy [PH96] demonstraram que qualquer regra que tenha um desempenho aceitável em uma base de exemplos integral, terá um desempenho aceitável em pelo menos um fragmento da mesma base. A esse fenômeno é dado o nome de "Propriedade de invariância de particionamento". Isto sugere que um conjunto de regras gerado através de classificadores em bases disjuntas conterá as mesmas regras presentes no conjunto completo. Em um trabalho de Provost e Hennessy [PH96] foram identificadas as mesmas regras presentes no conjunto de regras gerado a partir da base integral, além de outras novas regras. Entretanto no trabalho de Hall et. al. [HCBK99] é apresentado um cenário onde foram utilizados três classificadores: um classificador para uma base de exemplos integral e dois classificadores na base dividida. Como resultado, as bases disjuntas não apresentaram nenhuma das regras presentes na base completa. Isto ocorreu porque o classificador (C4.5) utiliza o ganho de informação para escolha do atributo de testes e é altamente dependente da base de treinamento (algoritmo instável). Portanto, o algoritmo utilizado no classificador tem forte influência na técnica utilizada para a integração e resolução dos conflitos. Além disso, cada algoritmo contém um *bias* explícito ou implícito que tende a favorecer certas generalizações em detrimento de outras: o ponto forte de um pode ser o fraco de outro [DIE89] [SB02]. Em geral, a combinação de indutores incrementa a precisão reduzindo o *bias*. O objetivo da integração é diminuir as limitações de técnicas individuais, através da hibridização ou fusão de várias técnicas [SB05].

Hall et. al [HCBK99] propõem um método de integração de conjuntos de regras geradas a partir de bases de dados disjuntas. Dados N conjuntos de treinamento, são gerados N conjuntos de regras em processadores distintos. Ao final, é gerado um modelo livre de conflitos e com confiabilidade equivalente a um conjunto de regras geradas a partir da base de treinamento. Cada regra terá uma medida de desempenho, baseada na precisão e quantidade de exemplos cobertos pela regra. Regras que tenham desempenho menor do que um limiar são descartadas. Algumas características relevantes da técnica:

- nenhuma regra é eliminada sem ter sido avaliada por todos os subconjuntos;
- regras que indicam a mesma classe são simplificadas, eliminando uma das regras e mantendo a mais abrangente;
- regras que se apresentarem muito próximas de um limiar são especializadas. A especialização consiste em recuperar os exemplos cobertos pela regra e prolongar a árvore criando uma regra especializada.

O método apresentado por Hall mostrou desempenho comparável em termos de confiabilidade em relação ao classificador obtido a partir da base de dados integral.

No trabalho de Bernardini [BER02], foi desenvolvido um método de combinação de classificadores simbólicos, que consiste em selecionar regras de classificadores previamente construídos por algoritmos de aprendizagem e compor um classificador final. O critério de seleção das regras é baseado em medidas de avaliação de regras. Apesar de identificar boas regras dentro do conjunto, de acordo com um especialista, as taxas de erro ficaram altas quando comparadas com os classificadores base.

O trabalho de Hamberger e Lavrac [GL00] [GLK02] introduz uma abordagem de tomada de decisões baseada em consenso. Basicamente é composta pelas seguintes etapas:

- é gerado um conjunto de regras que apontam para a mesma classe, sendo mais próxima do processo de decisão feito por humanos;
- para que a regra seja incluída é necessário um valor mínimo de suporte;
- é apresentado um modelo de decisão no qual diferentes regras podem ser incorporadas. Estas regras podem ser geradas por diferentes classificadores ou codificadas por seres humanos;
- o classificador recusa-se a classificar um exemplo se as regras entram em conflito ou nenhuma regra for disparada.

Este modelo é recomendado nas situações onde o erro de uma previsão não é permitido. Por outro lado perde na completeza já que alguns exemplos não são cobertos.

3.6 Considerações Finais

Neste capítulo foram apresentados alguns conceitos sobre mineração distribuída de dados e técnicas de integração de conhecimentos. No próximo capítulo será apresentada uma definição de inteligência artificial distribuída, suas subdivisões e técnicas de resolução distribuída de problemas. Os conceitos de IAD (*Inteligência Artificial Distribuída*) são elementos importantes para o trabalho, considerando que o problema apresentado ocorre em um ambiente distribuído e a metodologia é inspirada em modelos de resolução distribuída de problemas.

Capítulo 4

Inteligência Artificial Distribuída

Nas próximas seções são apresentados alguns conceitos sobre inteligência artificial distribuída, agentes de software, sistemas multi-agente, resolução distribuída de problemas e alguns mecanismos de coordenação e resolução de conflitos. As técnicas de IAD formam a base para o desenvolvimento do método de combinação de modelos em um ambiente distribuído discutido nos capítulo 5.

4.1 Definição

IAD é um ramo da IA (*Inteligência Artificial*) focada no desenvolvimento de princípios computacionais e modelos para construir, descrever, implementar e analisar os padrões de interação e coordenação, tanto para pequenas quanto grandes sociedades de agentes [LES99a].

Segundo Bond e Gasser [BG88], a IAD pode ser dividida em duas áreas fundamentais de pesquisa: RDP (*Resolução Distribuída de Problemas*) e SMA (*Sistemas Multi-Agente*). Segundo Jennings et. al. ([JSW98]), a RDP descreve como um problema em particular pode ser resolvido por um grupo de módulos (*nós*), que cooperam através da divisão e compartilhamento do conhecimento acerca de um problema e da sua solução. Em um sistema RDP puro, todas as estratégias de interação são incorporadas como parte integral do sistema. Por outro lado, a pesquisa em SMA está concentrada no comportamento individual de uma coleção de agentes em um ambiente, com o objetivo de resolver um certo problema. Sob uma perspectiva reativa, um SMA pode ser definido como uma rede de solucionadores de problemas com baixo acoplamento, que trabalham juntos para resolver problemas que estão acima das capacidades ou conhecimento de cada solucionador [DL89].

Bond e Gasser [BG89] enumeram vários benefícios ao utilizar IAD para a resolução

de problemas:

- Adaptabilidade: sistemas modelados em IAD se encaixam com a realidade de problemas distribuídos em termos espaciais, lógicos, temporais ou semânticos;
- Custo: pequenas unidades computacionais podem ser mais eficientes quando os custos de comunicação não são relevantes;
- Desenvolvimento e Gerenciamento: a inerente modularidade do sistema permite o desenvolvimento das partes do mesmo de forma separada e paralela;
- Eficiência e Velocidade: a concorrência e a distribuição dos processos em diferentes máquinas pode aumentar a velocidade de processamento haja visto o paralelismo na execução das tarefas.
- Integração: a integração de recursos distribuídos e até mesmo heterogêneos tais como hardware e software de diferentes plataformas;
- Isolamento/Autonomia: o controle de processos locais pode ser interpretado como uma maneira de proteção ou de aumento da segurança do sistema;
- Naturalidade: alguns problemas são naturalmente melhor resolvidos através de uma configuração distribuída;
- Confiabilidade: os sistemas distribuídos podem exibir um grau maior de confiabilidade e de segurança quando comparados aos sistemas centralizados, pois podem prover redundância de dados e múltiplas verificações.
- Limitações de Recursos: os agentes computacionais individuais ligados a recursos escassos podem, através da cooperação, superar limites técnicos;
- Especialização: cada agente pode ter um papel bem definido na resolução do problema.

As duas principais áreas da IAD (SMA e RDP) diferem na forma de construção da solução do problema, mas que têm em comum entidades chamadas agentes. A Resolução Distribuída de Problemas adota uma visão *top-down*, dividindo o problema em partes que corresponderão a módulos computacionais, sendo o processo de coordenação das ações definido ainda em tempo de projeto. Os sistemas multi-agente são compostos por entidades computacionais, denominadas agentes, com capacidades e objetivos individuais que, uma vez agrupados em sociedade, trabalham juntos visando atingir o objetivo do sistema,

sendo que os agentes devem raciocinar a respeito das ações e do processo de coordenação em si.

Quanto ao termo autonomia citado como uns dos benefícios da IAD, Castelfranchi e Facone em [CF98] detectam vários níveis de autonomia, uma vez que este é um conceito que está diretamente associado a relação que um agente tem com os outros agentes. Ele pode ter níveis de influência sobre os outros.

Segundo os autores, os seguintes itens devem ser satisfeitos para que o agente seja completamente autônomo:

- O agente tem os seus próprios objetivos e não são derivados dos objetivos dos outros;
- O agente é capaz de tomar decisões sobre objetivos que estão em conflito;
- O agente pode adotar objetivos de outros agentes de forma deliberada, para atingir seus objetivos;

4.2 Agentes

Wooldrige e Jennings [WJ95] propuseram duas maneiras de visualizar um agente: uma noção forte e uma noção fraca. A noção fraca define um agente como um sistema baseado em hardware ou mais especificamente em software dotado das propriedades de autonomia, habilidade social, reatividade e pró-atividade. Já a noção forte de agente o caracteriza como uma entidade que além das propriedades acima possui características ou conceitos aplicados usualmente em seres humanos tais como crenças, conhecimento, intenção, emoção e uma interface que representa o estado desses agentes visualmente.

Coletivamente, agentes podem ser considerados elementos de software que exibem um comportamento autônomo e orientado a objetivos. Um agente pode desempenhar atividades integralmente atuando como um sistema *standalone*. No entanto, na maioria dos casos, esse elemento é modelado em um contexto multi-agente, onde um comportamento global depende da interação entre os agentes.

Zambonelli e Jennings [ZJW01] criam duas categorias de sistemas multi-agente:

- sistemas de resolução distribuída de problemas, onde o agente foi desenvolvido especificamente para trabalhar em grupo para atingir um objetivo comum;
- sistemas abertos, onde agentes não necessariamente têm um objetivo em comum e podem apresentar um comportamento competitivo.

Em sistemas abertos, o agente nem sempre é dotado das capacidades para realizar todas as ações necessárias, tornando-se dependente de outros agentes, podendo ou não ser atendido. Segundo Castelfranchi e outros autores [CdRFP98], duas teorias proporcionam níveis razoáveis de colaboração e organização: (i) A delegação e (ii) a adoção. Em (i), a colaboração entre agentes ocorre através da alocação de tarefas de um dado agente para outro agente, através de um compromisso. Quando um agente A precisa de uma ação de um agente B, a intenção é registrada no seu plano. Em (ii) um agente B tem um objetivo que também é objetivo de outro agente. Tanto (i) quanto (ii) podem ser unilaterais: B pode ignorar a delegação de A, assim como A pode ignorar a adoção de B.

No cenário proposto de mineração distribuída, um agente pode englobar todas as capacidades necessárias para um classificador local: comunicação, cooperação, múltiplos comportamentos (ora um agente pode solicitar a avaliação de um conjunto de hipóteses por outros agentes, ora ser revisor das hipóteses). Neste trabalho, será dado um maior enfoque no estudo de técnicas de resolução distribuída de problemas já que a atividade caracteriza-se mais com a tarefa de cooperação de agentes que possuem objetivos comuns e papéis bem definidos.

4.3 Sistemas Multi-Agente

SMA é uma sub-area da IAD e se concentra na modelagem e classificação de agentes individuais em um universo multi-agente. Em um SMA não é necessário que o agente seja individualmente inteligente para alcançar um objetivo globalmente inteligente. De acordo com Ferber [FER03], um SMA pode ser definido como uma aplicação distribuída composta de indivíduos independentes, heterogêneos, distribuídos e inteligentes chamados agentes, que podem cooperar entre si para a resolução de problemas complexos. De acordo com Lesser [LES99b], um sistema multi-agente é um sistema computacional no qual dois ou mais agentes interagem ou trabalham juntos para realizar uma tarefa ou para satisfazer um conjunto de objetivos.

A principal característica de um SMA é prover mecanismos para a criação de sistemas computacionais a partir de entidades independentes de software chamados agentes, os quais interagem dentro de um ambiente compartilhado entre todos os membros da sociedade, a qual provê alterações no estado deste ambiente. De acordo com Wooldrige e Jennings [WJ95], é necessário prover um mecanismo de interação e coordenação dessas entidades, já que cada uma delas possui capacidades distintas, bem como diferentes objetivos em relação aos estados esperados do ambiente em que estão inseridos. Apesar da característica de baixo acoplamento e independência dos agentes é necessário padro-

nizar o seu desenvolvimento para permitir um ambiente comum de compatibilidade e interoperabilidade no ambiente. A FIPA (*Organização para Agentes Inteligentes Físicos / Foundation for Intelligent Physical Agents*) é um órgão que tem por objetivo facilitar a comunicação entre agentes físicos. É responsável pela padronização da arquitetura, protocolos de comunicação e interação e representação dos conhecimentos [FIP07]. Vários *frameworks* de desenvolvimento de agentes são compatíveis com a especificação FIPA, como por exemplo JADE (*Ambiente de Desenvolvimento de Agentes em Java / Java Agent Development Environment*) [BCPR03] e JACK [YOS03] .

4.4 Resolução Distribuída de Problemas

RDP é uma sub-área da IAD com ênfase na utilização de grupos de agentes para resolver um problema complexo. Além das características de conhecimento, capacidade, informação e especialização, um agente é um sistema resolvedor de problemas que não é capaz de realizar suas tarefas sozinho ou pelo menos não consegue fazê-lo de forma tão precisa em comparação ao que conseguiria trabalhando junto com outros agentes [DUR99].

Resolver problemas distribuídos requer coerência e competência dos agentes. Em outras palavras, um agente pode ter um incentivo para trabalhar em grupo e deve saber desempenhar sua atividade. Problemas que utilizam RDP já possuem um grau de coerência intrínseco, já que é esperado que os agentes desenvolvidos desempenhem suas tarefas com o objetivo de um resultado comum. Portanto, as técnicas de RDP são concentradas no item competência. RDP presume a existência de um problema a ser resolvido e as expectativas de como a solução deverá ocorrer. De forma geral, os agentes designados para a solução do problema deverão formular as soluções de cada sub-problema e sintetizá-las em uma solução comum. Além da competência necessária para resolver um sub-problema, o agente precisa da competência para planejar como a resolução ocorrerá, ou seja, como os agentes farão a decomposição dos problemas em sub-problemas, alocá-los para outros agentes, compartilhar soluções dos sub-problemas e sintetizá-las em uma única solução. Portanto, a atividade de planejamento distribuído é fortemente interligada a RDP.

Segundo Durfee e Lesser [DL91], existem várias razões para construir sistemas baseados em RDP: (i) Uso do paralelismo com o objetivo de maximizar a utilização de recursos. (ii) Em certas situações, a característica do problema é originalmente distribuída. Nesse tipo de sistema, a maior dificuldade é criar diversas capacidades para resolver sub-problemas que além de serem complexos dependem de capacidades distintas. Outra motivação é a disposição física dos elementos envolvidos em um problema. Um

exemplo pode ser a tarefa de mineração distribuída aonde é inviável centralizar dados geograficamente dispersos. (iii) As crenças dos agentes envolvidos no problema também podem ser distribuídas, ou seja, é possível encapsular um grau de inteligência no comportamento do agente para resolver o problema, sem que seja necessário criar uma figura de agente centralizador. (iv) O resultado da resolução do problema deve ser distribuído, viabilizando a execução por diversos agentes. A centralização deve ser evitada por uma série de motivos (paralelismo, disponibilidade do coordenador, latência de comunicação). É preferível que os agentes atualizem seus planos de forma unilateral ou com um baixo nível de comunicação quando isto não for possível.

O compartilhamento de tarefas entre agentes compreende os seguintes passos:

- Decomposição de tarefas: Gerar o conjunto de tarefas que podem ser potencialmente repassadas para outros. Nesta etapa uma grande tarefa é dividida em sub-tarefas que podem ser executadas em agentes diferentes;

- Alocação das tarefas: Consiste em associar tarefas a agentes com as competências necessárias para a realização da tarefa;

- Realização da tarefa: Os agentes apropriados realizam cada sub-tarefa, o que pode incluir nova decomposição e alocação de tarefas recursivamente até encontrar uma atividade atômica;

- Agrupamento dos resultados: Quando um agente finaliza sua tarefa e passa o resultado para um agente apropriado que conhece as razões da decomposição e sabe como compor o resultado em um resultado global.

Segundo Gatti e Amigoni [GA04], na negociação cooperativa, cada agente tem uma visão parcial do problema e os resultados são compartilhados utilizando negociação, na tentativa de resolver conflitos, resultantes das visões parciais. O protocolo *Contract-net* proposto por Smith [SD83] pode ser utilizado como uma estratégia de coordenação eficiente entre agentes. A seção 4.4.1 apresenta uma definição deste protocolo, voltado para o compartilhamento de tarefas.

Existem diversas outras abordagens sofisticadas para coordenação de sistemas distribuídos cooperativos como por exemplo o PGP (*Planejamento Global Parcial / Partial Global Planning*) e o GPGP (*Planejamento Global Parcial Generalizado / Generalized Partial Global Planning*) apresentado por Decker e Lesser em [DL95] e técnicas de DCOP (*Satisfação Distribuída de Restrições / Distributed Constraint Optimization Problems*), tais como SBB (*Divisão e Conquista Síncrona / Synchronous Branch and Bound*) [HY97]

e *Adopt* [PJMY02] . Neste trabalho são descritas apenas as técnicas mais relevantes para a criação do método proposto.

4.4.1 Contract-net

Baseado em protocolos de mercado público, o *Contract-net* [SD83] destina-se a permitir a alocação eficiente de tarefas para agentes capazes de efetuar a resolução de um problema específico. Este protocolo pode ser considerado como um processo de contratação, iniciado por um agente que pretende solicitar algum serviço ou tarefa. O protocolo é composto de quatro etapas: anúncio da tarefa, encaminhamento das propostas, análise das propostas e emissão do contrato. Durfee [DUR99], cita o *Contract-net* como um protocolo eficiente para decomposição de tarefas. Além disso o processo de negociação cooperativa tem sido usado nas áreas de pesquisa de alocação de tarefas e mineração distribuída [SB02] [GA04] [BD03] [MLH03] [ZLP05].

A modelagem de agentes para resolução distribuída de problemas, gera um conjunto de agentes com diferentes capacidades de resolução. Portanto, para que um agente receba uma determinada tarefa e faça a decomposição em sub-tarefas, é necessário alocá-las para agentes apropriados. Conceitualmente é possível que um agente tenha uma tabela contendo as suas capacidades, sendo simples selecionar o agente apropriado para aquela tarefa. Porém, certas decisões ocorrem de maneira dinâmica. O *Contract-net* estabelece uma forma de comunicação entre agentes para a alocação de tarefas. Um agente, que no protocolo *Contract-net* é chamado de solicitante, decompõe um problema maior em uma série de sub-problemas e anuncia cada sub-problema na rede, juntamente com as especificações de quais agentes são elegíves para o sub-problema e instruções de como eles devem retornar uma proposta para o sub-problema. O receptor do anúncio decide se é elegível e então formula uma proposta. O solicitante coleta as propostas e repassa a especificação do sub-problema para o(s) agente(s) contratado(s) (aqueles que apresentaram as melhores propostas). O contratado recebe os detalhes do sub-problema, resolve-o (talvez quebrando-o em outros sub-problemas menores e contratando outros) e ao final retorna a solução para o solicitante.

Uma possível situação é a inexistência de agentes adequados para uma determinada tarefa no ambiente. Uma alternativa simples é reenviar o anúncio periodicamente, assumindo que os respondentes podem estar ocupados ou em breve estarão disponíveis no ambiente. O intervalo entre os reenvios pode ser um parâmetro perigoso: Se o reenvio acontece em intervalos muito longos, pode ser que alguns agentes fiquem parados por muito tempo sem necessidade. Por outro lado se o intervalo entre os reenvios for muito

rápido, a rede pode ficar congestionada devido ao excesso de troca de mensagens. Uma estratégia para dominar essa situação é inverter o uso do protocolo. Ao invés de anunciar tarefas e coletar propostas, o que implica que podem existir muitos respondentes para cada tarefa, o protocolo pode ser usado pelos respondentes para anunciar a disponibilidade, e os solicitantes podem responder para os anunciantes propondo suas tarefas. Ainda é possível mesclar as duas maneiras de utilização do protocolo dependendo de onde estiver o gargalo.

Uma alternativa ao reenvio, especialmente quando não existem agentes respondentes elegíveis na rede é o solicitante ajustar os anúncios a cada iteração, relaxando os requisitos até começar a receber as propostas. Um aspecto a ressaltar deste processo de relaxamento é que a especificação da elegibilidade refletirá nas preferências sobre diferentes classes de respondentes, ou mais especificamente, sobre a qualidade dos serviços que diferentes contratados disponibilizam. Outra dificuldade que um solicitante pode ter é quando nenhum agente tem a capacidade de resolver o problema especificado. O solicitante pode decompor o problema em outros menores, para que os agentes possam resolvê-lo.

4.4.2 Planejamento Distribuído

Segundo Durfee [DUR99], o planejamento distribuído pode ser entendido com a especialização de uma resolução distribuída de problemas, sendo que o problema a ser resolvido é desenvolver um plano. O planejamento é uma forma de coordenação entre agentes, com o objetivo de coordenar as ações dos agentes para alcançar um objetivo comum e evitar relações negativas entre os agentes. As relações negativas podem ser entendidas como obstruções para o objetivo final do agente, como por exemplo, a escassez de um determinado recurso ou a incompatibilidade de objetivos. Ferber [FER03] apresenta uma classificação da atividade de criação e planejamento de tarefas entre agentes:

- Planejamento Centralizado para Múltiplos Agentes

  Neste contexto, apenas um agente é designado para criar os planos, que contém as ações para os demais agentes. O agente planejador deve inicialmente mapear um plano geral, identificar pontos onde é possível paralelizar as atividades, sendo possível a criação de sub-planos. Esses sub-planos podem ser alocados por outros agentes que serão responsáveis pelo gerenciamento do sub-plano. Os demais agentes serão meros executores de atividades.

- Coordenação Centralizada para Planos Parciais

A figura do agente centralizador existe apenas para coordenar as ações dos planos criados por outros agentes. Esse agente tem a tarefa de identificar regiões de conflito e dependências entre os planos. As medidas podem ser a criação de semáforos ou ordenação de ações.

- Coordenação Distribuída para Planos Parciais

  Nesta situação não existe uma figura que elabora e coordena os planos. Os agentes são responsáveis por identificar situações de conflito de recursos ou dependências entre suas ações através de trocas de mensagens.

4.4.3 Eco-resolução

A Eco-resolução é uma abordagem distribuída para a resolução de conflitos e busca em espaços de estados para a solução de um problema. Um eco-problema é composto por uma população de agentes chamados eco-agentes, que têm por objetivo alcançar um estado de estabilização, chamado de solução do problema [FER03]. Esta técnica parte do pressuposto que um agente simples não faz planejamento e apenas reage no seu meio. Transformando a metáfora da natureza para um problema real, o princípio é modelar o problema em pequenas partes, modelar as funções que são disparadas pelos estímulos externos e deixar que as partes alcancem a estabilização.

Cada eco-agente tenta atingir seu objetivo individualmente. As percepções locais são transformadas em ações que tendem a alterar o ambiente e permitir que outros agentes tenham a chance de alterar seus estados.

Segundo Ferber, um eco-agente pode ter um dentre os quatro estados mentais: *satisfeito, em busca de satisfação, em fuga* ou *em busca de fuga.* A necessidade de satisfação, fará o eco-agente agir em seu ambiente. O ambiente recebe diversas interações dos agentes, como num jogo de sobrevivência. Abaixo é apresentado um cenário comum entre eco-agentes:

- o eco-agente pode perceber que outros eco-agentes são obstáculos para sua meta pessoal;
- o agente intruso tem a obrigação de escapar;
- o agente que sofreu o ataque tenta fugir, sendo que seu estado é alterado;
- o agente que está fugindo pode atacar um outro agente durante a fuga;
- esta corrente de ataque é quebrada assim que houver um eco-agente que possa atingir o estado de satisfação sem atacar um outro eco-agente.

O desejo de satisfação do eco-agente pode ser descrito através de um simples algoritmo. O Algoritmo 1 é chamado periodicamente por todos os agentes. Esta periodicidade vai depender do problema que está sendo modelado. Isto permite definir dependências e sucessões de satisfações de todos os eco-agentes.

Algoritmo 1 Busca de Satisfação de um Eco-agente [FER03]

Pré-requisito: x: um agente; y: obstrutores;

```
1: função TentarSatisfazer(x, y)
2: se objetivo( x ) satisfeito e x não está satisfeito então
3:    para todos os y obstrutores faça
4:       TentarEscapar( y , x)
5:       Satisfazer( x )
6:    fim para
7: fim se
```

A função *Satisfazer(agente)* é dependente do domínio do problema e deve checar se o estado atual atende suas expectativas. Já o Algoritmo 2 ($TentarEscapar(x, y)$), representa o comportamento de fuga, o qual abrirá espaço para que os outros agentes percorram seus espaços de busca.

Algoritmo 2 Tentativa de Fuga de um Eco-agente [FER03]

```
1: função TentarEscapar(x, y)
2: se x não estiver satisfeito então
3:    obstrutor torna-se insatisfeito
4: fim se
5: p = EncontrarLocalParaFuga(x, y)
6: se p não foi encontrado então
7:    solução não encontrada
8:    parar de buscar solução
9:    sair
10: else
11:    para todas as possibilidades de fuga z do obstruidor x faça
12:       TentarEscapar(z, x)
13:    fim para
14: fim se
15: Escapar(x, p)
```

A função *Escapar(x, p)* também é dependente do problema que está sendo modelado e representa a tentativa de mudar seu estado, permitindo que outros agentes atinjam a satisfação.

A eco-resolução é um mecanismo eficiente de resolução de conflitos e de busca em espaços de estados e atenua problemas *NP-hard*. No entanto modelar um eco-problema (eco-agentes, estado inicial, objetivos e critérios de parada) não é uma atividade trivial pois uma modelagem incorreta pode gerar situações de impasse (*deadlock*).

4.5 Considerações Finais

Neste capítulo foram apresentados os principais mecanismos de decomposição de tarefas, colaboração e resolução de problemas utilizados em sistemas IAD. Uma vez que a heurística para a combinação de classificadores deve permitir a fragmentação do problema em pequenas partes, a distribuição de tarefas requer um mecanismo formal de comunicação. Os conceitos abordados, como por exemplo, *Contract-net*, RDP e eco-agentes, foram elementos fundamentais para a criação do método.

Capítulo 5

SDICCS - Um Sistema Distribuído para Combinação de Classificadores Simbólicos

Neste capítulo, é descrito o sistema de combinação de classificadores simbólicos, o SDICCS, responsável por:

1. Construir um classificador simbólico usando diversos classificadores gerados a partir de diferentes amostras de dados. O classificador deve extrair as "melhores" regras das hipóteses distribuídas, desde que não existam regras conflitantes entre os agentes cooperativos;

2. Usar um *ensemble* de classificadores nas situações onde o classificador combinado não tenha cobertura suficiente. Um *ensemble* é um conjunto de classificadores cujas decisões individuais são combinadas de alguma forma para classificar um exemplo ainda não classificado [FS97]. Neste caso, através de voto simples;

3. Fornecer ao usuário um módulo de classificação de exemplos não conhecidos, que explicará qual ou quais regras foram disparadas.

O método compreende as fases de: (i) preparação dos dados, a qual determina como ocorre a divisão dos exemplos de treinamento e teste, (ii) aprendizagem dos agentes classificadores, (iii) combinação das hipóteses distribuídas e finalmente (iv) a classificação de novos exemplos. Em (iii), o foco é a obtenção de um conhecimento unificado, sendo que o resultado final é de um conjunto de regras ordenadas capaz de descrever padrões não divergentes entre os agentes. Em (iv), além do objetivo de explicar porque um exemplo não conhecido foi classificado, é necessário garantir a completeza do classificador: uma vez que o modelo unificado deixa de cobrir certas regiões do espaço de tuplas, um *ensemble* é usado.

Antes de descrever as fases (i) a (iv), são apresentadas duas seções: A primeira apresenta um conjunto de definições que é usado no restante do documento e a segunda

descreve uma base de exemplos, que é usada para fins de ilustração da abordagem. Após descrever as fases citadas anteriormente, a seção de implementação é apresentada, a qual detalha a arquitetura do sistema e o mecanismo de cooperação entre os agentes, uma vez que as seções anteriores mencionam as trocas de mensagens entre os agentes em alto nível. Ao final, são apresentados trabalhos com características comuns e considerações finais.

5.1 Definições

As definições a seguir são utilizadas no decorrer do texto para facilitar a compreensão da abordagem.

Uma base de exemplos S é dividida em duas partes, resultando nas bases de exemplo T e V, treinamento e teste respectivamente. A base T é dividida em n partes, resultando nas bases distribuídas $t_1, t_2, ..., t_n$, onde n é a quantidade de fragmentos que será usada. Cada t_{ji} é um conjunto de j instâncias $\{(x_1, y_1), (x_2, y_2), ..., (x_j, y_j)\}$, para alguma função desconhecida $y = f(x)$. Cada instância x_j é tipicamente um vetor na forma $(xj_1, xj_2, ...xj_w)$, sendo que seus componentes são chamados de atributos ou características e w é a quantidade de atributos. Para classificação, os valores de y são representados por um conjunto discreto contendo Ncl classes; $y \in \{C_1, C_2, ..., C_{Ncl}\}$.

Um conjunto de agentes $A = \{a_1, a_2, ..., a_n\}$ é associado a T, formando uma relação $A \otimes T = \{(a_1, t_1), (a_2, t_2), ..., (a_n, t_n)\}$, onde n é a quantidade de fragmentos da base de exemplos T. Dados os conjuntos de exemplos t, cada agente executa um algoritmo de aprendizagem, que induz um classificador chamado *hip*, sendo uma hipótese da função objetivo e desconhecida $f(x)$. Dados novos valores x, o classificador *hip* prediz o valor correspondente de y. Ao final temos um conjunto Hip de hipóteses $\{hip_1, hip_2, ..., hip_n\}$, cada uma pertencente ao respectivo agente $(a_1, a_2, ..., a_n)$.

Neste trabalho cada hipótese *hip* é um conjunto de regras ordenadas, ex. $hip_i = \{R_1, R_2, ..., R_j\}$, onde ji é a quantidade de regras do classificador. Uma regra R pode ser considerada um conjunto de testes no formato $< x_i \; op \; valor >$, onde x_i é um atributo ou característica, *op* é um operador pertencente ao conjunto $\{=, \neq, \geq, \leq, >, <\}$ e valor é um dado contínuo ou discreto. Portanto, uma regra R assume simbolicamente o formato B $\rightarrow$ H, onde B é a conjunção de testes atributo-valor (*body*) e H (*head*) é a classe associada C_i, sendo que $C_i \in \{C_1, C_2, ..., C_{Ncl}\}$.

Os processos de combinação e classificação são realizados por dois tipos de agentes, sendo que cada um tem um papel bem definido.

- Agente de aprendizagem ou a_i: É responsável por realizar a etapa de aprendizagem,

enviar o conjunto de regras, avaliar conjuntos de regras e classificar novos exemplos quando solicitado.

- Agente coordenador ou a_{coord}: É responsável por solicitar os conjuntos de regras de cada agente de aprendizagem, solicitar a avaliação das regras para cada agente, montar a fila de avaliação de regras e usar um algoritmo de busca em espaços de estado para encontrar a melhor regra. Ao final deve enviar a hipótese combinada Hip' para os agentes de aprendizagem.

O agente a_{coord} cria uma árvore de combinação de regras durante a etapa de combinação que é detalhado adiante. Esta árvore é composta por vértices e arestas. Por convenção, um vértice é chamado V_l, sendo que l é o nível o qual o vértice pertence e V_0 é o vértice raiz.

5.2 Descrição do Conjunto de Exemplos Ilustrativo

Para ilustrar o funcionamento do combinador de classificadores e o mecanismo de classificação nas próximas seções, é utilizada uma base de exemplos hipotética contendo dois atributos (x_1 e x_2) e uma classes alvo composta de dois possíveis valores (a e b). Isso permite que o conjunto de exemplos e as regras sejam representados em um plano cartesiano. A função objetivo f é definida na fórmula 5.1.

$$f(x_1, x_2) = \begin{cases} a, & \text{se} \quad 10 \leq x \leq 50, 30 < y \leq 50 \\ a, & \text{se} \quad 50 < x \leq 90, 10 \leq y \leq 30 \\ b, & \text{se} \quad 10 \leq x \leq 50, 10 \leq y \leq 30 \\ b, & \text{se} \quad 50 < x \leq 90, 30 < y \leq 50 \end{cases} \tag{5.1}$$

A Figura 5.1 apresenta a função objetivo f no plano cartesiano.

Dada a função objetivo f, vários exemplos ilustrativos podem ser gerados nas próximas seções. O objetivo é criar um ambiente que reproduza a incerteza e visão parcial típica de classificadores distribuídos.

5.3 Preparação dos Dados

Com o objetivo de simular um ambiente distribuído, uma base de exemplos S é utilizada para a montagem das bases distribuídas de treinamento. Essa base de exemplos é dividida da seguinte forma: 20% dos exemplos é reservado para a avaliação final (V) do classificador *SDICCS*. Os 80% restantes (T) são divididos em n partes, onde n é a

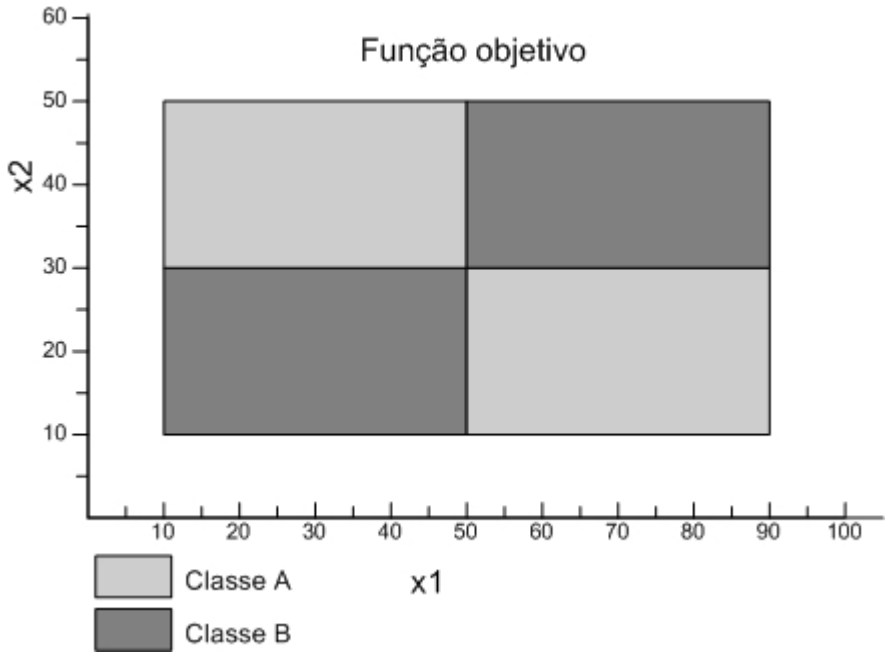

Figura 5.1: Função Objetivo f

quantidade de agentes de aprendizagem disponíveis no ambiente. A Figura 5.2 ilustra o processo de preparação da base de exemplos. Por tratar-se de um ambiente de simulação, a quantidade de exemplos de testes é relativamente menor do que a quantidade de exemplos de treinamento, o que é o inverso de um ambiente real.

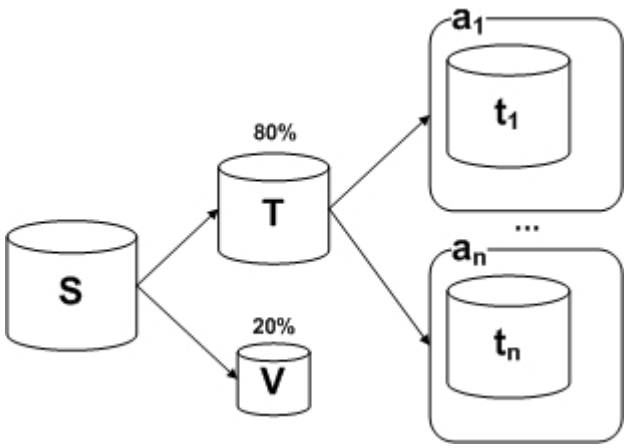

Figura 5.2: Preparação dos Dados

A divisão da base de exemplos naturalmente reduz a capacidade de predição do classificador, uma vez que a quantidade de exemplos de treinamento é menor e muitas vezes insuficiente para criar um conjunto de hipóteses que cubra todos os casos. Para simular uma situação na qual os agentes têm visões incompletas sobre um determinado problema, são gerados como exemplo 3 conjuntos de dados utilizados por 3 agentes. Os conjuntos t_i obedecem às funções $fAg_i(x_1, x_2)$, onde Ag_i é um agente e as funções podem

ser interpretadas como visões parciais da função objetivo $f(x_1, x_2)$ (fórmulas 5.2, 5.3 e 5.4).

$$fAg_1(x_1, x_2) = \begin{cases} a, & \text{se} \quad 10 \leq x \leq 50, 30 \leq y \leq 50 \\ a, & \text{se} \quad 50 < x \leq 90, 10 < y \leq 20 \\ b, & \text{se} \quad 10 \leq x \leq 50, 10 < y \leq 20 \\ b, & \text{se} \quad 50 < x \leq 90, 30 \leq y \leq 50 \end{cases} \tag{5.2}$$

$$fAg_2(x_1, x_2) = \begin{cases} a, & \text{se} \quad 10 \leq x \leq 50, 40 \leq y \leq 50 \\ a, & \text{se} \quad 50 < x \leq 90, 10 \leq y \leq 30 \\ b, & \text{se} \quad 10 \leq x \leq 50, 10 < y \leq 30 \\ b, & \text{se} \quad 50 < x \leq 90, 40 \leq y \leq 50 \end{cases} \tag{5.3}$$

$$fAg_3(x_1, x_2) = \begin{cases} a, & \text{se} \quad 10 \leq x \leq 40, 30 \leq y \leq 50 \\ a, & \text{se} \quad 50 < x \leq 90, 10 < y \leq 30 \\ b, & \text{se} \quad 10 \leq x \leq 50, 10 < y \leq 30 \\ b, & \text{se} \quad 60 < x \leq 90, 30 \leq y \leq 50 \end{cases} \tag{5.4}$$

Estes conjuntos foram denominados t_1, t_2 e t_3. O último conjunto (V) é usado para teste e apresenta distribuição de acordo com a função objetivo apresentada na seção anterior. Cada conjunto tem um total de 100 exemplos gerados aleatoriamente.

A Figura 5.3 apresenta como os agentes interpretam a função objetivo $f(x_1, x_2)$. Os três agentes não conseguem representar completamente o problema, sendo necessário combiná-los ou usá-los em conjunto para a realização da predição. Além dos classificadores terem visões parciais é provável a ocorrência de divergências em uma suposta união de regras sem critérios, uma vez que:

- regras ordenadas não podem ser usadas isoladamente. Elas possuem um *else* implícito como elemento de ligação. Sendo assim, uma regra pode ser complemento de outra, causando uma dependência estatística. Usá-las isoladamente aumentará a taxa de erros;

- a otimização de certos algoritmos generaliza o conhecimento através de regras *default*, baseado no conjunto de treinamento disponível. De acordo Bacardit e Goldberg [BGB07], regras *default* são criadas na maioria das vezes baseadas na distribuição das classes (maioria ou minoria). Em certas bases de dados esta alternativa produz altas taxas de erros.

Uma vez que a base foi particionada, o próximo passo é a realização da aprendizagem pelos agentes.

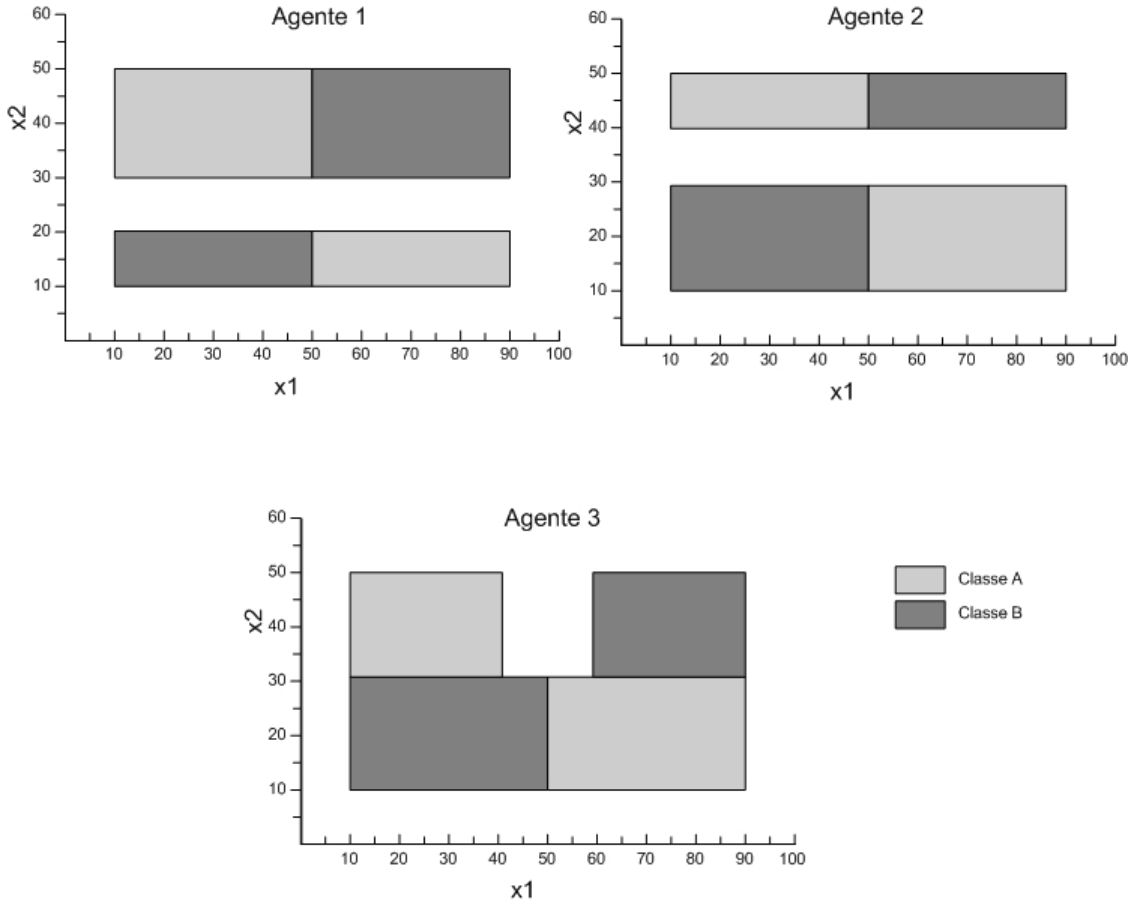

Figura 5.3: Função Verdadeira $fAg_i(x1, x2)$ para os 3 Agentes de Aprendizagem

5.4 Etapa de Aprendizagem Local

Segundo Prati e Flach [PF05], a maioria dos classificadores pertencem a uma de duas famílias chamadas *remover-para-conquistar* e *dividir-para-conquistar*. As duas famílias compartilham diversas características, sendo que a mais representativa é que o espaço de exemplos contém grandes regiões que pertencem à mesma classe.

Nos algoritmos que pertencem à família *remover-para-conquistar* [FUR99], o processo de busca é geralmente implementado através de um algoritmo "guloso", sendo que em cada iteração ele busca a melhor regra (de acordo com algum critério) e remove os exemplos cobertos (a parte remover). Este processo se repete usando os exemplos que sobraram até que todos os exemplos tenham sido cobertos ou algum critério de parada tenha sido disparado (a parte conquistar). Alguns desses algoritmos induzem regras que obedecem uma seqüência, formando um conjunto ordenado ou uma lista de decisão, como é muitas vezes chamado. Este conjunto de regras deve ser usado na seqüência especificada. A classificação de novos exemplos é dada a partir da primeira regra disparada. Caso nenhuma regra atenda a especificação do corpo da regra, então a regra *default* é

disparada. A regra *default* é um tipo especial de regra que não possui condições, apenas a classe alvo e muitos algoritmos a definem baseados na distribuição das classes no conjunto de treinamento.

Já no caso de algoritmos que pertencem a família *dividir-para-conquistar* [QUI93], o classificador é obtido usando uma estratégia *top-down*, com refinamentos consecutivos. O resultado é geralmente uma árvore de decisão, que divide o espaço de exemplos em hiper-retângulos, sem sobreposição. A representação da árvore em forma de regras nada mais é do que percorrer os nós da árvore até as folhas. Sendo assim, as regras geradas são mutuamente exclusivas, também chamadas de regras não ordenadas.

Para a realização dos experimentos, o algoritmo RIPPER [COH95] apresentado por Cohen, foi selecionado por tratar-se de um algoritmo que pertence a família *remover-para-conquistar* ou *set-covering algorithm*, foco deste trabalho. Além disso, este algoritmo tem desempenho comparável aos algoritmos de árvore de decisão C4.5 e C4.5rules [QUI93], tendo melhor desempenho do que C4.5rules em bases ruidosas [COH95].

5.4.1 Execução

Cada agente a_i induz um classificador usando o algoritmo RIPPER [COH95], formando uma hipótese $h_i = \{R_1, R_2, ..., R_j\}$, composta de ji regras ordenadas. A Figura 5.4 ilustra este processo de aprendizagem local.

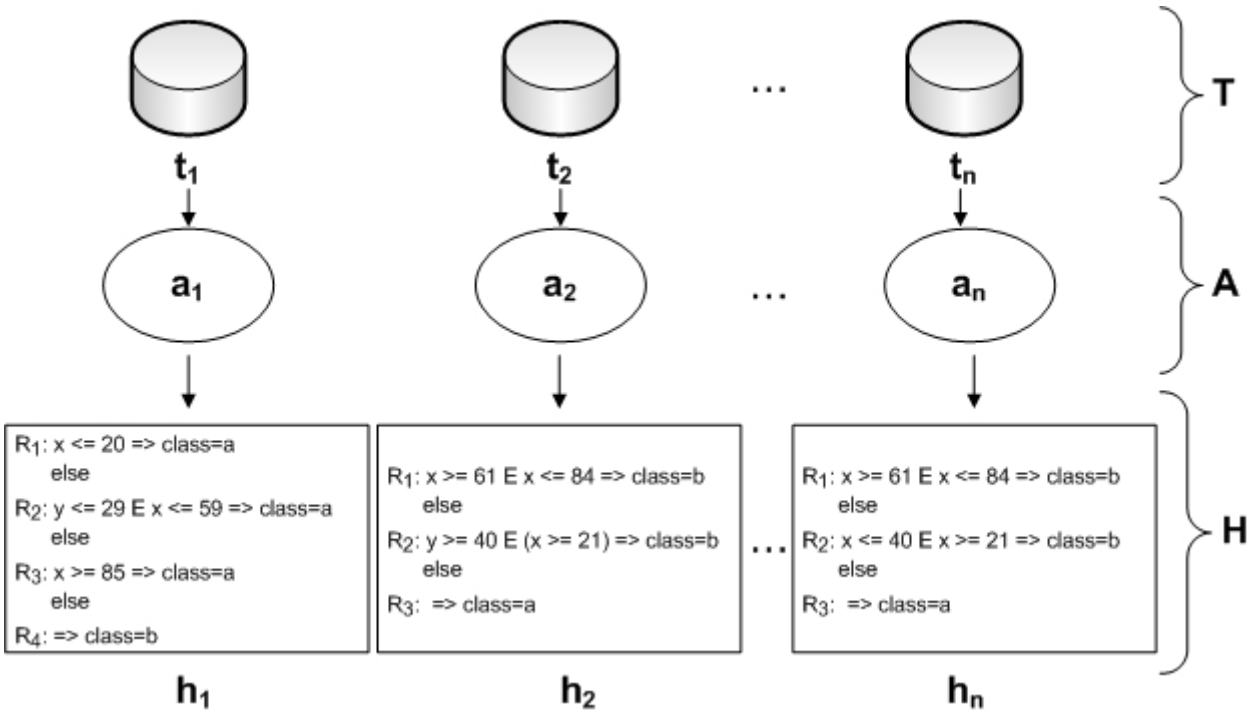

Figura 5.4: Processo de Aprendizagem Local

Geralmente os algoritmos de indução de regras criam uma regra *default*, com o objetivo de classificar exemplos que não foram cobertos devido ao processo de otimização de regras. Esse tipo de regra que apresenta apenas a classe H e nenhum antecedente funciona como um *else* implícito dentro do conjunto de regras. Devido a dificuldade de mensuração da qualidade e capacidade de explicação nula, esse tipo de regra é eliminado do conjunto h_i. Para suprir a eliminação da regra *default*, além do mecanismo de combinação de hipóteses, é utilizado o módulo de *ensemble* que será detalhado posteriormente.

Regras não ordenadas podem ser avaliadas isoladamente. Existem várias propostas de mecanismos de avaliação destas regras, sendo possível estabelecer *thresholds* e selecionar aquelas com a característica desejada, como por exemplo: grau de suporte, precisão e novidade. Vários trabalhos usam técnicas de seleção e ordenação de regras usando medidas para seleção e combinação de regras [SHA00] [GL00] [BER02] [HCBK99] [CL04] [PASE06] [SEN06]. A abordagem proposta foi desenvolvida para manipular regras ordenadas, as quais tornam a operação de combinação mais complexa. Existem poucas propostas para combinação de regras ordenadas na literatura. O fato de cada partição ter um *bias* e o mecanismo de geração de regras ser muitas vezes instável pode agravar a combinação. Além disso esse tipo de regra não pode ser avaliada isoladamente, pois uma regra pode ser o complemento de outra. Portanto, encontrar a melhor combinação é um processo exaustivo e demanda poder de processamento dos agentes. Segundo Furnkranz e Flach [FF03], enquanto medimos a precisão ao classificar exemplos desconhecidos para regras individuais, a avaliação de uma regra incompleta ou ordenada deve capturar o potencial para ser refinada.

A Figura 5.5 apresenta as regras geradas pelos três classificadores usando a base de ilustração com o algoritmo RIPPER. A área pontilhada representa a função objetivo $f(x_1, x_2)$. Nesta figura é perceptível a existência de áreas de divergências entre os classificadores, mesmo que o conjunto de exemplos não apresente nenhum ruído. Por exemplo: Os agentes 1 e 2 não conseguem descrever qual é o padrão correto para a classe A. O contrário acontece para o agente 3 que não consegue descrever a classe B, apesar de ser o classificador que melhor representa a função objetivo. Parte do conhecimento é perdido devido a utilização da regra *default*.

Uma vez que os agentes finalizaram o processo de aprendizagem é possível ir para a próxima etapa, responsável pela combinação dos classificadores.

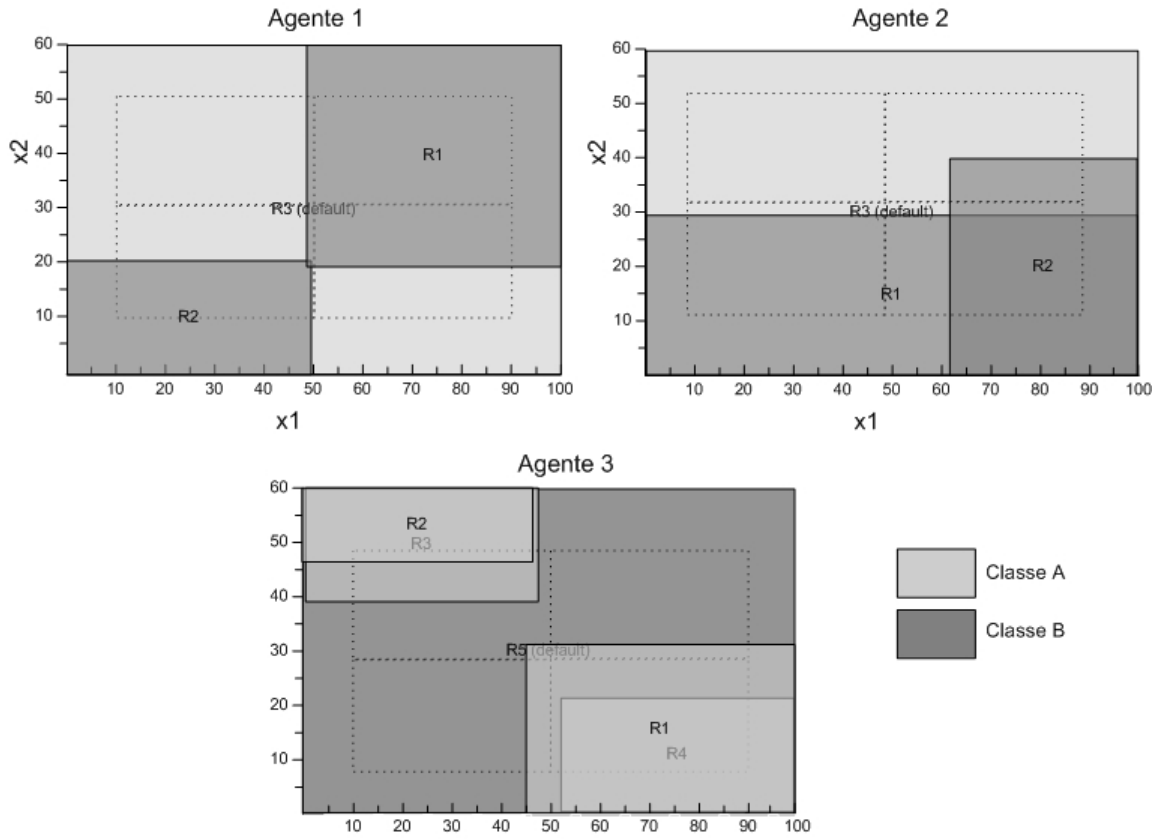

Figura 5.5: Regras Geradas pelos Agentes a_1, a_2 e a_3

5.5 Combinação dos Classificadores

Dado um conjunto de hipóteses $Hip = \{hip_1, hip_2, ..., hip_n\}$ e um conjunto de treinamento com N exemplos, $T = \{(x_i, y_i), i = 1, ..., N\}$, pertencentes a um agente do conjunto $A = \{a_1, a_2, ..., a_n\}$, o processo de combinação de classificadores cria uma nova hipótese Hip', contendo l regras $R \in Hip$, através da cooperação entre os agentes, onde l é a quantidade de regras selecionadas pelo processo.

A etapa de combinação de classificadores pode ser dividida em:

- compartilhamento das hipóteses distribuídas (Hip);
- criação da hipótese combinada (Hip');
- compartilhamento da hipótese combinada (Hip').

5.5.1 Compartilhamento das Hipóteses Distribuídas (Hip)

A etapa de compartilhamento das hipóteses distribuídas ocorre a partir do momento em que o agente coordenador solicita aos agentes de aprendizagem para que enviem suas hipóteses ao grupo. O objetivo do compartilhamento é permitir que o agente

coordenador monte seu plano de execuções e os agentes de aprendizagem usem o conhecimento comum formando um *ensemble* nas situações onde o classificador unificado não cobre algum exemplo. O Algoritmo 3 apresenta a etapa de compartilhamento.

Algoritmo 3 Compartilhamento das Hipóteses Distribuídas entre os Agentes

Pré-requisito: $A = \{a_1, ..., a_n\}$: conjunto de agentes; $H = \{h_1, ..., h_n\}$:conjunto de hipóteses

função compartilharHipoteses(A,H)
//
// Comportamento para envio da hipótese para cada agente
para cada agente de aprendizagem $a_i \in A$ **faça**
  enviar h_i **para** $A_{vizinhos} = \{a \in A | a \neq a_i\}$
fim para
//
// Comportamento para recebimento e avaliação das hipóteses
para cada agente de aprendizagem $a_i \in A$ **faça**
  receber h_i **dos agentes vizinhos**
  avaliar cada regra em h_i **usando** t_i **e atualizar** TP **e** FP
  criar uma lista de conflito ($Conflitos$**) para cada regra em** h_i
  //
  // Envia a lista de conflitos e cobertura e suporte de cada regra
  enviar $Conflitos$**,**$h_{i_{atualizado}}$ **para agente coordenador (**a_{coord}**)**
fim para
//
// Coordenador: Receber as regras e criar a fila de combinação
$coordenador \leftarrow \{agente \in A | agente = coordenador\}$
Criar uma fila (Q**) de regras ordenada por** $\frac{TP}{TP+FP}$
Armazenar a fila Q **para combinação posterior**

- Uma vez que os agentes de aprendizagem (a_i) entram no ambiente, o agente coordenador (a_{coord}) envia uma mensagem para que o processo de compartilhamento inicie;
- Cada agente a_i envia seu conjunto de regras para os demais agentes;
- Uma vez recebidas as regras, os agentes de aprendizagem realizam a avaliação das regras individualmente em suas bases de treinamento. O agente identifica qual é a quantidade de exemplos cobertos correta (TP) e incorretamente (FP) para cada regra tomada isoladamente. Esse processo se repete para todas as regras de todos os agentes. Estes valores são usados posteriormente pelo agente a_{coord} para determinar a precisão de cada regra e conseqüentemente a ordem da fila de avaliação Q.
- Cada agente de aprendizagem (a_i) devolve as regras avaliadas juntamente com os valores respectivos de TP e FP para o agente a_{coord};

- O agente de aprendizagem (a_i) também envia para o agente coordenador (a_{coord}) uma lista de regras conflitantes para cada regra. Entende-se por conflito entre duas regras a cobertura simultânea de exemplos, sendo que as regras prevêem classes diferentes. O agente a_i chega a esta conclusão validando as coberturas das regras em sua base de treinamento t_i;

- O agente coordenador recebe a relação de regras avaliadas, soma todos os valores de TP e FP por regra e cria uma fila Q, que é usada posteriormente para o processo de combinação de regras. A fila é ordenada da maior para a menor medida de precisão, que é dada pela fórmula $\frac{TP}{TP+FP}$. Esta medida permite avaliar o nível individual da precisão da regra, o que favorece a avaliação antecipada de regras mais precisas, dada a ordem da fila.

5.5.2 Criação da Hipótese Hip'

Nesta etapa, uma fila contendo a lista de regras $Q = \{r_1, ..., r_n\}$, ordenada pela medida de precisão individual ($\frac{TP}{TP+FP}$), é usada como entrada para a criação de uma árvore, chamado *árvore de combinação*, que é composta por:

- Vértices: Pode ser um vértice raiz, intermediário ou folha. Por convenção é chamado de V_l, sendo que l representa o nível da árvore. O vértice raiz possui um conjunto de regras vazio ($\emptyset$), denominado V_0. Um vértice intermediário ou folha é composto por um conjunto de regras, resultante da etapa de busca e assume a notação V_l.

- Arestas: As arestas unem dois vértices e armazenam o ganho obtido na passagem de um vértice para outro. O ganho é a subtração da medida de suporte do vértice filho pela medida de suporte do vértice pai, como apresentado na fórmula 5.5.

$$ganho_{V_l} = suporte(V_l) - suporte(V_{l-1}) \tag{5.5}$$

A medida de suporte é obtida pela fórmula 5.6.

$$suporte(V) = \frac{TP}{N} \tag{5.6}$$

onde TP é a quantidade de exemplos cobertos corretamente e N é a quantidade total de exemplos, considerando as amostras de todas as bases distribuídas.

Para expandir a árvore de combinação, foi criado um algoritmo inspirado na solução de Dijkstra [DIJ59], que resolve o problema do menor caminho em grafos, dado um vértice inicial e final. Foi necessária uma adaptação no algoritmo original, uma vez que o vértice final não é conhecido. O critério de parada neste caso é a inexistência de regras a avaliar na fila Q.

Com este algoritmo é possível avaliar cada combinação de regras e o ganho obtido ao adicionar, remover ou trocar uma regra de posição. Além disso é característica do algoritmo permitir *backtracking*, uma vez que é possível retomar a expansão de arestas anteriormente descartadas devido ao baixo ganho em um certo momento, porém convenientes em outro.

Algoritmo 4 Algoritmo de Combinação de Hipóteses

Pré-requisito: $Q = r_1, r_2, ..., r_n$: fila de regras

```
1:  função combinarHipoteses(Q)
2:  O ← ∅ {inicializar a lista aberta}
3:  D ← ∅ {inicializar a lista fechada}
4:  raiz ← vertice(∅) {iniciar com um vértice vazio}
5:  raiz.ganho ← 0
6:  adicionarVertice(O,raiz)
7:  // continuar enquanto houverem vértices
8:  // e não tenha atingido o nível máximo da árvore
9:  enquanto O ≠ ∅ e nivelArvore(maximaPrecisao(O)) ≠ Q.tamanho faça
10:   verticeVisitado ← maximaPrecisao(O)
11:   remove verticeVisitado de O
12:   D.adicionar(verticeVisitado)
13:   expandir(verticeVisitado)
14:   para cada aresta e pertencente ao verticeVisitado faça
15:     // realizar o relaxamento das arestas
16:     se e.filho.ganho < verticeVisitado.ganho + e.ganho então
17:       e.filho.ganho ← verticeVisitado.ganho + e.ganho
18:       e.pai = verticeVisitado
19:       adicionarVertice(O,e.filho)
20:     fim se
21:   fim para
22: fim enquanto
23: maiorGanho ← maximaPrecisao(O)
24: retornar  maiorGanho.classificador
```

O Algoritmo 4 mostra como a hipótese é construída. A partir do recebimento dos parâmetros de entrada o algoritmo segue os seguintes passos:

1. A árvore inicia com um vértice V_0, que contém um conjunto de hipóteses vazio ($Hip' = \emptyset$);

2. As listas aberta e fechada são inicializadas;

Algoritmo 5 Algoritmo de Expansão de Arestas

Pré-requisito: V: vértice contendo a hipótese $Q = r_1, r_2, ..., r_n$: fila de regras

1: **função expandir(V,Q)**
2: **se V.nivel = tamanho(Q) então**
3: **retornar**
4: **fim se**
5:
6: **criar um vértice com a nova regra (final do classificador)**
7: **se** $precisao(V.classificador) < 0.50$ **então**
8: **descartar vértice**
9: **fim se**
10: **criar um vértice descartando a nova regra**
11: **se** $precisao(V.classificador) < 0.50$ **então**
12: **descartar vértice**
13: **fim se**
14: **para cada regra em conflito com a nova regra faça**
15: **criar um vértice V com a nova regra antes da regra em conflito**
16: **se** $precisao(V.classificador) < 0.50$ **então**
17: **descartar vértice**
18: **fim se**
19: **criar um vértice V com a nova regra sobrepondo a regra em conflito**
20: **se** $precisao(V.classificador) < 0.50$ **então**
21: **descartar vértice**
22: **fim se**
23: **fim para**
24: **retornar**

3. O laço principal executa enquanto houverem vértices na lista aberta e existam elementos a processar na fila Q;

4. É recuperada a regra com maior medida de precisão da fila Q;

5. É removida a regra em avaliação da lista aberta;

6. As arestas são expandidas para a regra em avaliação (Ver algoritmo 5);

7. Uma aresta é criada apontando para um vértice (V_1) que contém a primeira regra da fila Q. Relembrando, a fila Q está ordenada pela medida de precisão, da maior para a menor. A primeira regra desta fila é a que tem menor independência estatística;

8. Cada agente a_i realiza a avaliação da hipótese contida no vértice, usando sua base de treinamento e devolve os valores TP e FP do conjunto para o agente coordenador;

9. O agente coordenador, após receber as avaliações dos n agentes, soma os valores TP e FP obtidos dos agentes a_i e calcula o suporte do conjunto (ver fórmula 5.6);

10. A diferença entre o suporte do vértice anterior e do vértice atual é usada para atribuir o peso da aresta. No caso do primeiro vértice, é computado apenas o valor do suporte (ver fórmula 5.5);

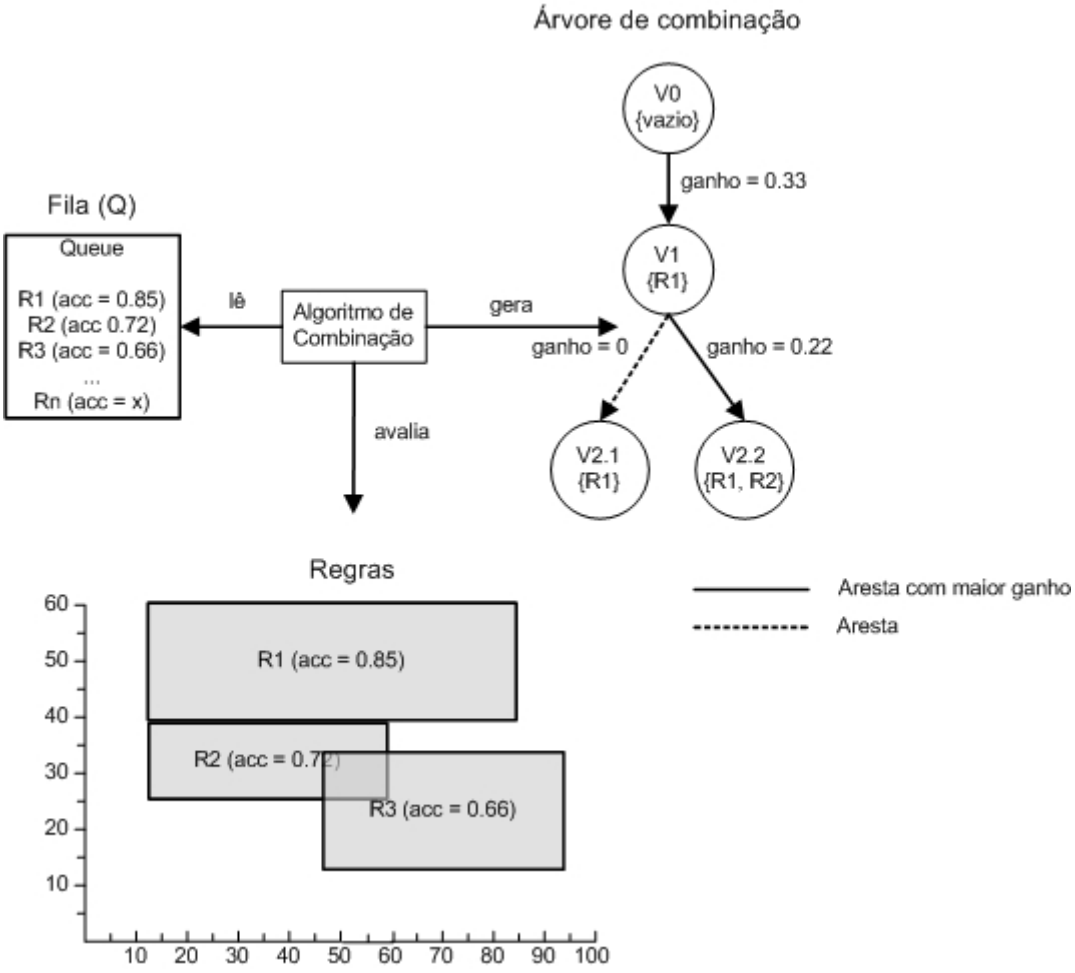

Figura 5.6: Processo de Combinação dos Classificadores

11. Na expansão de vértices candidatos, o agente coordenador (a_{coord}) lê a próxima regra da fila Q e cria duas possibilidades: Uma onde consta o conjunto de regras anterior acrescido da regra obtida da fila e outra onde consta apenas o conjunto anterior, desprezando a regra da fila. Isto permite descartar regras com baixa precisão ou muito específicas para a partição onde foi criada. A Figura 5.6 ilustra uma expansão de vértices. A partir do vértice V_1 que contém a regra R_1, o algoritmo expande os vértices $V_{2.1}$ e $V_{2.2}$, sendo que o vértice $V_{2.1}$ contém apenas a regra $R1$ e o ganho para passar de um vértice para outro é nulo;

12. No caso do vértice $V_{2.2}$ há um ganho de 0.22. Relembrando: O ganho de cada aresta é obtido a partir da subtração do ganho do vértice anterior pelo ganho do vértice corrente (ver fórmula 5.5). Por exemplo: Se o conjunto apresentasse mau

desempenho ao adicionar a regra $R2$, seu ganho seria negativo, o que comprometeria as próximas expansões, uma vez que o algoritmo só expande vértices com melhor desempenho;

13. Durante a expansão, caso a cobertura da regra seja menor do que a cobertura original, significa que parte dos exemplos que ela cobriria está sendo coberta por outra. Se faz necessário investigar se ela terá melhor desempenho que a conflitante ou se funcionará melhor se a outra regra for complemento desta.

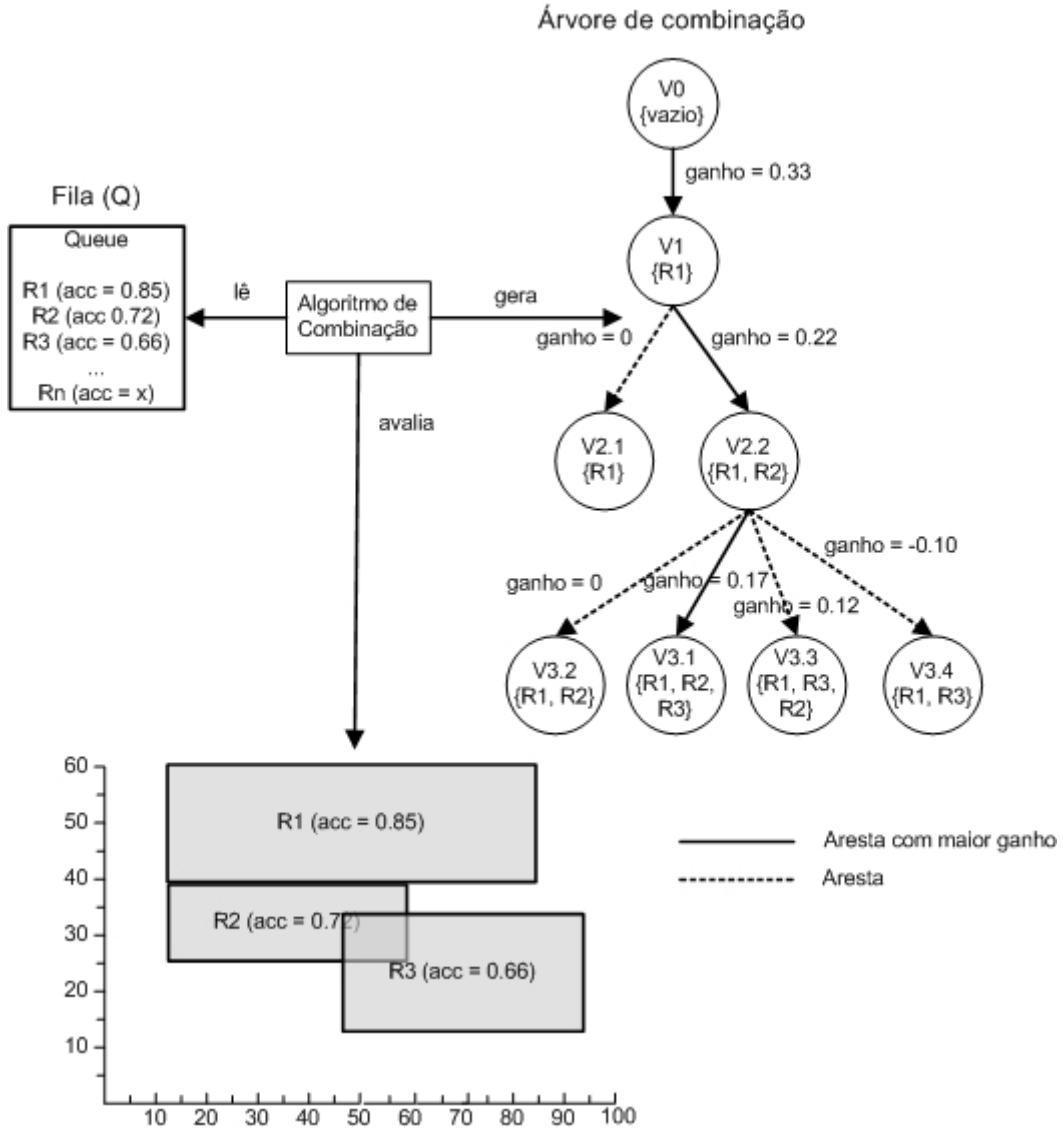

Figura 5.7: Combinação de Regras Conflitantes

Portanto, para cada regra conflitante, o algoritmo criará dois novos vértices: um contendo a regra conflitante após a regra da fila e outro substituindo a regra conflitante pela regra da fila. É o caso da regra R3 da Figura 5.6: É possível visualizar uma provável intersecção ou conflito entre a regra R3 e R2. Na próxima com-

binação, possivelmente R3 não cobrirá todos os exemplos "potenciais" uma vez que os exemplos da base de treinamento cobertos pela regra R2 não estão mais visíveis (estratégia de classificação *best first*). Por esta razão é necessário: (i) combinar as regras R3 e R2: pode ser que as duas tenham melhor desempenho combinadas ou (ii) substituir a regra R3 pela regra R2: pode ser que na combinação deste vértice a regra R2 tenha melhor desempenho que a R3. A Figura 5.7 apresenta uma situação de conflito entre regras e a resolução correspondente na árvore. Esse processo se repete para todas as regras conflitantes;

14. Para evitar expansões de vértices desnecessárias, foi estabelecido um *threshold* empírico: regras com medida de precisão inferiores a 50% são descartadas da árvore;

15. As expansões ocorrem sucessivamente até que o nível da árvore atinja a quantidade de regras da fila, considerando que houve chance para que as combinações tenham sido realizadas;

16. É escolhido o vértice que apresentar o melhor custo. Este vértice contém o conjunto de hipóteses que tem o maior ganho durante a fase de busca.

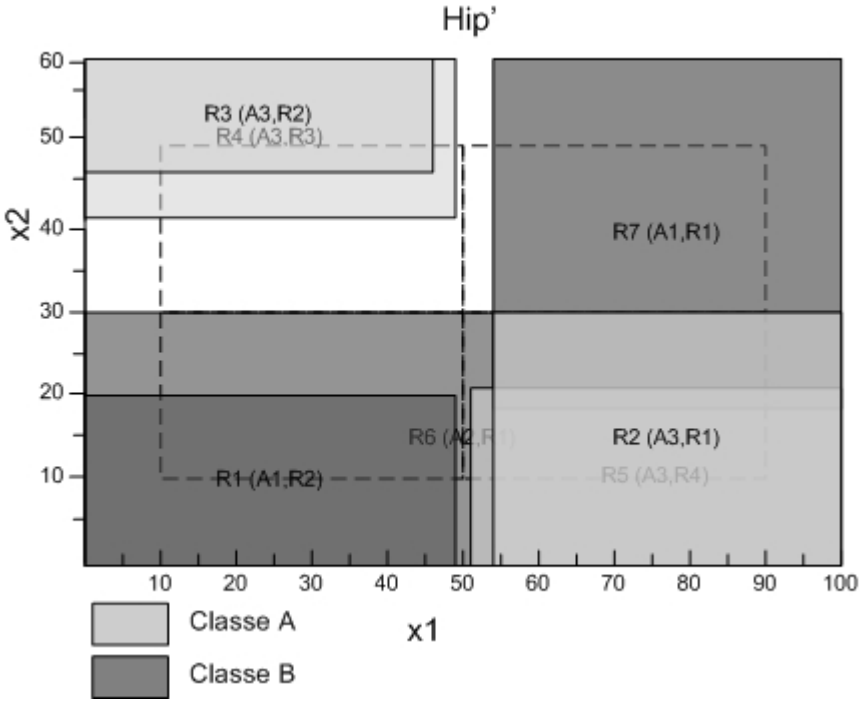

Figura 5.8: Hipótese Combinada (Hip')

A Figura 5.8 apresenta a hipótese combinada para o conjunto de exemplos de ilustração (x_1, x_2). É possível perceber que regras conflitantes foram removidas e a área co-

berta "com explicação" aumentou. Isto significa que houve aumento no poder de predição e explicação do classificador e inconsistências foram removidas.

5.5.3 Compartilhamento da Hipótese Hip'

A partir da finalização do processo de busca apresentado na seção anterior, o agente coordenador (a_{coord}) envia a hipótese Hip' para os agentes pertencentes ao conjunto $A = \{a_1, a_2, ..., a_n\}$. A hipótese Hip' é obtida a partir do vértice que tem o classificador com a "melhor" combinação.

Assim que os agentes recebem o conjunto de regras, os mesmos estão aptos a criar o mecanismo de inferência. É possível que a hipótese Hip' não cubra todos os exemplos, devido a ausência da regra *default* e a eliminação natural de regras incompatíveis. Por esta razão, os agentes criam um classificador híbrido composto pela hipótese Hip', prioritária e um mecanismo de voto simples, composto pelos classificadores de todos os agentes. A próxima seção detalha a classificação de novos exemplos pelos agentes.

5.6 Classificação de Novos Exemplos

Os agentes envolvidos no processo de combinação são dotados de capacidades distintas: coordenação, para o agente que produz a *árvore de combinação* (a_{coord}) e aprendizagem para os n agentes que criam os classificadores locais (a_i). Já a capacidade de classificação e explicação de novos exemplos está presente em qualquer agente, uma vez que eles compartilham suas hipóteses em dois momentos:

- Antes de iniciar a combinação: Os agentes de aprendizagem enviam suas hipóteses para os demais, com o objetivo de utilizá-las na classificação por voto, quando a hipótese combinada não cobrir um novo exemplo;
- Ao final do processo de combinação, quando o agente coordenador envia a hipótese combinada Hip'.

Dado um exemplo x, submetido para consulta a qualquer agente a_i pertencente ao conjunto A, o agente realiza a inferência e devolve a predição da classe e a regra disparada ou o conjunto de regras do *ensemble* que foi disparado. O algoritmo 6 mostra como a inferência é realizada. Cada agente armazena a hipótese combinada $Hip' = \{R_1, R_2, ..., R_N\}$ e o conjunto de hipóteses $H = \{h_1, h_2, h_N\}$, sendo que cada um contém um conjunto de regras ordenadas assim como descrito para Hip'.

Algoritmo 6 Classificação de Exemplos

Pré-requisito: $Hip' = \{R_1, R_2, ..., R_n\}$:hipótese combinada, $H = \{h_1, h_2, ..., h_n\}$:hipóteses dos agentes, X: exemplo a ser classificado

```
1: função classificar(Hip',H,x)
2:
3: // Modo consensual usando Hip'
4: regraEscolhida ← falso
5: i ← 1
6: enquanto negação(regraEscolhida) e i ≤ n faça
7:    R = Hip'[i]
8:    se cobertura(corpo(R), x) então
9:       regraEscolhida ← verdadeiro
10:   else
11:      i ← i + 1
12:   fim se
13:   se regraEscolhida então
14:      classe ← cabeca(R)
15:      retornar  [R,classe]
16:   fim se
17: fim enquanto
18:
19: // Modo votação
20: regrasEscolhidas ← ∅
21: para cada h ∈ H faça
22:   para cada r ∈ h faça
23:      se cobertura(corpo(r), x) então
24:         regrasEscolhidas.adicionar(r)
25:      fim se
26:   fim para
27: fim para
28: se regrasEscolhidas ≠ ∅ então
29:   classe ← cabeca(maioria(regrasEscolhidas))
30:   retornar  [regrasEscolhidas,classe]
31: fim se
```

Inicialmente o algoritmo usa a hipótese Hip' e a estratégia *best-first* para predizer a classe de X. Esta hipótese é prioritária, uma vez que pode ser entendida como um "consenso" entre os agentes de aprendizagem. Este consenso é obtido a partir das diversas avaliações que são realizadas à medida que a árvore de busca do melhor conjunto cresce. Sendo assim, do ponto de vista de explicação da predição é desejável que o novo exemplo seja coberto por esta hipótese. No entanto não há como garantir uma cobertura total, pois algumas regras conflitantes podem ter sido eliminadas durante a combinação. Além disso muitos classificadores usam uma regra *default* para exemplos não cobertos sendo usada normalmente a classe com a maior distribuição no conjunto de treinamento. Por não ter

validade do ponto de vista de explicação esta regra é eliminada assim que o processo de combinação inicia.

Para garantir a completeza do classificador nas situações onde não há cobertura, foi adotado uma técnica simples de construção de *ensembles* chamada voto [BMP06]. O processo de voto consiste em realizar N classificações, sendo que N é a quantidade de classificadores. É selecionada a classe que recebe o maior número de votos. Mesmo usando o mecanismo de voto, cada classificador utiliza a estratégia *best-first* para classificar o exemplo. O voto acontece a partir do momento em que todos os classificadores realizaram suas inferências.

5.7 Implementação

Para testar a viabilidade do modelo, a realização dos experimentos e obtenção dos resultados, o sistema *SDICCS* foi desenvolvido usando a linguagem de programação Java, o *framework* de desenvolvimento de agentes JADE [BCPR03], compatível com a especificação FIPA [FIP07] e a plataforma aberta de mineração de dados *WEKA (Ambiente para Análise de Conhecimentos Waikato / Waikato Environment for Knowledge Analysis)* [WF05]. A Figura 5.9 apresenta a arquitetura do sistema (os agentes e suas capacidades).

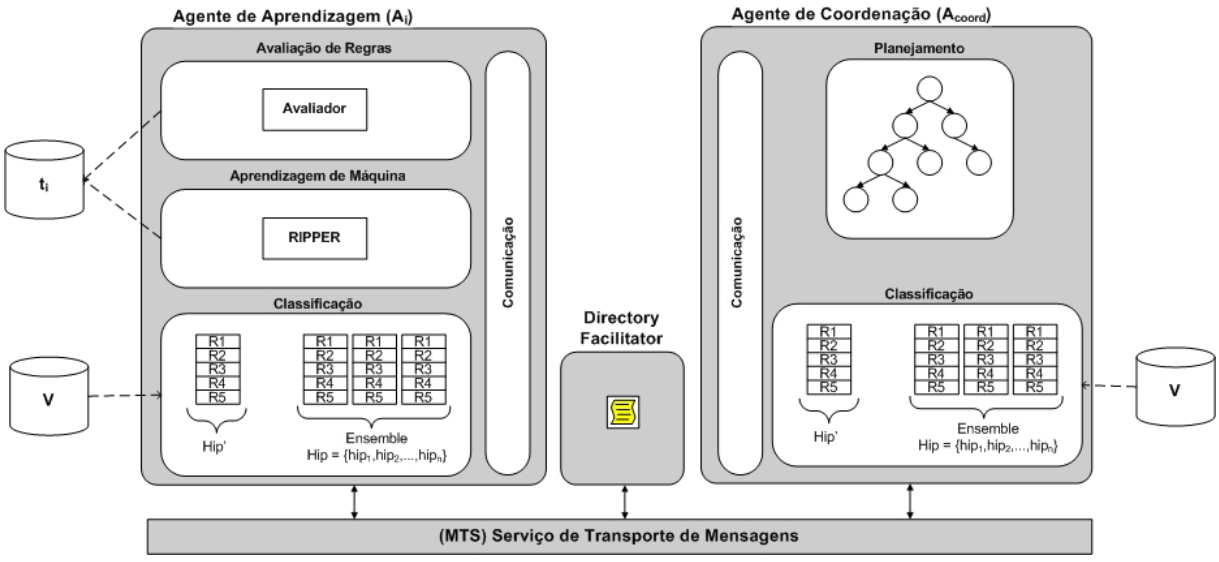

Figura 5.9: Arquitetura do Sistema *SDICCS*

O processo é iniciado no momento em que um agente, que pode ser um coordenador (A_{coord}) ou de aprendizagem (a_i) se registra na plataforma e torna-se conhecido. Segundo Liu e Williams [LW04], para que os agentes tenham capacidades sociais e de cooperação é necessário que o agente tenha uma base de relações ou *acquaintance base*. Os autores enumeram possíveis formas de um agente se tornar conhecido:

1. Registrando-se no serviço de *páginas amarelas*, como é conhecido na especificação FIPA [FIP07] ou DF (*Facilitador de Diretórios / Directory Facilitator*) como é conhecido no ambiente JADE;

2. Enviando uma mensagem para todos os agentes da plataforma (*broadcast*);

3. Registrando-se diretamente na lista de endereços do agente alvo da cooperação. Neste caso, o agente precisa ter essa capacidade implementada.

Nesta implementação, a opção 1 foi escolhida: O agente registra-se no DF assim que ele inicia seu ciclo de vida. Se for um agente do tipo coordenador, ele ficará esperando até que todos os agentes de aprendizagem entrem no ambiente. Para evitar situações de *busy-wait*[1], o que consome recursos de máquina e *polling*[2] , o que desperdiçaria recursos de rede e do serviço de troca de mensagens *MTS (Serviço de Transporte de Mensagens / Message Transport Service)*, o agente coordenador fica em estado de espera (*sleeping*), aguardando uma mensagem *inform*, com o objetivo de notificar que o agente está pronto. O agente coordenador não inicia o processo de combinação até que todos os n agentes avisem. Quando o agente coordenador precisar enviar alguma mensagem para os agentes aprendizes ele utiliza o agente DF e busca agentes com as capacidades de "aprendizes". O DF retorna uma mensagem com a performativa *inform*, retornando a lista de agentes deste tipo. A Figura 5.10 apresenta um diagrama de sequência do padrão UML (*Linguagem Unificada de Modelagem / Unified Modeling Language*) [OMG07] ilustrando a etapa de registro no ambiente.

Uma vez que todos os agentes de aprendizagem se registram no DF e enviam uma mensagem com a performativa *inform* para o agente coordenador, o processo de classificação local para os agentes é iniciado. A Figura 5.11 apresenta um diagrama de sequência com as trocas de mensagens realizadas para classificação local.

1. O agente coordenador envia uma mensagem com a performativa *request* solicitando ao DF a lista de agentes de aprendizagem;

2. O agente DF retorna uma lista de endereços dos agentes de aprendizagem;

3. Para cada agente de aprendizagem, o agente coordenador envia uma mensagem com a performativa *request* solicitando que os agentes iniciem as classificações locais;

[1]O termo busy-wait é usado para descrever processos que aguardam a mudança de estado de algum dispositivo por um certo tempo e não liberam recursos de processamento para os demais processos concorrentes.

[2]O termo *Polling* é freqüentemente usado para descrever processos que consultam o estado de um dispositivo de tempos em tempos. O dispositivo neste caso pode ser, por exemplo, uma fila de mensagens ou um servidor de e-mail.

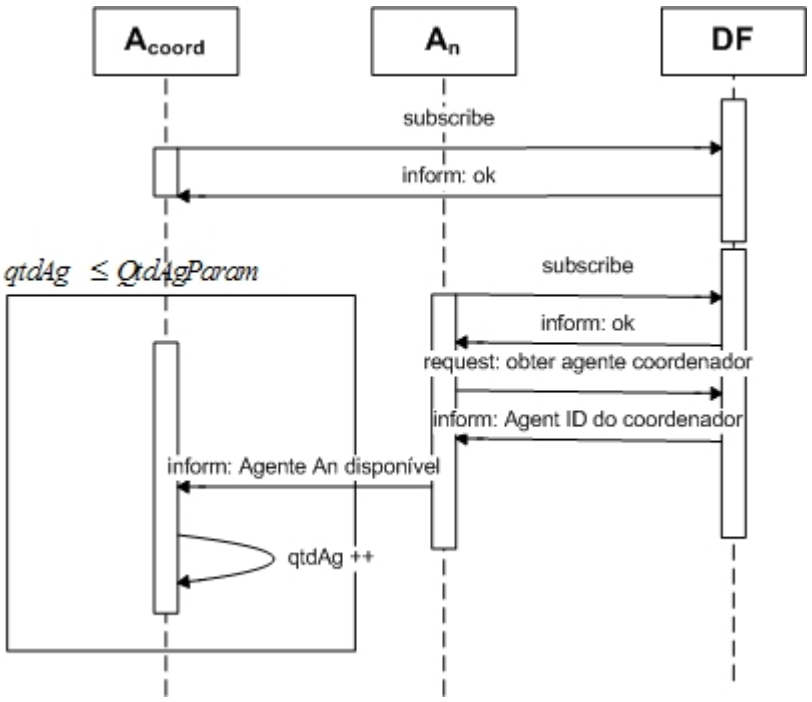

Figura 5.10: Registro no Ambiente

4. O agente coordenador fica em estado de espera até que todas as mensagens *inform* sejam recebidas;

5. Ao final, as hipóteses dos classificadores hip_i são unidas e o conjunto Hip é criado;

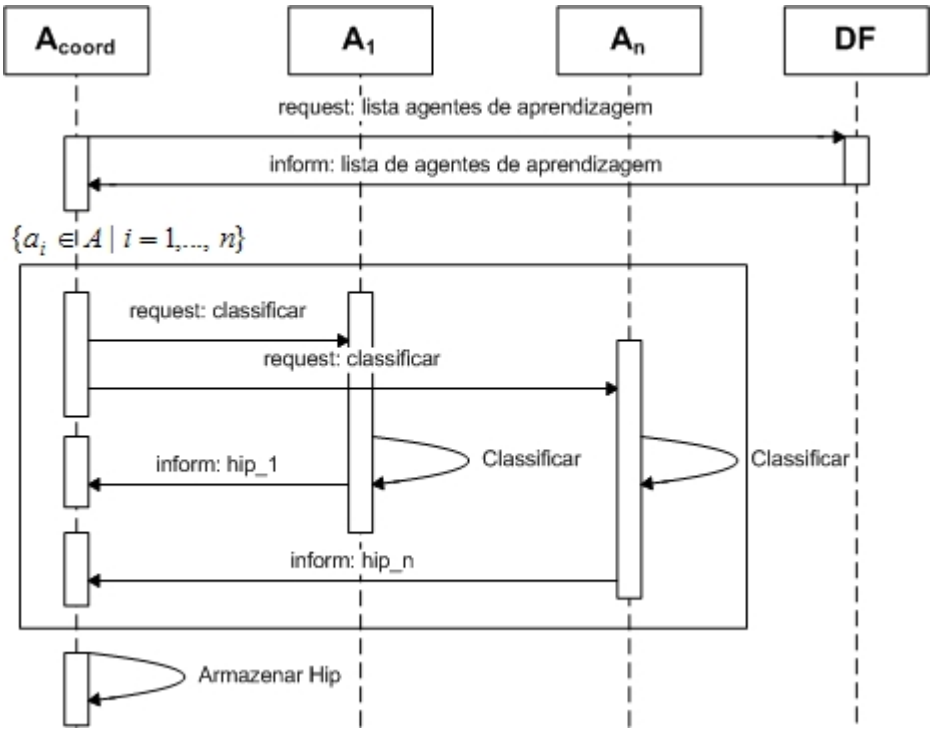

Figura 5.11: Etapa de Classificação Local

Agora que o conjunto Hip está criado o próximo passo é enviar o conjunto para os agentes de aprendizagem. O conjunto será usado para as seguintes finalidades:

- Contar a quantidade de exemplos cobertos correta e incorretamente para cada regra usando a base de treinamento;
- Identificar regras com conflito em potencial;
- Armazenar este conjunto para ser usado na etapa de classificação ($Hip' + ensemble$)

A Figura 5.12 apresenta um diagrama de sequência com as trocas de mensagens realizadas para a etapa de avaliação.

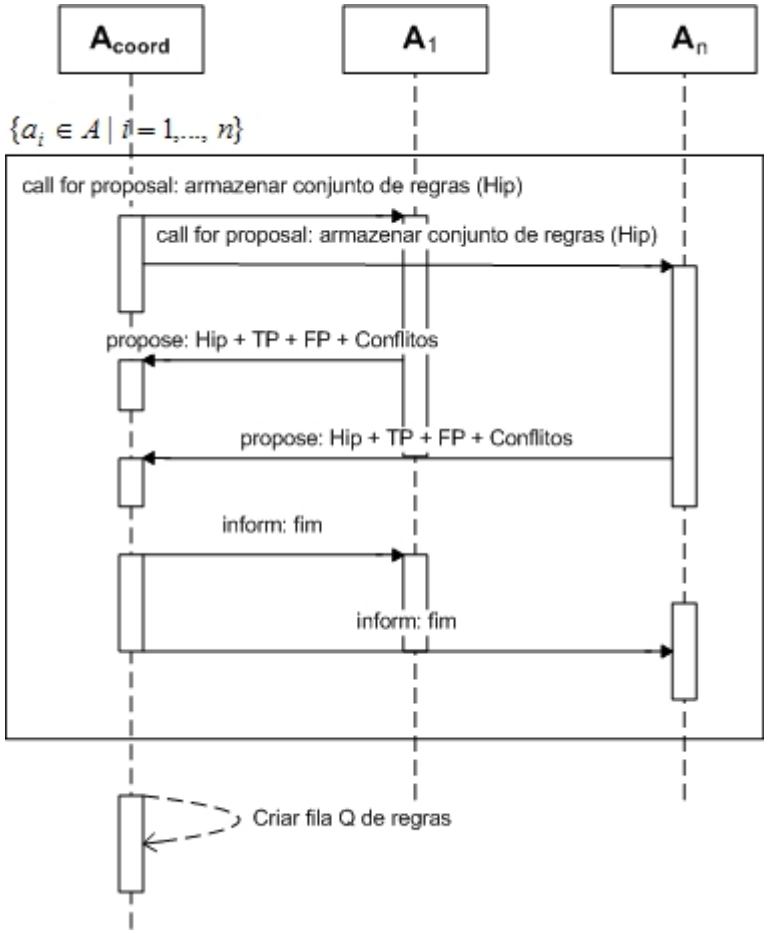

Figura 5.12: Avaliação de Regras em Hip' (Hipóteses distribuídas)

1. Usando o protocolo *Contract-net*, o agente coordenador envia as regras para os agentes avaliarem individualmente nas bases de treinamento;
2. Cada agente de aprendizagem envia a hipótese com as quantidades de exemplos cobertos correta e incorretamente para o coordenador e os conflitos usando a performativa *propose*;

3. O agente coordenador envia uma mensagem *inform* para finalizar o ciclo;

4. Após enviar todos os conjuntos, o agente coordenador soma as informações de cobertura e suporte enviados pelos agentes e cria a fila Q.

O agente coordenador ordena a fila de regras Q pela precisão obtida a partir da fórmula $\frac{TP}{TP+FP}$ e inicia a otimização usando o algoritmo do menor caminho para a montagem da *árvore de combinação*. A Figura 5.13 apresenta um diagrama de sequência com as trocas de mensagens realizadas para a etapa de otimização.

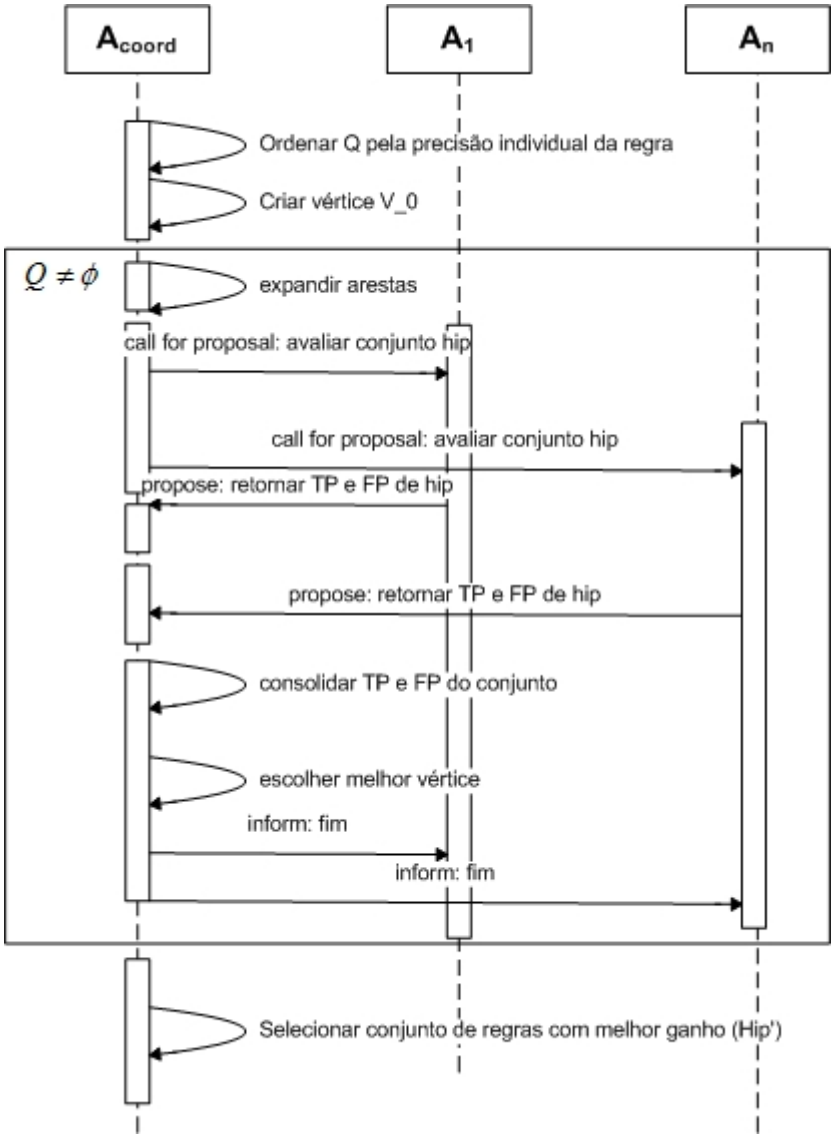

Figura 5.13: Etapa de Combinação dos Classificadores em Hip para obter Hip'

1. Continuar até que a fila Q esteja vazia (ver algoritmo 4);

2. Expandir as arestas (ver algoritmo 5);

3. Usando o protocolo *Contract-net*, o agente coordenador envia as regras obtidas pelo vértice com o melhor ganho para os agentes de aprendizagem avaliar;

4. Cada agente de aprendizagem envia a cobertura e o suporte do conjunto de regras através de uma mensagem *propose*;

5. O agente coordenador envia uma mensagem *inform* para finalizar o ciclo

6. Após sair do laço de busca, o agente seleciona o vértice com o maior ganho e extrai o conjunto combinado Hip'. Este conjunto é posteriormente enviado aos agentes de aprendizagem.

Após a finalização do processo de combinação, todos os agentes têm armazenado em sua base de conhecimento dois conjuntos de regras: um conjunto combinado e um ensemble contendo os classificadores de todo o ambiente para votação quando o conjunto combinado não cobrir um novo exemplo.

Qualquer agente, tanto coordenador quanto aprendiz possui a capacidade de classificar um novo exemplo x. Uma vez que um agente qualquer envia um *request* com o exemplo a ser classificado, o agente envia um *inform* com a predição da classe e a regra ou o conjunto de regras disparadas. A Figura 5.14 apresenta um diagrama de sequência com as trocas de mensagens realizadas.

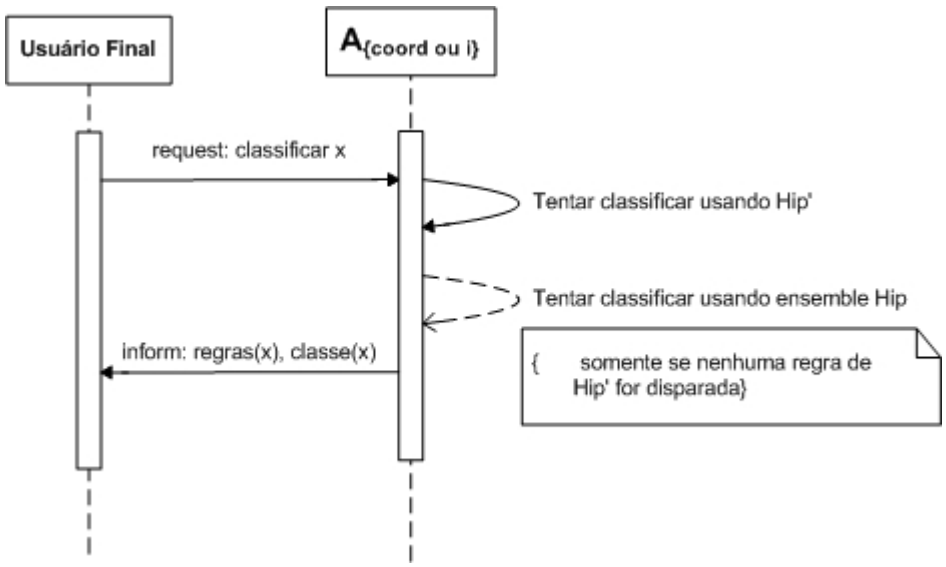

Figura 5.14: Inferência Usando o SDICCS

5.8 Trabalhos Relacionados

Existem poucos trabalhos que realizam a combinação de classificadores simbólicos ordenados. Este número ainda é menor quando o critério é a escolha de sistemas que usam técnicas de IAD. Acredita-se que a causa é a explosão combinatória gerada para combinar regras e descobrir dependências estatísticas, inviabilizando em muitos casos o processamento paralelo, distribuído e descentralizado. A necessidade de combinação leva pesquisadores a buscar heurísticas que abreviem este espaço de busca, encontrando uma solução, não necessariamente ideal, mas muito próxima dela.

O trabalho de Prati e Flach [PF05], apresenta uma alternativa para reduzir a quantidade de regras ordenadas através da análise de curvas ROC (*Receiver Operating Characteristic*). Apesar de não ser um algoritmo voltado para a mineração distribuída, busca uma combinação de regras ordenadas onde a área da curva ROC não tenha desníveis. Os resultados são encorajadores pois são comparáveis aos resultados obtidos por árvores C4.5 sem poda. Se comparado à nossa abordagem, o trabalho de Prati e Flach traz um modelo exaustivo que maximiza o ganho através das combinações. Porém a falta de uma heurística prejudica o desempenho do algoritmo e o inviabiliza em um sistema distribuído por gerar muitas trocas de mensagens. Já o trabalho proposto, apesar de não retornar o melhor resultado, abrevia a busca exaustiva favorecendo o desempenho.

O trabalho de Ana Bazzan [SB02] apresenta um trabalho de combinação de classificadores simbólicos através da cooperação entre agentes. Os agentes avaliam as regras usando a medida de Laplace para medir a qualidade individual das regras. O mecanismo de identificação de intersecções é baseado na análise dos antecedentes das regras. A ausência de um processo centralizador é um aspecto relevante, uma vez que impede gargalos de processamento. Por outro lado, a análise do antecedente da regra em bases com atributos contínuos pode levar a um alto grau de intersecção e perda de áreas de cobertura. Além disso não há garantia que todas as regras conflitantes sejam encontradas, uma vez que o conflito pode existir mesmo que com antecedentes apontando para diferentes atributos.

O trabalho de Ana Carolina Pilatti [PASE06] apresenta um modelo de integração de classificadores simbólicos independente do mecanismo de ordenação de regras. O modelo usa um agente que coordena as validações dos conjuntos de regras dos agentes. Cada agente necessita que suas regras sejam avaliadas pelos outros do grupo. Caso o outro agente aceite a regra, esta é incorporada no seu modelo, caso contrário é descartada. É usada uma fórmula baseada no grau de precisão, cobertura e intersecção da regra, usando pesos baseados em dados empíricos. Apesar do algoritmo mostrar bons resultados, é

apresentado um alto grau de instabilidade, uma vez que a regras ordenadas são usadas individualmente.

O trabalho de Hall [HCBK99] apresenta um mecanismo de mineração de dados distribuída que combina regras não ordenadas através da análise e reconstrução dos antecedentes da regra. Caso exista sobreposição, o algoritmo altera as regras conflitantes para suavizar ou remover o conflito e os exemplos não cobertos são suprimidos por uma regra default. Os erros aumentam à medida que a base de dados cresce. A razão é que os conflitos vão aumentando à medida que mais exemplos são cobertos pela regra.

Apesar de todos estes trabalhos se preocuparem com a privacidade dos dados, uma vez que não há necessidade de tráfego de dados privados pela rede, somente os classificadores, alguns possuem estratégias de busca exaustiva e outros usam diferentes medidas de avaliação de regras, o que pode comprometer a aplicação da abordagem quando as regras são ordenadas. Além disso não há uma solução única para os exemplos não cobertos, sendo usada uma regra default em muitos casos. O trabalho apresentado usa uma heurística para minimizar a quantidade de combinações de regras e tenta chegar a uma solução próxima da "melhor". Mesmo assim, é necessário realizar diversas buscas e conseqüentemente diversas trocas de mensagens e avaliações parciais.

5.9 Considerações Finais

Neste capítulo foi apresentada a metodologia para combinação de classificadores, detalhando os procedimentos realizados por esta fase. Também foi apresentado o mecanismo de cooperação utilizado pelos agentes na seção de implementação além da discussão apresentando trabalhos com características comuns a este. No próximo capítulo são apresentados os experimentos realizados e considerações sobre os resultados obtidos.

Capítulo 6

Experimentos e Resultados

Neste capítulo é apresentada uma avaliação experimental da proposta com o objetivo de comprovar sua adequabilidade na construção de um conjunto de regras que aumenta o poder descritivo de vários classificadores e também apresenta um desempenho superior ao desempenho apresentado pelos classificadores locais e comparável ao desempenho do método de voto.

Os conjuntos foram selecionados do repositório de dados da UCI [MMA97]. Bases com diferentes características foram selecionadas tais como: quantidade de atributos, quantidade de exemplos, quantidade de atributos contínuos e discretos, quantidade de classes e distribuição de classes. O objetivo é avaliar a qualidade das regras em termos de precisão, complexidade baseada na quantidade de antecedentes da regra e o percentual de cobertura da hipótese final. Na Tabela 6.1 são apresentadas as características dos conjuntos de dados utilizados nos experimentos. A coluna "Qtd Atributos" não inclui o atributo-meta. A distribuição das classes que usa o termo "desbalanceada" diz respeito à distribuição não uniforme dos exemplos em função das classes.

Tabela 6.1: Características das Bases Utilizadas nos Experimentos

Base	Qtd Exemplos	Valores Faltantes	Qtd Atributos	Atributos Nominais	Atributos Numéricos	Qtd Classes	Distribuição
Audiology	226	sim	69	69	-	24	desbalanceada
Car	1728	não	6	6	-	4	desbalanceada
Iris	150	não	4	-	4	3	balanceada
Monk 1	124	não	6	6	-	2	desbalanceada
Monk 2	169	não	6	6	-	2	desbalanceada
Segment	2310	não	19	-	19	7	balanceada
Soybean	683	sim	35	35	-	19	desbalanceada
Tic-Tac-Toe	958	não	9	9	-	2	desbalanceada
Vehicle	846	não	18	-	18	4	desbalanceada
Vowel	990	não	13	3	10	11	balanceada

6.1 Preparação dos Dados

É importante relembrar as fases do método proposto: (i) preparação dos dados, (ii) a etapa de aprendizagem local dos agentes classificadores, (iii) a etapa de combinação das hipóteses distribuídas e (iv) classificação de novos exemplos.

Em (i) os processos descritos a seguir foram executados 10 vezes, com o objetivo de permitir 10 iterações das etapas seguintes:

- Os exemplos foram embaralhados usando uma semente associada a cada iteração;
- As bases foram divididas em duas partes, usando amostragens estratificadas sem reposição: a primeira com 80% dos exemplos foi destinada ao treinamento pelos agentes. Os 20% restantes foram reservados para testar as hipóteses;
- A base de treinamento foi divida em 10 partes, cada uma com 10% da base de treinamento, usando amostragens estratificadas com reposição, sendo que cada parte foi destinada a um agente de aprendizagem, que por sua vez criou um conjunto de hipóteses. Isso significa que cada agente utilizou apenas $80 \times 0.1 = 8\%$ dos dados originais.

Um aspecto que deve ser observado é que a quantidade de exemplos para cada agente em certos casos é muito pequena e afeta diretamente a etapa (iii). Por exemplo: as bases Audiology, Iris, Monk 1 e Monk 2 receberam menos de 20 exemplos treinamento. O objetivo da divisão é criar um cenário próximo da realidade, onde os dados estão distribuídos e não há como realizar uma preparação preliminar, introduzindo espaços de tuplas incompletos e possivelmente conflitantes. A Tabela 6.2 apresenta a distribuição de exemplos por agente para cada conjunto.

Tabela 6.2: Quantidade de Exemplos de Treinamento por Agente

Base	Qtd Original	Qtd por Agente
Audiology	226	18
Car	1728	138
Iris	150	12
Monk 1	124	10
Monk 2	169	13
Segment	2310	184
Soybean	683	54
Tic-Tac-Toe	958	76
Vehicle	846	67
Vowel	990	79

6.2 Análise dos Resultados

Todos os experimentos foram realizados usando 10 iterações das bases de dados disponíveis conforme mencionado na seção anterior. Para avaliar o algoritmo, os experimentos foram comparados com os resultados obtidos nas mesmas bases de dados para os classificadores locais usando o classificador RIPPER e o método de voto simples.

Como o objetivo do método é gerar um classificador que seja compreensível e com boa taxa de acerto, as comparações se deram sob as perspectivas de: (i) compreensibilidade do modelo e (ii) taxa de acerto. Em (i) foram avaliadas a quantidade de regras geradas, complexidade e grau de suporte do classificador. Em (ii) foram avaliadas as taxas médias de acerto. Além da média dos resultados é apresentado o valor de 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.

A Tabela 6.3 apresenta as taxas médias de acerto obtidas nas 10 iterações para os classificadores SDICCS, voto simples e a média dos classificadores locais. Nesta comparação o objetivo é avaliar a taxa de acerto para o classificador global, que é composto da hipótese combinada (Hip') e do *ensemble* usando votação simples. Os melhores resultados obtidos estão destacados em negrito.

Tabela 6.3: Taxas Médias de Acerto

Base	SDICCS	Voto	Local
Audiology	$\mathbf{57.25 \pm 8.88}$	41.74 ± 10.69	35.24 ± 4.70
Car	78.32 ± 2.72	$\mathbf{79.57 \pm 1.73}$	73.64 ± 1.43
Iris	$\mathbf{94.67 \pm 3.58}$	91.00 ± 7.38	71.50 ± 4.46
Monk 1	$\mathbf{75.38 \pm 9.96}$	66.92 ± 13.35	56.62 ± 4.15
Monk 2	$\mathbf{65.29 \pm 3.87}$	62.65 ± 3.41	58.35 ± 2.97
Segment	$\mathbf{95.48 \pm 1.13}$	93.70 ± 1.67	83.00 ± 1.76
Soybean	$\mathbf{85.72 \pm 3.60}$	70.07 ± 4.71	50.63 ± 2.75
Tic-Tac-Toe	72.34 ± 4.04	$\mathbf{73.33 \pm 2.76}$	67.82 ± 2.01
Vehicle	$\mathbf{73.00 \pm 2.81}$	68.53 ± 4.76	53.78 ± 1.66
Vowel	$\mathbf{63.08 \pm 1.95}$	55.00 ± 3.68	33.59 ± 1.78

Sob a perspectiva de predição, as taxas médias de acerto se mostraram significativamente acima das execuções individuais e comparáveis com o método de voto, inclusive no grau de estabilidade apontado pelo desvio padrão.

A Tabela 6.4 apresenta os resultados médios obtidos sob a perspectiva de cobertura nas bases de teste para o conjunto de regras Hip'. Na etapa de classificação este conjunto é usado antes do voto por tratar-se de um conjunto estático e de consenso entre os agentes de aprendizagem em bases disjuntas. Além de apresentar o grau de cobertura, a tabela apresenta o percentual de exemplos cobertos pela regra *default* dos classificadores locais.

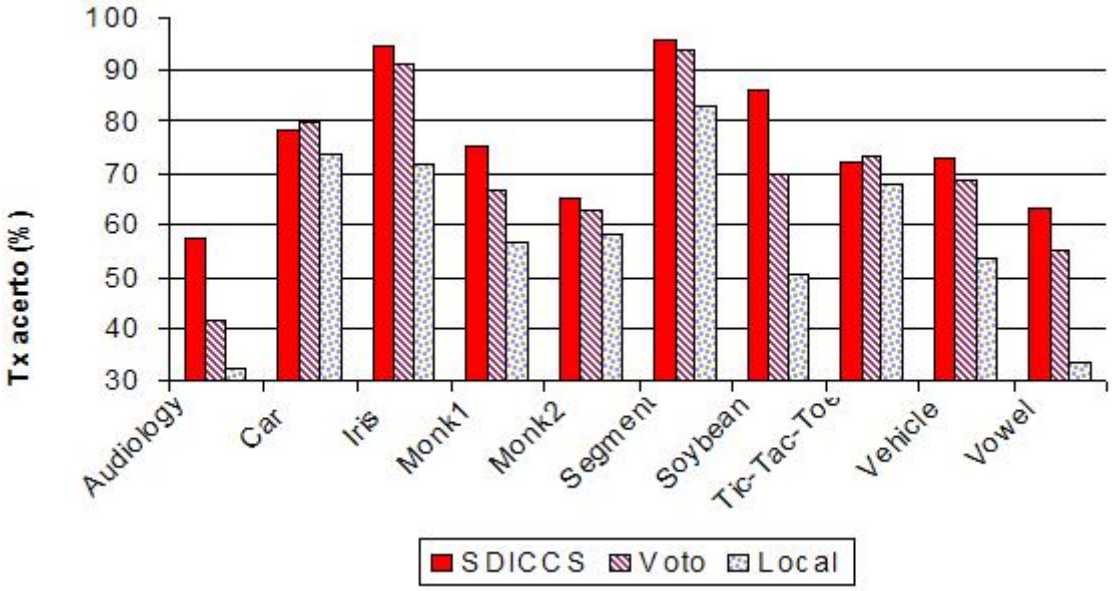

Figura 6.1: Comparação Gráfica dos Resultados

Tabela 6.4: Grau de Cobertura do Conjunto Hip' e da Regra *Default* Local

Base	SDICCS		Local
	Cobertura (%)	100 - Cobertura (%)	Cobertura Regra *default* (%)
Audiology	$73,48 \pm 15,40$	$26,52$	$70,37 \pm 7,05$
Car	$37,28 \pm 4,86$	$62,72$	$74,01 \pm 2,31$
Iris	$98,33 \pm 4,23$	$1,67$	$49,27 \pm 2,16$
Monk 1	$89,23 \pm 9,73$	$10,77$	$76,46 \pm 8,32$
Monk 2	$50,29 \pm 36,72$	$49,71$	$78,59 \pm 8,21$
Segment	$99,70 \pm 0,61$	$0,30$	$20,09 \pm 1,13$
Soybean	$97,17 \pm 3,01$	$2,83$	$37,72 \pm 3,36$
Tic-Tac-Toe	$55,57 \pm 4,39$	$44,43$	$68,45 \pm 5,38$
Vehicle	$97,94 \pm 2,19$	$2,06$	$42,84 \pm 3,52$
Vowel	$92,32 \pm 6,22$	$7,68$	$42,44 \pm 3,18$

Para todos os casos é possível perceber que a regra *default* é constantemente usada no classificador local. Isto ocorre porque o modelo é adaptado de acordo com as características e distribuição da base de treinamento. Quando as regras não são disparadas ao classificar um novo exemplo, a regra *default* é disparada.

O conjunto de regras Hip' não possui regras *default*, uma vez que há uma etapa de eliminação destas regras antes do processo de combinação. Esta ação reduz automaticamente o grau de cobertura deste conjunto. A etapa de combinação pode eliminar regras com baixo desempenho, reduzindo novamente o grau de cobertura. Em algumas bases, o classificador RIPPER gerou conjuntos que só tinham a regra *default*. Conseqüentemente, a etapa de eliminação de regras *default* desprezou o conhecimento obtido daquela base

disjunta. Este efeito reduz o grau de cobertura e aumenta a instabilidade do classificador. O sintoma pode ser observado no conjunto monk2 e audiology. Por exemplo: Na base monk2, a iteração 4 produziu 6 classificadores deste tipo. Na base audiology esse número chegou a 4 na sexta iteração. Isto também ocorreu nas bases monk1 e tic-tac-toe.

Houve um aumento do grau de cobertura em todas as bases. A coluna (100 - cobertura) apresenta os percentuais onde o classificador Hip' não atuou. Sob a perspectiva de capacidade de predição é desejável que este número seja sempre menor do que cobertura da regra *default*. Nos casos onde este número é muito próximo da cobertura da regra *default* os agentes combinaram regras com forte independência estatística. Isto pode ser verificado na base Car. Já no outro extremo, onde a quantidade de exemplos não cobertos pelo conjunto Hip' é pequeno, o algoritmo expandiu a cobertura através de regras inseridas. Foram observados casos com grau de intersecção positivo. Entende-se por grau de intersecção positivo as regras que cobrem áreas comuns e apontam para a mesma classe alvo. Com exceção da base Car, este comportamento foi percebido em todas as outras bases.

Tabela 6.5: Quantidade Média de Regras Geradas

Base	SDICCS	Local
Audiology	$7,90 \pm 3,41$	$2,74 \pm 0,44$
Car	$13,20 \pm 3,55$	$6,53 \pm 0,55$
Iris	$7,70 \pm 1,77$	$2,81 \pm 0,10$
Monk 1	$4,70 \pm 1,25$	$1,74 \pm 0,21$
Monk 2	$2,90 \pm 2,18$	$1,77 \pm 0,25$
Segment	$36,50 \pm 4,43$	$9,23 \pm 0,42$
Soybean	$29,20 \pm 2,90$	$9,89 \pm 0,47$
Tic-Tac-Toe	$8,30 \pm 2,00$	$3,63 \pm 0,31$
Vehicle	$22,60 \pm 2,80$	$5,33 \pm 0,33$
Vowel	$44,10 \pm 5,95$	$9,53 \pm 0,50$

A Tabela 6.5 apresenta a quantidade média de regras do conjunto Hip'. Foi verificado que as bases que apresentaram maior quantidade de regras possuem atributos contínuos. Algumas regras descobertas, apesar de consistentes possuem um certo grau de intersecção com outras. As regras apresentadas na figura 6.2, foram extraídas do conjunto Hip' da base Segment na primeira iteração. Como pode ser observado, essas regras podem ter intersecção para a primeira condição de ambas, que usam o atributo *intensity_mean*, caso sejam tratadas como não ordenadas. Por tratarem-se de regras ordenadas, a segunda regra funciona como um complemento da primeira: nas situações onde a regra 1 não for disparada, a regra 2 poderá ser. Elas são complementares e além disso apontam para a mesma classe alvo *window*. Este grau de intersecção é considerado positivo pois a classe alvo é a mesma.

Em geral, é possível perceber um aumento na quantidade de regras. Esse aumento

$$R1_{ag9} = \begin{cases} \text{se} & intensity\_mean \leq 22,6667 \\ \text{e} & exred\_mean \leq -10,4444 \\ \text{e} & hedge\_mean \leq 1,05556 \quad \rightarrow class = window \end{cases}$$

$$R2_{ag1} = \begin{cases} \text{se} & intensity\_mean \leq 2,96296 \\ \text{e} & rawred\_mean \geq 0,777778 \quad \rightarrow class = window \end{cases}$$

Figura 6.2: Regras da Base Segment com Intersecção

pode ser explicado pela substituição da regra *default* em muitos casos e descoberta de novas regras complementares. Entende-se por regras complementares aquelas regras que possuem conflito com outras e apresentam desempenho melhor quando combinadas, antes ou depois do conflito.

Tabela 6.6: Cobertura dos classificadores locais (Hip) e SDDICS (Hip')

Base	Cobertura Local	SDDICS	Base	Cobertura Local	SDDICS
Audiology			Segment		
Car			Soybean		
Iris			Tic Tac Toe		
Monk 1			Vehicle		
Monk 2			Vowel		

A tabela 6.6 apresenta gráficos que comparam a cobertura dos classificadores locais (Hip) e o classificador gerado pelo SDDICS (Hip'). Em todas as bases houve acréscimo na cobertura.

Tabela 6.7: Complexidade Média das Regras Selecionadas

Base	SDICCS	Local
Audiology	$1,41 \pm 0,10$	$0,73 \pm 0,14$
Car	$2,94 \pm 0,16$	$2,21 \pm 0,11$
Iris	$1,13 \pm 0,11$	$0,68 \pm 0,04$
Monk 1	$1,09 \pm 0,13$	$0,39 \pm 0,11$
Monk 2	$1,30 \pm 0,59$	$0,42 \pm 0,12$
Segment	$1,87 \pm 0,11$	$1,44 \pm 0,07$
Soybean	$1,57 \pm 0,05$	$1,18 \pm 0,02$
Tic-Tac-Toe	$2,23 \pm 0,16$	$1,35 \pm 0,13$
Vehicle	$1,72 \pm 0,13$	$1,32 \pm 0,09$
Vowel	$1,79 \pm 0,08$	$1,44 \pm 0,05$

A Tabela 6.7 apresenta a complexidade média das regras nos classificadores *SDICCS* e Local. Entende-se por complexidade de regra, a quantidade de antecedentes.

É possível perceber que nas bases Audiology, Iris, Monk1 e Monk2, a complexidade média é inferior a 1 para o modelo local, indicando que certos conjuntos tinham apenas a regra *default*. A complexidade das regras geradas pelo *SDICCS* foi maior em todos os casos basicamente por duas razões: (i) A regra *default* é substituída por conhecimento obtido em outras bases e (ii) regras genéricas geradas por classificadores disjuntos não terão mesma performance no conjunto completo pois há uma tendência de permanecer apenas regras mais específicas com alto grau de suporte.

A partir dos resultados acima relacionados, leva-se a crer que o sistema SDICCS, o qual usa um mecanismo de combinação de classificadores e de inferência baseado em um conjunto de regras combinadas e voto, resulta em valores comparáveis à técnica de voto, com a vantagem de aumentar o poder descritivo do classificador. Além disso, se mostrou significativamente superior aos modelos locais sob as perspectivas de predição e poder descritivo.

Capítulo 7

Conclusões

A utilização de técnicas de Inteligência Artificial Distribuída (IAD), uma sub-área de Inteligência Artificial (IA) que estuda técnicas de resolução distribuída de problemas (RDP) pode contribuir para aplicações em Mineração de Dados Distribuída (DDM) visando melhorar a escalabilidade, taxa de acerto, poder de predição e compreensibilidade através da cooperação entre agentes capazes de combinar classificadores e classificar novos exemplos.

O presente trabalho apresentou um sistema distribuído para mineração de dados distribuídos simulando um ambiente onde as bases de dados estão particionadas. Em cada subconjunto agentes realizaram a aprendizagem local sendo os resultados avaliados para se chegar a um conjunto de hipóteses combinado. O sistema usa cooperação para classificar novos exemplos quando o conjunto combinado não tiver cobertura.

A comparação com os métodos de classificação por voto e classificação local permitiu avaliar o modelo sob duas perspectivas:

- Predição:

  O modelo se mostrou comparável ao modelo de voto, sendo que em alguns casos, com melhores resultados. Além disso o modelo se mostrou significativamente superior se comparado aos classificadores locais.

- Poder descritivo:

  Quando comparamos com os modelos locais é possível perceber que o método preenche a lacuna gerada por regras *default*, trazendo maior legibilidade e poder descritivo ao classificador, além de expandir o conhecimento sobre bases de treinamento não conhecidas. Não há como usar esse critério de comparação com o modelo de voto, já que este modelo tem um poder de descrição baixo por não ser estático e de difícil avaliação, uma vez que o conhecimento está distribuído.

Foi observado que a quantidade de regras aumenta nas bases onde há presença de atributos contínuos. Isto ocorre devido à intersecção nos intervalos dos testes dos atributos presentes nos antecedentes da regra. Futuramente, deverá ser investigada uma abordagem para reduzir a complexidade das regras, através da alteração dos antecedentes da regra. O trabalho de [SHA00] apresenta um método com essa característica.

A principal contribuição deste trabalho foi o desenvolvimento de uma metodologia que pode ser aplicada em ambientes distribuídos para a criação de um classificador consistente e unificado, que tem poder de predição comparável ao modelo de voto e significativamente melhor do que os modelos locais. O modelo também se destaca nas situações onde é necessário uma explicação sobre a decisão tomada pelo classificador, uma vez que usa parte do conhecimento de um classificador consensual. Além disso expande a área de cobertura da regra sendo uma solução para a regra *default*.

Este trabalho traz contribuições para as áreas de algoritmos de aprendizagem simbólica, técnicas de resolução distribuída de problemas e mineração distribuída de dados.

7.1 Trabalhos Futuros

Diversas extensões deste trabalho podem ser exploradas e podem são divididas em cinco perspectivas:

- Mecanismos de Avaliação de Regras: atualmente o modelo está baseado nas medidas de precisão e suporte. No entanto abordagens que exploram, como por exemplo, as características de novidade e interessabilidade devem ser avaliadas;
- Estratégias de Busca: outras estratégias de busca heurística podem ser testadas, com enfoque na diminuição das combinações e maximização do ganho;
- Dados: diferentes distribuições do conjunto de treinamento e teste devem ser realizadas;
- Critério de Comparação: uma vez que o método se mostrou comparável ao modelo de voto, devem ser adicionadas outras abordagens de classificação para comparação.
- Mecanismos de aprendizagem: Explorar a possibilidade de usar exemplos não cobertos pelo conjunto de regras unificado como dados de treinamento para a criação de novas regras.

Como pode ser percebido, existem diversas possibilidades promissoras que certamente vão aprimorar o modelo em estudos futuros. Nesta etapa, a concentração dos esforços foram na obtenção de um modelo com taxas de acerto comparáveis ao modelo de voto e ganho no poder de compreensibilidade.

Referências Bibliográficas

[AW97] C. APTÉ and S. WEISS. Data mining with decision trees and decision rules. pages 197–210. Elsevier Science, 1997.

[BCPR03] F. BELLIFEMINE, G. CAIRE, A. POGGI, and G. RIMASSA. Jade, a white paper. *Exp - Volume 3 - n. 3*, 2003.

[BD03] J. BIGHAM and L. DU. Cooperative negotiation in a multi-agent system for realtime load balancing of a mobile cellular network. *In Proceedings of the Second International Joint Conference on Autonomous agents and multiagent systems*, pages 568–575, 2003.

[BER02] F. C. BERNARDINI. Combinação de classificadores simbólicos para melhorar o poder preditivo e descritivo de *ensembles*. Master's thesis, ICMC/USP, 2002.

[BG88] A. H. BOND and L. GASSER. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA, 1988, 1st edition, 1988.

[BG89] A. H. BOND and L. GASSER. A comparison of atms and csp techniques. pages 367–384. Morgan Kaufmann Publishers, 1989.

[BGB07] J. BACARDIT, E. GOLDBERG, and M. V. BUTZ. Improving the performance of a pittsburgh learning classifier system using a default rule. *Lecture Notes in Computer Science*, 4399:291–307, June 2007.

[BMP06] F. C. BERNARDINI, M. C. MONARD, and R. C. PRATI. Constructing ensembles of symbolic classifiers. *International Journal on Hybrid Intelligent Systems (IJHIS)*, 3(3):159–167, 2006.

[CdRFP98] Cristiano Castelfranchi, Fiorella de Rosis, Rino Falcone, and Sebastiano Pizzutilo. Personality traits and social attitudes in multiagent cooperation. *Applied Artificial Intelligence*, 12(7-8):649–675, 1998.

[CF98] Cristiano Castelfranchi and Rino Falcone. Towards a theory of delegation for agent-based systems. *Robotics and Autonomous Systems*, 24(3-4):141–157, 1998.

[CL04] F. COENEN and P. LENG. An evaluation of approaches to classification rules selection. *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 359–362, 2004.

[COH95] W. W. COHEN. Fast effective rule induction. *Proceedings of the 12th International Conference on Machine Learning (ICML'95)*, pages 115–123, 1995.

[CS95a] P. K. CHAN and S. J. STOLFO. A comparative evaluation of voting and meta-learning on partitioned data. In *International Conference on Machine Learning*, pages 90–98, 1995.

[CS95b] P. K. CHAN and S. J. STOLFO. Learning arbiter and combiner trees from partitioned data for scaling machine learning. In *Knowledge Discovery and Data Mining*, pages 39–44, 1995.

[DIE89] T. DIETTERICH. Limitations on inductive learning. *In Proceedings of the Sixth International Workshop on Machine Learning*, pages 124–128, 1989.

[DIE97] T. DIETTERICH. Machine learning research: Four current directions. *AI magazine*, pages 97–136, 1997.

[DIJ59] E W. DIJKSTRA. A note on two problems in connexion with graphs. *In Numerische Mathematik*, pages 269–271, 1959.

[DL89] E. H. DURFEE and V. R. LESSER. *Negotiating task decomposition and allocation using partial global planning.* Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1st edition, 1989.

[DL91] E. H. DURFEE and V. R. LESSER. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1167–1183, September 1991.

[DL95] K. S. DECKER and V. R. LESSER. Designing a family of coordination algorithms. *Proceedings of the Thirteenth International Workshop on Distributed AI*, pages 65–84, 1995.

[DUR99] E. H. DURFEE. Distributed problem solving and planning. *Lecture Notes in Computer Science*, pages 121–164, chapter 3, 1999.

[EV06] F. ENEMBRECK and B. C. ÁVILA. Knoma: A new approach for knowledge integration. In *11th IEEE Symposium on Computers and Communications*, pages 898–903, 2006.

[FER03] J. FERBER. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1st edition, 2003.

[FF03] J. FURNKRANZ and P. A. FLACH. An analysis of rule evaluation metrics. *Proc. 20th International Conference on Machine Learning (ICML 03)*, pages 202–209, 2003.

[FIP07] FIPA. Foundation for intelligent physical agents. `http://www.fipa.org`, 2007. Data de Acesso: 12/10/2007.

[FL98] A. FREITAS and S. H. LAVINGTON. *Mining Very Large Databases with Parallel Processing*. Kluwer Academic Publishers, 1st edition, 1998.

[FPSS96] U. M. FAYYAD, G. PIATETSKY-SHAPIRO, and P. SMYTH. From data mining to knowledge discovery: an overview. pages 1–34, Menlo Park, CA, USA, 1996. American Association for Artificial Intelligence.

[FS97] Y. FREUND and R. SCHAPIRE. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[FUR99] J. FURNKRANZ. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

[GA04] N. GATTI and F. AMIGONI. A cooperative negotiation protocol for physiological model combination. *In Proceedings of the Third Internation Joint Conference on Automomous Agents and Multi-Agent Systems*, pages 665–662, 2004.

[GL00] D. GAMBERGER and N. LAVRAC. Confirmation rule sets. *4th European Conference on Principles of Data Mining and Knowledge*, 1:34–43, 2000.

[GLK02] D. GAMBERGER, N. LAVRAC, and KRSTACIC. Confirmation rule induction and its applications to coronary heart disease diagnosis and risk group discovering. *Journal of Intelligent and Fuzzy Systems: Applications in Engineering and Technology*, 12:35–48, 2002.

[HCBK99] L. O. HALL, N. CHAWLA, K. W. BOWYER, and W. P. KEGELMEYER. Learning rules from distributed data. In *Large-Scale Parallel Data Mining*, pages 211–220, 1999.

[HK00] J. HAN and M. KAMBER. Data mining. concept and techniques. *Morgan Kaufman Publishers*, 2000.

[HY97] K. HIRAYAMA and M. YOKOO. Distributed partial constraint satisfaction problem. *Principles and Practice of Constraint Programming CP97*, pages 222–236, 1997.

[JSW98] N. R. JENNINGS, K. SYCARA, and M. WOOLDRIDGE. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, pages 7–38, 1998.

[KLM03] M. KLUSCH, S. LODI, and G. MORO. Agent-based distributed data mining: The kdec scheme. *Lecture Notes in Computer Science*, 2586:104–122, 2003.

[KPHJ00] H. KARGUPTA, B. PARK, D. HERSHBERGER, and E. JOHNSON. Collective data mining: A new perspective toward distributed data mining. *Advances in Distributed and Parallel Knowledge Discovery. AAAI/MIT Press*, pages 131–178, 2000.

[LES99a] V. R. LESSER. Cooperative multiagent systems: a personal view of the state of the art. *Transactions on Knowledge and Data Engineering*, 11(1):133–142, 1999.

[LES99b] V. R. LESSER. Cooperative multiagent systems: A personal view of the state of the art. *Knowledge and Data Engineering*, 11(1):133–142, 1999.

[LFZ99] N. LAVRAC, P. FLACH, and B. ZUPAN. Rule evaluation measures: A unifying view. In S. Dzeroski and P. Flach, editors, *Ninth International Workshop on Inductive Logic Programming (ILP'99)*, pages 174–185. Springer-Verlag, June 1999.

[LS95] P. LANGLEY and P. SIMON. Applications of machine learning and rule induction. *Communications of the ACM*, 38:55–64, 1995.

[LW04] W. LIU and M. WILLIAMS. A framework for multi-agent belief revision. *Studia Logica*, pages 291–312, 2004.

[MLH03] M. MAILLER, V. R. LESSER, and B. HORLING. Cooperative negotiation for soft real-time distributed resource allocation. *In Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 5–40, 2003.

[MMA97] C. MERZ, P. MURPHY, and D. AHA. Uci repository of machine learning databases. Disponível em: `http://www.ics.uci.edu/mlearn/MLRepository.html`, 1997.

[OCA01] J. ORTEGA, M. COPPEL, and S. ARGAMON. Arbitraining among competing classifiers using learned referees. *Knowledge and Information Systems*, pages 470–490, 2001.

[OMG07] OMG. Unified modeling language (uml) specification. `http://www.omg.org/technology/documents/formal/uml`, 2007. Data de Acesso: 12/10/2007.

[PASE06] A.C.M. PILATTI, B. C. AVILA, E. SCALABRIN, and F. ENEMBRECK. Multiagent-based model integration. *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2006 Workshops)(WI-IATW'06)*, pages 11–14, December 2006.

[PBM02] R. C. PRATI, J. A. BARANAUSKAS, and M. C. MONARD. Padronização da sintaxe e informações sobre regras induzidas a partir de algoritmos de aprendizado de máquina simbólico. *Revista Eletrônica de Iniciação Científica - Sociedade Brasileira de Computação*, 2002.

[PC00] A. PRODOMIDIS and P. K. CHAN. *Meta-learning in distributed data mining systems: Issues and Approaches*. AAAI press, 2000. Book on Advances of Distributed Data Mining.

[PF05] R. C. PRATI and P. A. FLACH. Roccer: an algorithm for rule learning based on roc analysis. *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 823–828, August 2005.

[PH96] F. J. PROVOST and D. HENNESSY. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 74–79, Portland, OR, 1996.

[PJMY02] M. Tambe P. J. MODI, W. SHEN and M. YOKOO. An asynchronous complete method for distributed constraint optimization. *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 161–168, 2002.

[QR89] J. R. QUINLAN and R. L. RIVEST. Inferring decision trees using the minimum description length principle. *Information and Computation, 80:227–248*, 1989.

[QUI93] J. R. QUINLAN. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1st edition, 1993.

[QUI95] J. R. QUINLAN. MDL and categorial theories (continued). In *International Conference on Machine Learning*, pages 464–470, 1995.

[RN04] S. RUSSEL and P. NORVIG. *Inteligência Artificial*. Elsevier Editora, 2nd edition, 2004.

[SB02] L.F. SCHROEDER and A. L. C. BAZZAN. A multi-agent system to facilitate knowledge discovery: an application to bioinformatics. *In Proceedings of the Workshop on Bioinformatics and Multi-Agent Systems, Bologna, Italy*, pages 44–50, 2002.

[SB05] C. T. SANTOS and A. BAZZAN. Integrating knowledge through cooperative negotiation - a case study in bioinformatics. *Int. Workshop on Autonomous Intelligent Agents: Agents and Data Mining*, 2005.

[SD83] R. SMITH and R. DAVIS. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[SEN06] L. G. M. SENKO. Um método baseado em lógica paraconsistente para detecção de inconsistências em classificadores à base de regras. Master's thesis, PPGIA/PUCPR, 2006.

[SHA00] N. SHAWLA. Ride: Rule learning in a distributed environment. Master's thesis, University of South Florida, 2000.

[SPT+97] S. STOLFO, A. PRODOMIDIS, S. TSELEPIS, W. LEE, and D. FAN. Jam: Java agents for meta-learning over distributed databases. *In Proceedings of the Third Internation Conference on Knowledge Discovery and Data Mining*, pages 74–81, 1997.

[TD00] L. TODOROVSKI and S. DZEROSKI. Combining classifiers with meta decision trees. *Proceedings of 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-00), Springer Verlag*, pages 54–64, 2000.

[WF05] I. H. WITTEN and E. FRANK. Data mining: Practical machine learning tools and techniques. *Morgan Kaufmann*, pages 131–178, 2005.

[WJ95] M. WOOLDRIDGE and N. R. JENNINGS. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[WOL90] D. H. WOLPERT. Stacked generalization. Technical Report LA-UR-90-3460, Los Alamos, NM, 1990.

[YOS03] K. YOSHIMURA. Ipa jack: A plugin for jack intelligent agents. *School of Computer Science and Information Technology*, 2003.

[ZJW01] F. ZAMBONELLI, N. R. JENNINGS, and M. WOOLDRIDGE. Organisational abstractions for the analysis and design of multi-agent systems. *Lecture Notes in Computer Science*, 1957, 2001.

[ZLP05] X. ZHANG, V. R. LESSER, and R. PODOROZHNY. Multi-dimensional, multistep negoriation for task allocation in a cooperative system. *Autonomous Agents and Multi-Agent Systems*, pages 5–40, 2005.