

HARVARD - Um sistema para Extracção Automática de Conhecimento

Ruy Cesar Ramos Filho

Tese apresentada para o grau de
Doutor em Engenharia Electrotécnica e de Computadores



Departamento de Engenharia Informática
Faculdade de Engenharia
Universidade do Porto
Portugal
Setembro de 2012

HARVARD - Um sistema para Extracção Automática de Conhecimento

Ruy Cesar Ramos Filho

Professor Doutor Rui Camacho

(Orientador)

Submetida para obtenção do grau de Doutor em Engenharia
Electrotécnica e de Computadores

Setembro de 2012

À

Dulclerci, Dominique, Guilherme, e minha mãe, Helena (in memoriam)

Abstract

With the use of information technologies in organisations and the widespread use of Internet services (Web, e-mail, newsgroup, among others) the amount of data available in digital form is huge and increases at a high pace each year. It is nowadays barely impossible to extract information from very large amounts of data manually. Analysis of such amounts of data requires a computational approach with large amounts of computational resources. Knowledge Discovery in Databases (KDD) has a valuable and significant contribution in the process of automatic interpretation and analysis of data. KDD techniques enable us to extract valuable knowledge from large data repositories. Using idle computational resources in a Local Area Network (LAN) we may achieve a distributed computational setting able to handle complex knowledge discovery problems. Although there are a lot of data analysis algorithms becomes necessary to combine these algorithms to distributed computing environments to analyze large amounts of data or build complex models. This thesis proposes a computational framework capable of performing knowledge discovery tasks on large amounts of data using Data Mining (DM) techniques. Machine Learning (ML) algorithms like Inductive Logic Programming (ILP) that have a highly expressive representation power can also be used. Our implementation uses several computational environments that include clusters, Grid Computing and p2p (peer-to-peer). ILP algorithms require large computational resources to process large data sets. Processing extensive Databases and providing quick solution usually requires large and expensive computational resources. Our proposal HARVARD (*HARVesting Architecture of idle machines foR Data mining*) enables the processing to be carried out using idle computers distributed in a local area network since High Performance Computers (HPC) are highly expensive and only available in a few organisations. This thesis also proposes a new approach to the parallel execution of ILP algorithms that are adequate to be used within in the HARVARD framework.

Résumé

Avec l'utilisation de technologies d'information dans les organisations et le l'utilisation étendue de services Internet (le Web, l'e-mail, le newsgroup, parmi d'autres) la quantité de données disponible dans la forme numérique est énorme et les augmentations au haut pas chaque année. Il est de nos jours impossible d'extraire les renseignements de très grandes quantités de données manuellement. Cela être beaucoup fait par la machine et bien qu'il exige de grandes quantités de quantitatifs ressources. La *Knowledge Discovery in Databases* (KDD) a un de valeur et la contribution significative dans le processus d'interprétation automatique et l'analyse de données. Les techniques de KDD nous permettent d'extraire de valeur connaissance de grands ensembles de données. Nous pouvons utiliser paresseux quantitatif les ressources dans un réseau local d'entreprise nous pouvons accomplir un distribué le cadre quantitatif capable de manipuler la découverte de connaissance complexe problèmes. Cette thèse propose un cadre quantitatif capable de l'exécution des tâches de découverte de connaissance sur la grande utilisation de quantités de données *Data Mining* (DM) techniques. La *Machine Learning* (ML) les algorithmes comme La *Inductive Logic Programming* (ILP) qui ont un pouvoir de représentation extrêmement expressif peuvent aussi être utilisé. Notre implémentation utilise plusieurs environnements quantitatifs cela incluez des groupes (*cluster*), une *Grid Computing* et p2p (peer-to-peer). ILP les algorithmes exigent à de grandes ressources quantificatives de traiter de grandes données jeux. Le traitement des Bases de données étendues et en fournissant la solution rapide exigez d'habitude des ressources quantificatives grandes et chères. Notre la proposition HARVARD (*HARVesting Architecture of idle machines foR Data mining*) permet au traitement d'être réalisée chantent des ordinateurs paresseux distribué dans un réseau local d'entreprise depuis de *High Performance Computers* (HPC) sont extrêmement chers et seulement disponibles dans quelques organisations.

Resumo

Com o advento das tecnologias de informação nas organizações e a disseminação cada vez mais popular de serviços da Internet a quantidade de dados disponíveis em suporte digital é imensa e cresce a taxas elevadíssimas a cada ano. Extrair informação de grandes quantidades de dados atualmente é uma tarefa impossível de realizar manualmente e que requer, quando realizado por máquinas, elevada capacidade computacional. A disciplina de *Knowledge Discovery in Databases* (KDD) tem uma contribuição significativa no processo de interpretação e análise automática de dados. As técnicas de KDD permitem extrair conhecimento útil de conjuntos de dados de grande dimensão. Considerando o uso de recursos computacionais de uma rede local podemos ter um ambiente de computação distribuída capaz de tratar complexos processos de extração de conhecimento. Embora existam muitas ferramentas de análise de dados torna-se necessário associar essas ferramentas a ambientes de computação distribuída para analisar grandes quantidades de dados ou construir modelos complexos. Esta tese apresenta um ambiente computacional capaz de extrair conhecimentos a partir de grandes quantidades de dados usando técnicas de *Data Mining* (DM) com a aplicação de algoritmos de *Machine Learning* (ML). O conhecimento é obtido por intermédio dos algoritmos de ML que incluem *Inductive Logic Programming* (ILP). Devido ao elevado poder computacional exigido por esses algoritmos e da necessidade de processamento de extensas bases de dados e celeridade na obtenção das respostas, a implementação do ambiente utiliza diversas abordagens computacionais de processamento distribuído, que incluem: *Clusters* e *Grid Computing*. O sistema HARVARD (*HARVesting Architecture of idle machines foR Data mining*) permite que a análise seja feita usando recursos computacionais distribuídos e ociosos de uma rede local, dispensando computadores de alto desempenho que são recursos caros e estão acessíveis para poucas organizações. Esta tese propõe ainda uma nova abordagem à execução paralela de sistemas ILP que é adequada a uma execução no ambiente HARVARD.

Apoio **FCT**: SFRH/BD/32542/2006.

Esta Tese teve o apoio da *Fundação para a Ciência e a Tecnologia - FCT*.

Agradecimentos

Agradeço, primeiramente, à minha esposa *Dulclerci* por seu amor e dedicação; e meus filhos *Dominique* e *Guilherme* por me terem acompanhado de forma incondicional neste grande desafio, demonstrando o valor de uma verdadeira família.

Agradeço ao *Prof. Rui Camacho*, meu supervisor, pelo apoio, pela troca de conhecimentos, pelo companheirismo e amizade demonstrados em todas as fases de desenvolvimento deste trabalho.

Agradeço ao *Prof. Pavel Brazdil*, Coordenador do LIAAD, pelo apoio institucional e acolhimento naquele laboratório, e assim ter permitido uma rica experiência de convivência cultural e acadêmica com os demais colegas do laboratório.

*Há um tempo em que é preciso abandonar as roupas usadas, que já tem a forma do
nosso corpo, e esquecer os nossos caminhos, que nos levam sempre aos mesmos
lugares. É o tempo da travessia: e, se não ousarmos fazê-la, teremos ficado, para
sempre, à margem de nós mesmos. (Fernando Pessoa)*

Glossário

API - *Application Programm Interface*

ARFF - *Attribute-Relation File Format*

AVL - *Attribute-Value Language*

CRISP-DM - *Cross-Industry Standard Processo for Data Mining*

DB - *Database*

DBMS - *Database Management System*

DM - *Data Mining*

DPKDD - *Distributed and Parallel Knowledge Discovery in Databases*

DW - *Data Warehouse*

HARVARD - *HARVesting Architecture of idle machines foR Data mining*

HPC - *High Performance Computing*

HTML - *Hypertext Markup Language*

IDS - *Intrusion Detection System*

ILP - *Inductive Logic Programming*

KDD - *Knowledge Discovery in Databases*

LAN - *Local Area Network*

ML - *Machine Learning*

MPI - *Message Passing Interface*

MRDM - *Multi Relational Data Mining*

OLAP - *On-Line Analytical Processing*

P2P - *peer-to-peer*

PVM - *Parallel Virtual Machine*

QoS - *Quality of Service*

RMI - *Remote Method Invocation*

XML - *Extensible Markup Language*

Conteúdo

Abstract	vii
Résumé	ix
Resumo	xi
Agradecimentos	xiii
Glossário de Termos	xv
1 Introdução	1
1.1 Objetivos	3
1.2 Contribuições	4
1.3 Avaliação e resultados	6
1.4 Organização da tese	6
2 Extração de Conhecimento, Aprendizagem Computacional e Com- putação Distribuída	7
2.1 Extração de Conhecimento - noções básicas	10
2.1.1 O Processo de Extração de Conhecimento	10
2.1.1.1 Fluxo do processo do KDD	13
2.1.2 Diferentes aspectos do processo de KDD	16
2.1.2.1 Tarefas de <i>Data Mining</i>	16

2.1.3	Medidas de Interesse	19
2.1.4	Algoritmos para as tarefas de DM	21
2.1.4.1	Regras de Associação	21
2.1.4.2	Classificação	22
2.1.5	Extração de Conhecimento - implementação	23
2.1.6	Questões de Pré-processamento em Extração de Conhecimento	25
2.2	Extração de Conhecimento - métodos e ferramentas	29
2.2.1	A metodologia CRISP-DM	30
2.2.2	Implementações em SGBD	32
2.3	Aprendizagem Computacional	35
2.3.1	Indução de Árvores de Decisão	36
2.3.2	Indução de Programas em Lógica	37
2.3.2.1	Implementações ILP em computação distribuída . . .	43
2.3.3	Data Mining Multi-relacional	44
2.3.4	Ferramentas para Aprendizagem Computacional	48
2.3.4.1	Linguagem e ambiente R	48
2.3.4.2	Weka	49
2.3.4.3	Yale	50
2.3.4.4	Rapidminer	51
2.3.4.5	SPSS Clementine	51
2.3.4.6	SAS Enterprise Miner	52
2.3.4.7	KNIME	53
2.4	Computação Distribuída	53
2.4.1	Computação em <i>Cluster</i>	53
2.4.2	Grid Computing	56
2.4.3	Iniciativas <i>Grid Computing</i>	61
2.4.3.1	Globus	62
2.4.3.2	Condor	62

Conteúdo	xix
2.4.3.3 BOINC	63
2.4.3.4 Hadoop	64
2.4.3.5 MapReduce	65
2.4.4 Computação Peer-to-peer(p2p)	66
2.4.5 Computação em Nuvem (<i>Cloud Computing</i>)	67
2.5 Conclusões	69
3 Sistema HARVARD	71
3.1 Arquitetura do Sistema HARVARD	73
3.2 Processo de Análise de Dados no HARVARD	75
3.3 A especificação das tarefas	77
3.4 O uso da linguagem XML no HARVARD	79
3.5 A estrutura do HARVARD	80
3.5.1 O Servidor	80
3.5.2 Cliente	82
3.5.3 Funcionamento guiado por eventos	84
3.6 Extensões ao Sistema HARVARD	85
3.6.1 Macro Tarefas	85
3.6.2 HARVARD e o ambiente <i>Grid Computing</i>	88
3.7 Conclusões	97
4 Contribuição para Data Mining Relacional	99
4.1 Uma nova abordagem para uma execução paralela de ILP	100
4.2 Implementação rápida	105
4.3 Avaliando o algoritmo	107
4.4 Melhoramentos para a implementação	110
4.5 Comparando com outras implementações	111
4.6 Conclusões	111

5	Utilização e Avaliação do Sistema HARVARD	113
5.1	Utilização em pré-processamento de <i>Data Mining</i>	114
5.2	Utilização em tarefas de análise de dados	119
5.3	Executando um sistema de ILP no HARVARD	121
5.4	Conclusões	123
6	Conclusões	125
6.1	Contribuições	126
6.2	Trabalho futuro	127
	Bibliografia	131
	Apêndices	139
A	Lista de Traduções	139
B	Resultados do ILP	141
C	Exemplos e Scripts	151
	Índice	174

Lista de Figuras

2.1	Processo de Extração de Conhecimento [FPSSU96]	11
2.2	<i>Knowledge Discovery in Databases e as disciplinas envolvidas</i> [FPSSU96] 12	
2.3	Etapas de pré-processamento, <i>Data Mining</i> e pós-processamento . . .	14
2.4	Tarefas de <i>Data Mining</i> : <i>classificação (a)</i> , <i>clustering (b)</i> , <i>regressão</i> <i>(c)</i> e <i>outlier detection (d)</i>	18
2.5	Etapas da aplicação do algoritmo Apriori conforme [AS94]	23
2.6	Etapas de um algoritmo de Classificação, adaptado de [Han01]	24
2.7	<i>Data Mining</i> Distribuído. Divisão dos dados inicialmente centraliza- dos.	26
2.8	Pré-processamento dos dados	28
2.9	Metodologia CRISP-DM conforme [CCK ⁺ 00]	31
2.10	Representação de Árvore de Decisão	37
2.11	Representação de dados relacionais em cláusula de 1 ^a ordem em sin- taxe Prolog	41
2.12	Uso de técnica ILP para Extração de Conhecimento	43
2.13	Esquemas simplificados de troca de mensagens dos diferentes tipos de algoritmos paralelos. (figura original de [FSSC09])	45
2.14	Tabela única com dados redundantes	47
2.15	Tabelas de Clientes, de Compras e de Transações Consolidadas repre- sentando um processo de junção de dados.	48

2.16	Interface <i>Explorer</i> do WEKA	50
2.17	Visualização de resultados no Rapidminer	52
2.18	Visão de um fluxo de análise de dados no <i>KNIME</i>	54
2.19	Representação de um <i>Cluster</i>	55
2.20	Arquitetura do Projeto <i>Hadoop</i> [Whi09]	65
2.21	Fluxo exemplo de execução das funções de mapeamento (<i>Map</i>) e redução (<i>Reduce</i>) de um conjunto de dados [Whi09]	66
2.22	Diagrama de execução das funções de mapeamento (<i>Map</i>) e redução (<i>Reduce</i>) de um conjunto de dados de forma recursiva [Whi09]	67
3.1	Arquitetura HARVARD	73
3.2	Descrição do fluxo do processo KDD em tarefas sequenciais e paralelas.	76
3.3	Descrição em detalhe de uma tarefa de KDD específica.	78
3.4	Exemplo de especificação de uma tarefa em UT usando a linguagem XML.	81
3.5	Exemplo da codificação em XML de descrição da máquina cujo o <i>hostname</i> é “tau4”.	82
3.6	Arquitetura HARVARD-g: (1) processamento local do HARVARD; (2) execução de tarefas em ambiente <i>Grid Computing</i> remoto	89
3.7	Módulo Conexão Grid (CGrid) e submódulos	91
3.8	Representação dos módulos do HARVARD-g (versão Globus)	93
3.9	Representação dos módulos do HARVARD-g (versão Condor)	95
3.10	Classes disponíveis na Condor-API-Java, descrição original	95
3.11	Exemplo de utilização da Condor-API-Java	96
3.12	Diagrama de compartilhamento de recursos: (1) Base de Dados e o Repositório de Algoritmos DM acessíveis para o HARVARD e o Condor	97

Lista de Tabelas

2.1	Tarefas de <i>Data Mining</i> e algumas técnicas utilizadas	19
2.2	Classificação das funcionalidades de Extração de Conhecimento disponíveis em alguns SGBD	33
4.1	Caracterização dos conjuntos de dados. Os dois valores nas células da coluna "número de exemplos" indicam o número de exemplos positivos e negativos respectivamente.	108
4.2	As 5 ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>mutagenesis</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	108
4.3	As 4 ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>carcinogenesis</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	109
4.4	<i>Speedups</i> para diferentes valores de parâmetro <i>language</i> no conjunto de dados <i>mutagenesis</i> . Os valores indicados são a média e desvio padrão de 5 sequências de saturação/redução.	110
5.1	Caracterização do conjunto de dados NSL-KDD usado para ilustrar a macro tarefa de <i>feature subset selection</i> implementada para conjuntos de dados no formato ARFF.	116

5.2	Caracterização do conjunto de dados relacional <i>carcinogenesis</i> usado para ilustrar a macro de <i>feature subset selection</i> implementada para conjuntos de dados no formato usado pelo Aleph.	117
5.3	Caracterização do conjunto de dados CPDBAS usado para ilustrar a macro tarefa de sintonização de parâmetros implementada para conjuntos de dados no formato ARFF.	119
5.4	Caracterização do conjunto de dados lowbwt usado para ilustrar a macro tarefa de validação cruzada implementada para conjuntos de dados no formato ARFF.	120
B.1	Ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>Triazines</i> . O <i>recall number</i> (número máximo de soluções alternativas para predicados não determinísticos) não está mostrado nas declarações de modo pois só os tipos são relevantes. <i>modeh()</i> é a declaração de modo da cabeça da cláusula.	142
B.2	Parte 1 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>CPDBAS</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	143
B.3	Parte 2 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>CPDBAS</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	144
B.4	Parte 3 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>CPDBAS</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	145
B.5	Parte 4 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>CPDBAS</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	146

B.6	Parte 1 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>DBPCAN</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	147
B.7	Parte 2 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>DBPCAN</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	148
B.8	Parte 3 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>DBPCAN</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	149
B.9	Parte 4 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados <i>DBPCAN</i> . O <i>recall number</i> não está mostrado nas declarações de modo pois só os tipos são relevantes.	150

Lista de Algoritmos

1	Calcula as <i>ilhas</i> a partir das declarações de modo	104
2	Algoritmo de Indução do tipo Cobertura Gananciosa	106

Capítulo 1

Introdução

“A mathematician is a device for turning coffee into theorems.”

– *Paul Erdős*.

O crescente uso das tecnologias da informação associado ao crescimento e desenvolvimento económico de vários setores (negócios, governo, comunidade científica e académica, entre outros) tem permitido que as organizações e a sociedade em geral reúnam uma enorme quantidade de dados e informações sistematicamente em Bases de Dados (DB) [FPSSU96]. Tudo passa a ser registado e armazenado, desde uma simples transação de compra e venda a uma biblioteca inteira, tudo pode ser mantido em suporte digital. Dados e informações passam a estar disponíveis para futura consulta ou simplesmente são armazenados como comprovativo de uma transação. No entanto, ter muitos dados não significa necessariamente ter muita informação - este o dilema por resolver para muitas organizações. Armazenar grandes quantidades de dados "em bruto" não representa necessariamente uma vantagem competitiva de uma empresa sobre os competidores, ou mais conhecimento. Para rentabilizar os dados armazenados é necessário extrair informação e conhecimento que estão implícitos nos dados e que sejam úteis.

Do ponto de vista humano, torna-se praticamente impossível interpretar, ana-

lisar e obter resultados a partir de grandes quantidades de dados sem o auxílio de computadores. Torna-se muito difícil diferenciar informações verdadeiramente importantes e úteis em tão grandes quantidades de dados [FPSSU96, Han01].

Certamente, muitos desses dados servirão apenas de registo histórico para comprovar determinada transação num determinado instante. No entanto, para a atividade empresarial, a análise dos dados armazenados pode trazer uma vantagem competitiva. Quando ocorrer a necessidade de analisar e interpretar determinado conjunto de dados com a intenção de responder a determinada questão ou planejar racionalmente uma qualquer tarefa, certamente milhares ou até milhões de registos deverão ser tratados de modo a permitir um resultado informativo adequado e útil. Neste caso, a aplicação de técnicas automáticas e métodos capazes de analisar os referidos dados tornam-se inevitáveis, dada a complexidade e a extensão das BD. Análises simples podem ser feitas com o auxílio de técnicas de OLAP (*On Line Analytical Processing*). No entanto, as análises mais complexas requerem novas abordagens.

Neste contexto, a disciplina de *Knowledge Discovey in Databases* (KDD) tem uma contribuição significativa no processo de interpretação e análise de dados. As técnicas de KDD incluem o pré-processamento de dados e a aplicação de algoritmos que extraem padrões úteis de um determinado conjunto de dados. É fundamental extrair informação que não seja obtida por simples consulta ou observação, esteja implícita, seja desconhecida e potencialmente útil [FPSSU96, Han01].

A aplicação de técnicas de KDD para auxiliar a análise de grandes quantidades de dados torna-se, hoje em dia, um caminho viável e aplicável, usando para isso os algoritmos de *Aprendizagem Computacional (AC)* ou *Machine Learning (ML)*.

Os algoritmos de AC foram originalmente construídos para abordagens sequenciais monolíticas pressupondo um ambiente computacional centralizado. Nesta abordagem "tradicional" é necessário que os dados estejam todos num único local para que as técnicas de AC possam ser aplicadas. Entretanto, a quantidade de dados

atualmente disponível é gigantesca e realizar a sua análise numa única máquina torna-se oneroso pois os recursos computacionais poderosos são caros. Há casos em que os dados dispersos não podem ser centralizados seja por: limitação de rede, espaço para armazenamento, poder computacional, questões de segurança, confidencialidade ou ainda por privacidade dos dados. Estas situações requerem novas abordagens para o tratamento dos dados nos, ou próximo dos, respectivos locais de armazenamento.

Apesar da grande quantidade de algoritmos e ferramentas de análise de dados, não tem havido a preocupação de associar esses algoritmos com ambientes de computação distribuída. Menos ainda de tornar essa associação fácil de utilizar. Esta tese aborda essa questão premente de associar, de forma fácil, algoritmos de análise de dados com computação distribuída com o intuito de tornar fácil/possível a avaliação de grandes quantidades de dados e a construção de modelos mais complexos.

1.1 Objetivos

Nesta tese propomos a utilização de um ambiente computacional capaz de: *i)* servir de suporte à extração de conhecimento a partir de grandes quantidades de dados; *ii)* realizar a extração de conhecimento a baixo custo, preferencialmente sem necessidade de recursos computacionais especiais e caros; *iii)* realizar tarefas de KDD num ambiente empresarial ou campus universitário sem perturbar seu normal funcionamento; *iv)* proporcionar um ambiente em que o utilizador realize as tarefas de KDD de modo fácil e sem grandes conhecimentos de KDD e computação distribuída. Para isso propomos o uso de técnicas de KDD com a aplicação de algoritmos de *Aprendizagem Computacional* e *Computação Distribuída*.

A implementação proposta utiliza uma abordagem de computação distribuída em LAN (*Local Area Network*). Poderá ainda fazer uso de um ambiente de *Grid Computing*, devido ao elevado poder computacional exigido por alguns dos algoritmos de *Aprendizagem Computacional*, da necessidade de processamento de extensas Ba-

ses de Dados, ou da utilização de ferramentas (como os algoritmos de ILP) que permitem a construção de modelos complexos para os dados. Tudo isso usando recursos computacionais distribuídos e ociosos de uma rede local, já que computadores de alto desempenho (super-computadores) são notoriamente recursos caros e geralmente disponíveis em poucas organizações.

1.2 Contribuições

Com esta abordagem construímos uma plataforma computacional que tem como características essenciais, quanto ao uso de recursos, a escalabilidade, a heterogeneidade e a adaptabilidade. É escalável enquanto puder fazer uso de recursos disponíveis, gerando um poder computacional compatível com a exigência das aplicações de KDD. Em complemento, é adaptativo e heterogêneo por fazer uso de recursos computacionais variados em diferentes sistemas operativos (Linux e Windows) com a adoção da plataforma Java como ambiente de integração e portabilidade.

É um ambiente de baixo custo pois utiliza recursos computacionais disponíveis e ociosos de uma organização, e portanto não perturba o normal uso dos recursos na rotina diária. É uma plataforma flexível e versátil pois utiliza diferentes ferramentas pré-existentes de análise de dados sem re-programações ou adaptações, sendo completamente independente da ferramenta requerida para a tarefa de KDD. É ainda uma plataforma fiável já que incorpora características de segurança e controle das operações e com esquemas de tolerância a falhas. Fácil de utilizar já que dispõe de uma linguagem de especificação de tarefas de KDD simultaneamente poderosa e de fácil utilização por não especialistas.

Em resumo, a plataforma computacional apresenta as seguintes características:

- escalável;
- baixo custo;
- adaptável;

- versátil;
- fiável;
- não perturba o ambiente organizacional onde é executada;
- flexível e versátil;
- fácil de utilizar.

Independentemente desta plataforma computacional, foi dada uma contribuição para a realização de análise complexa com a proposta de uma nova abordagem à execução de algoritmos de ILP em ambiente distribuído. Esta abordagem é adequada para ser executada na plataforma computacional proposta.

Como diferencial deste trabalho podemos destacar que:

- (a) proposta de uma nova abordagem de ILP Paralelo e sua interligação com o ambiente computacional proposto, permite a utilização de um sistema de ILP, em tarefas de *Multi-Relational Data Mining* (MRDM) em grandes bases de dados do mundo real superando as questões da exigência de elevado poder computacional;
- (b) o uso de recursos computacionais distribuídos e ociosos como uma alternativa para aplicações que necessitam elevado poder computacional;
- (c) o desenvolvimento de uma linguagem de descrição do processo de KDD pode viabilizar a disseminação desta tecnologia entre investigadores não habituados com o assunto, mas interessados em explorar conhecimentos “desconhecidos” em bases de dados;
- (d) interligação do sistema com ambientes *Grid Computing*;
- (e) utilização fácil de algumas tarefas rotineiras mas computacionalmente exigentes em ambiente distribuído (operações como o *cross validation*, *feature subset selection*, *parameter tuning* entre outras).

1.3 Avaliação e resultados

A fim de avaliar o trabalho desenvolvido tanto na construção da plataforma de Extração de Conhecimento como no algoritmo de ILP paralelo desenvolvemos uma série de Casos de Estudo que são detalhados no Capítulo 5. Neste Casos de Estudo foram testadas diversas tarefas de KDD em que a computação distribuída e em particular o sistema HARVARD pode ser extremamente útil. Foram utilizadas tarefas de pré-processamento, avaliação de modelos, sintonização de parâmetros, escolha automática do melhor conjunto de atributos, entre outros. Foram usadas tarefas de Classificação e Regressão num conjunto diversificado de dados. As avaliações centraram-se não só em algoritmos proposicionais de Aprendizagem Computacional mas também na utilização de sistemas de ILP. Foi ainda utilizada a ligação a um ambiente de Cluster, como o Condor, para a execução das tarefas. Os resultados são bastante promissores.

1.4 Organização da tese

Para além deste capítulo introdutório a tese tem a seguinte estrutura. O Capítulo 2 efetua uma revisão dos conceitos básicos e abordagens existentes em termos como *Extração de Conhecimento*, *Aprendizagem Computacional* e *Computação Distribuída* relevantes para a compreensão do trabalho de tese. No Capítulo 3 é apresentado o projeto de ambiente computacional para Extração de Conhecimento denominado HARVARD. O Capítulo 4, descreve a nova abordagem de ILP Paralelo adequado à execução em ambiente HARVARD. Exemplos de utilização e a avaliação das propostas apresentadas nesta dissertação são apresentados no Capítulo 5. O último capítulo apresenta as conclusões do trabalho, a sua aplicabilidade e importância no âmbito organizacional e na comunidade científica, e sugestões para futuros trabalhos.

Capítulo 2

Extração de Conhecimento, Aprendizagem Computacional e Computação Distribuída

“entities should not be multiplied beyond necessity.”

– *Ockham's razor*.

Os avanços da tecnologia da informação nos últimos anos tem permitido às empresas armazenar grandes quantidades de dados em suporte digital. A própria Internet tem contribuído para a geração e armazenamento de uma surpreendente quantidade de informação. Diferentes serviços, entre os quais *e-mail*, *newsgroup*, *ftp*, *web*, *blog*, permitem reunir informação de diferente natureza, de carácter pessoal ou organizacional. Em alguns desses serviços o registo e o partilhamento da informação nem sempre está organizado ou estruturado em forma de DB, exigindo um esforço adicional para recuperar a informação desejada.

Por outro lado a capacidade humana de analisar e mesmo extrair informação útil é bastante limitada sobretudo quando envolve uma grande quantidade ou complexidade de dados ou quando é diversificado o número de fontes e formatos

em que se pode encontrar a informação [Han05]. É habitual a informação estar armazenada em ficheiros com diferentes formatos e conteúdos: textos, áudios, vídeos, imagens ou fotografias. Uma simples consulta ou busca de informação num sítio de indexação eletrônica, como *Google* ou *Yahoo*, tende a tornar-se uma tarefa bastante complexa. Há sempre a possibilidade do indexador ou provedor de informação apresentar como resultado um conjunto significativo de páginas *Web* contendo, ou não, a informação desejada.

No contexto da análise de quantidades consideráveis de informação em simultâneo uma variedade de fontes e formatos surgiram estudos recentes que contribuíram para a área denominada de *Knowledge Discovery in Database (KDD)*, ou simplesmente *Extração de Conhecimento*¹ (*EC*).

Extração de Conhecimento é definido com sendo o processo de descoberta ou reconhecimento de padrões ou regularidades num determinado conjunto de dados. Os padrões identificados devem apresentar algum significado, permitir *insights* contribuindo para a tomada de decisões a partir deles, e assim obtendo-se alguma vantagem – quer sejam novas informações ou novos conhecimentos [FPSSU96, FU96, Han01].

A principal motivação para o uso de *Extração de Conhecimento* é a análise de grandes quantidades de dados para o suporte à tomada de decisão. Há inúmeros campos de aplicação e diversos domínios do conhecimento humano em que as técnicas de *Extração de Conhecimento* têm sido usadas com bastante sucesso. Essas áreas de conhecimento incluem a síntese de novos medicamentos, a identificação de fraudes ou definição de novas estratégias de vendas de produtos [FPSSU96, Han01]. Alguns outros exemplos que merecem destaque:

¹Sempre que não haja ambiguidade usaremos o termo *Extração de Conhecimento* para designar todo o processo de KDD bem como o seu passo intermédio de *Data Mining*.

- *gestão e análise de mercados*: algumas estratégias de marketing amplamente usadas como segmentação de mercado, venda cruzada (*cross selling*), *market basket* só podem ser implementadas na prática com o uso efetivo de um processo de *Extração de Conhecimento* sobre BD de clientes, transações, produtos e fornecedores. A análise de uma BD de clientes e suas transações permite identificar os produtos adequados a um determinado grupo e também definir critérios que permitirão atrair novos clientes;
- *detecção de fraudes financeiras e administrativas*: a partir da análise de registos históricos sobre operações financeiras pode-se, por exemplo, construir um modelo de comportamento fraudulento e assim identificar futuros casos similares de fraude. Pode-se identificar padrões incomuns de consumo no uso de um cartão de crédito; detectar grupos de pessoas que possam estar simulando acidentes para receber o prémio do seguro; transações bancárias que demonstrem irregularidades, pacientes ou mesmo médicos que possam estar defraudando o seguro de saúde através de tratamentos incompatíveis com a idade, sexo ou estado do paciente.
- *text mining*: a quantidade de dados e informações disponíveis na *Web* em diferentes serviços como *newsgroup*, *e-mail* e ficheiros eletrónicos e em diferentes formatos como: HTML (*Hypertext Markup Language*), PDF (*Portable Document File*), *ppt* (*Microsoft Powerpoint*), *Postscript*, entre outros, torna o processo de classificar e recuperar informação bastante complexo e oneroso. Um exemplo interessante é na área biomédica, onde o volume de publicações tem aumentado consideravelmente, principalmente na geração pós-genoma, chegando a registar a marca de 400.000 artigos por ano, somente na PubMed² [Cock03]. A simples pesquisa ou análise de resumos, nesta grande BD, pode já não ser tão eficaz para novos avanços na própria área sem uma classificação

²PubMed Central é repositório de artigos da área de ciências biomédicas e da vida e é mantido pelo NIH (*U.S. National Institutes of Health*)

mais adequada dos artigos publicados. É requerido identificar relações, buscar novos padrões e conexões entre os vários artigos visando um melhor aproveitamento e seleção da informação, e evitar assim a redundância de esforços.

Estes são apenas alguns exemplos e formas de como a *Extração de Conhecimento* pode auxiliar na tomada de decisão. Outra situação interessante que deverá lançar grandes desafios à comunidade científica é a entrada em operação do *LHC* (*Large Hadron Collider*) do CERN³ que deverá gerar *10 petabytes* de dados por ano. Os dados produzidos a partir de experiências com o *acelerador de partículas* deverá requerer não só técnicas de *Data Mining* para “descobrir” novas informações mas também poder computacional relevante. O próprio CERN tem um projeto paralelo de montar um sistema baseado em *Grid Computing* para suportar todo o processamento dos dados gerados. O próprio armazenamento de todos esses dados é complexo e oneroso.

2.1 Extração de Conhecimento - noções básicas

2.1.1 O Processo de Extração de Conhecimento

Knowledge Discovey in Database (KDD) ou Extração de Conhecimento⁴, é um processo não trivial de identificar, validar e reconhecer padrões de dados que possam prover informação válida gerando conhecimento útil e inexplorado sobre uma determinada Base de Dados. Os dados referem-se a representação de factos e os padrões são definidos por uma expressão que descreve um subconjunto desses mesmos dados [FPSSU96].

³URL do CERN com detalhe sobre o projeto: <http://lcg.web.cern.ch/LCG/>

⁴Embora a tradução direta de KDD seja Descoberta de Conhecimentos em Base de Dados usaremos a definição Extração de Conhecimento.

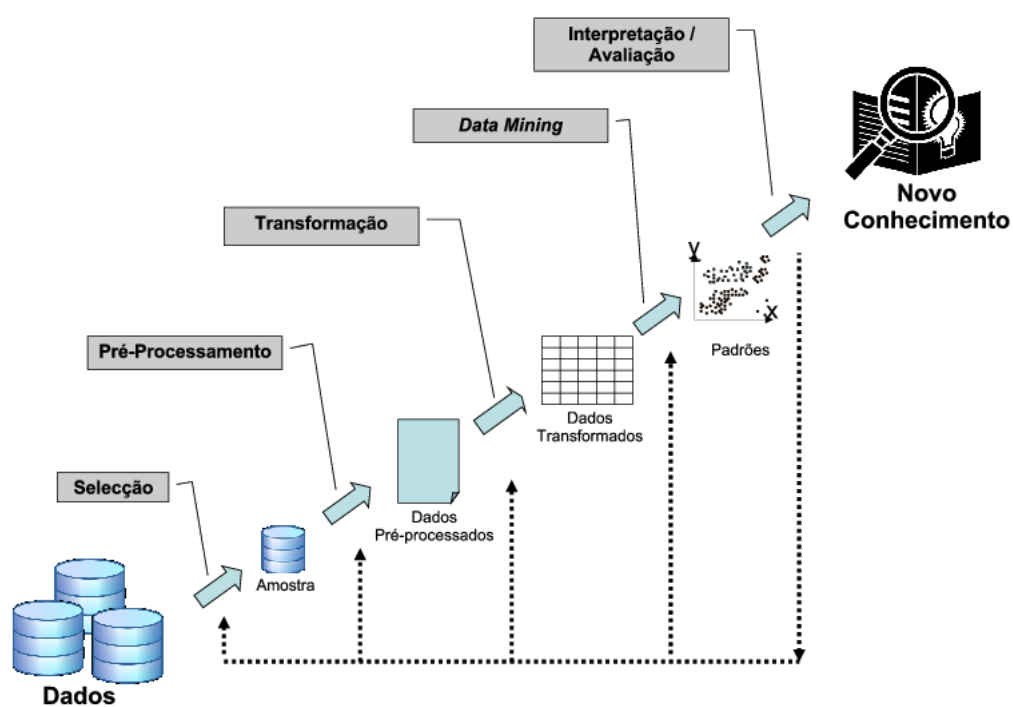


Figura 2.1: Processo de Extração de Conhecimento [FPSSU96]

Segundo Fayyad [FPSSU96], *Data Mining* especificamente é o núcleo principal do processo mais amplo denominado de KDD. O *Data Mining* utiliza técnicas de várias outras disciplinas conforme apresentado na Figura 2.2. As disciplinas mais influentes incluem a *Estatística*, as *Bases de Dados*, *Machine Learning*, as *Ciências da Informação*, a *Computação de Elevado Desempenho*, entre outras. Dependendo do objetivo estabelecido outras técnicas como a *Análise Espacial de Dados*, *Recuperação da Informação*, *Reconhecimento de Padrões*, *Análise de Imagens e Processamento de Sinais* podem ser usadas para extrair e representar a informação [Han01].

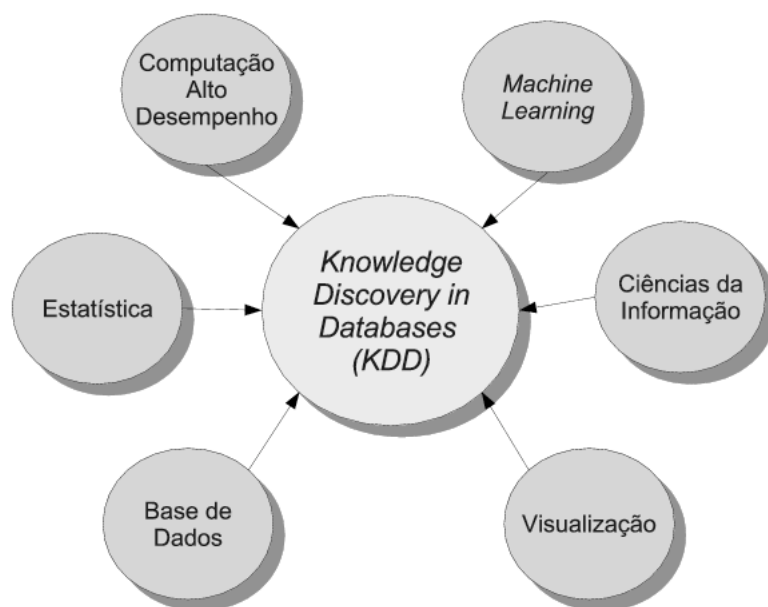


Figura 2.2: *Knowledge Discovery in Databases e as disciplinas envolvidas* [FPSSU96]

Ainda de acordo com *Fayyad* [FPSSU96] o processo de KDD, conforme ilustrado na Figura 2.1, é composto pelas fases:

- a) preparação de dados;
- b) seleção de dados;

- c) pré-processamento de dados;
- d) transformação de dados;
- e) *Data Mining*;
- f) interpretação e avaliação do conhecimento.

De uma forma mais elaborada e metódica, Han e Kamber [Han01] aglutinam algumas fases e propõem a seguinte organização:

- *pré-processamento*: reúne atividades que visam gerar uma representação conveniente para aplicação de algoritmos de *Data Mining*. Abrange etapas como seleção, limpeza, integração e transformação de dados;
- *Data Mining*: aplicação dos algoritmos de *Aprendizagem Computacional* ou *Estatística* para identificação de novos padrões de dados;
- *pós-processamento*: seleção e ordenação dos padrões de dados descobertos, mapeamentos de representação do conhecimento e geração de relatórios.

Na Figura 2.3 é mostrada a correspondência entre as diferentes abordagens de Fayyad [FPSSU96] e Han e Kamber [Han01], mas que consideram as mesmas etapas para o processo como um todo. É comum usar indistintamente KDD como sendo o passo de *Data Mining*. Nesta tese quando estivermos discutindo *Data Mining* estaremos referindo ao núcleo principal do processo KDD que trata da aplicação de algoritmos de Aprendizagem Computacional visando extrair novos conhecimentos [FPSSU96].

2.1.1.1 Fluxo do processo do KDD

O fluxo do processo de extração de conhecimento inclui as seguintes etapas [Han01]:

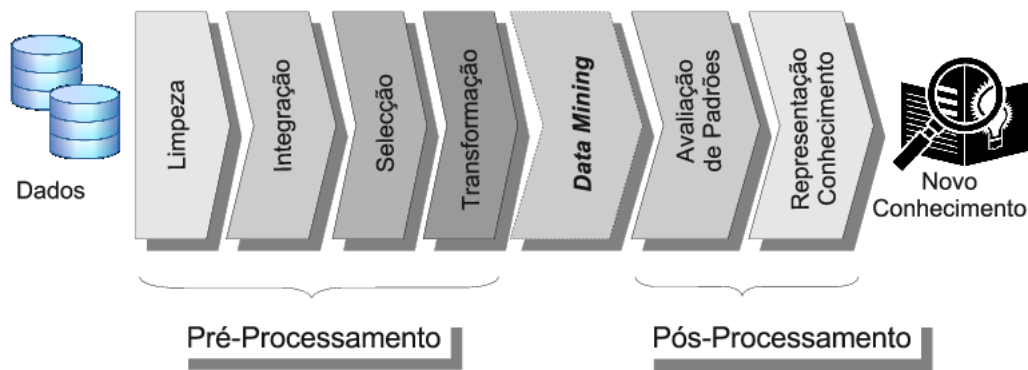


Figura 2.3: Etapas de pré-processamento, *Data Mining* e pós-processamento

1. definir claramente o domínio de conhecimento a ser tratado e os objetivos da aplicação de KDD;
2. definir um subconjunto de dados alvo na etapa de seleção de dados;
3. aplicar técnicas de limpeza e pré-processamento removendo inconsistências ou incoerências nos dados armazenados;
4. aplicar processos de *data reduction* e *data transformation* procurando identificar características úteis que possam uniformizar os dados de modo a facilitar a aplicação de alguns algoritmos;
5. escolher entre as tarefas ou funcionalidades de *Data Mining*: sumarização, classificação, regressão, associação ou *clustering*;
6. escolher o(s) algoritmo(s) de *Data Mining* apropriado(s) conforme a tarefa;

7. executar o *Data Mining* para extrair os padrões;
8. avaliar os padrões e representar o conhecimento, utilizando técnicas de visualização, transformação ou remoção de redundâncias;
9. interpretar informações ou conhecimento recém descobertos

Ainda de acordo com Han e Kamber [Han01] um ambiente de extração de conhecimento deve ter as seguintes características:

- capacidade para extrair diferentes tipos de conhecimento em Bases de Dados;
- requerer diferentes níveis de abstração para extrair conhecimento;
- possibilidade de incorporar conhecimento prévio;
- apresentar linguagens específicas para o processo *Extração de Conhecimento*;
- apresentar e visualizar os resultados obtidos;
- manipular dados incompletos, com interferências e inconsistências;
- avaliar padrões obtidos quanto ao grau de interesse;
- obter eficiência e escalabilidade na aplicação dos algoritmos;
- adotar métodos e técnicas de forma distribuída e paralela;
- gerir tipos de dados complexos;
- extrair informação de bases heterogêneas e da própria *Web* (*Web Mining*);
- analisar os impactos sociais devido as novas informações descobertas;
- proteger os dados quanto a segurança, integridade e privacidade;
- aplicar as diferentes tarefas de *Data Mining* (associação, classificação, agrupamento de dados, análise de tendências, sumarização)

2.1.2 Diferentes aspectos do processo de KDD

Há na prática dois objetivos gerais para justificar o uso de técnicas de extração de conhecimento: o primeiro trata da *predição* e permite prever determinados comportamentos futuros com base na análise de dados colectados previamente; o outro, a *descrição*, visa representar melhor um conjunto de dados de modo que ele possa apresentar informação ainda não estruturada, e até certo modo não planeada quando da criação da base de dados. Enquanto a *predição* permite observar padrões em dados colectados no passado com repercussões futuras quanto a tendências ou situações ainda não exploradas pelos utilizadores, a *descrição* procura apresentar os dados sob novos formatos e também em combinações inexploradas [FPSSU96].

2.1.2.1 Tarefas de *Data Mining*⁵

Segundo [Han01] e [Alp09], podemos considerar as seguintes tarefas para a etapa de *Data Mining*:

- *sumarização*: é a técnica que permite generalizar alguns conceitos sobre um determinado conjunto de dados. Os dados passam a ser tratados em faixas cujas as características ou propriedades são similares.
- *associação*: consiste em identificar um conjunto de características comuns entre os atributos sob análise, i.e., verificar as correlações entre os atributos e os respectivos valores. O mais comum é apresentar essas correlações na forma de regras de associação que representam as relações que decorrem entre atributos e a respectiva frequência num conjunto de dados. São representadas por meio de indução de regras do tipo $X \Rightarrow Y$:

⁵Os autores de [Han01] usam o termo *functionality*, sendo traduzido como *funcionalidades*. Em outras publicações usa-se o termo *task* (*tarefa*) para designar como sendo uma tarefa de *Data Mining* ou ainda a designação de uma *classe de problema* a ser tratado pelos algoritmos de *Data Mining*.

$A_1 \wedge \dots \wedge A_m \Rightarrow B_1 \wedge \dots \wedge B_n$ onde $A_i (i \in \{1, \dots, m\})$ e $B_j (j \in \{1, \dots, n\})$

- *classificação*: é o processo que consiste em modelar uma função capaz de caracterizar e distinguir classes de dados. A partir de um modelo pode-se classificar ou mesmo prever um conjunto de novos dados. Esta técnica requer um conjunto de dados na forma de exemplos pré-classificados (dados de treino) para que o algoritmo possa “caracterizar” determinados padrões e assim distinguir dentre diferentes classes. As técnicas ou algoritmos mais utilizados nesta funcionalidade são: Indução de Árvores de Decisão, Indução de Regras e Redes Neurais.
- *segmentação ou clustering*: neste tipo de problema não há necessidade de se conhecer a classe a que pertence o registo mas apenas os valores de cada atributo preditivo. A tarefa consiste em agrupar dados que possuem características similares.
- *detecção de desvios ou outliers*: um conjunto de dados pode conter alguns elementos que não têm comportamento uniforme ou homogêneo em relação aos demais. Esses elementos são considerados *outliers*. Dependendo da aplicação, devem apenas ser descartados pois podem indicar erros de medida ou coleta. Entretanto, em outros casos são considerados fortemente para análise, como é o caso de detecção de fraudes.
- *análise de tendências*: esta funcionalidade descreve o comportamento que os dados assumem ao longo do tempo. Pode ser aplicada em conjunto com as demais tarefas como segmentação, regressão ou classificação.
- *regressão linear*: consiste em identificar uma função linear do tipo $y = b + wx$ que possa representar os dados sob análise, onde x e y representam atributos numéricos.

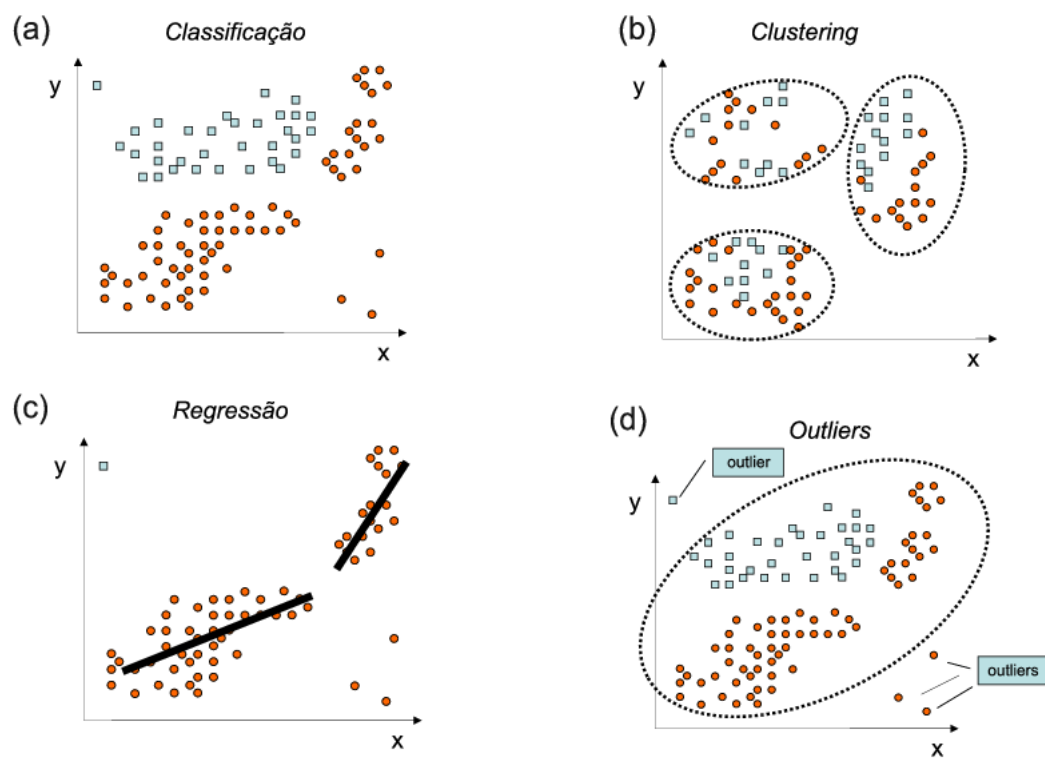


Figura 2.4: Tarefas de *Data Mining*: *classificação* (a), *clustering* (b), *regressão* (c) e *outlier detection* (d)

tipos de algoritmos X tarefas DM	Indução de Árvores de Decisão	Indução de Regras	Redes Neuronais	Algoritmos Genéticos
Sumarização	X	X		
Associação		X		
Classificação	X	X	X	X
Segmentação ou <i>Clustering</i>	X	X	X	X
Predição ou Previsão	X	X	X	X
Regressão	X	X	X	

Tabela 2.1: Tarefas de *Data Mining* e algumas técnicas utilizadas

Na Figura 2.4 são ilustradas algumas tarefas de *Data Mining* e na Tabela 2.1 apresentadas as tarefas e alguns tipos de algoritmos que estão associados à respectiva função de extração de conhecimento.

2.1.3 Medidas de Interesse

O processo de extração de conhecimento pode gerar inúmeros padrões alternativos sobre uma base de dados. A questão que se coloca é então quais dos padrões podem ser realmente úteis e poderão estar contribuindo para o processo de descoberta de novas informações. Para que um padrão descoberto tenha alguma utilidade é importante que ele faça algum sentido ao analista, seja válido sobre o domínio de conhecimento, tenha um considerável grau de confiança e permita validar alguma hipótese que se deseje confirmar. Quando um padrão permite a validação de uma nova hipótese consideramos que este padrão passa a representar novo conhecimento [FPSSU96, Han01].

As medidas de interesse permitem considerar alguns padrões como interessantes e válidos para as hipóteses que se pretende validar. Estas medidas dependem do tipo

de tarefa a realizar. Por exemplo, numa tarefa de Indução de Regras de Associação são importantes as medidas de Suporte e Confiança [Han01]:

Suporte: numa regra do tipo $X \Rightarrow Y$, o suporte é definido como sendo o n^o de registos contendo X e Y simultaneamente dividido pelo o n^o total de registos. Indica portanto a percentagem de transações na base de dados que satisfazem a regra:

$$Suporte(X \Rightarrow Y) = (X \cup Y)$$

Confiança: numa regra do tipo $X \Rightarrow Y$, a confiança é definido como sendo o n^o de registos contendo X e Y dividido pelo o n^o total de registos contendo apenas o elemento X . Indica o grau de acerto da regra.

$$Confiança(X \Rightarrow Y) = (X/Y)$$

Em tarefas de classificação: *Error Rate* e *Accuracy*.

Error rate: Nas tarefas de classificação é comum medir-se o desempenho dos classificadores através da taxa de erro (*error rate*). A taxa de erro é a proporção dos erros encontrados entre todas as instâncias e indica o desempenho global do classificador. Grande parte dos algoritmos de classificação assume que todos os erros têm o mesmo custo e são, normalmente, concebidos para minimizar o número de erros (perda 0/1). Nestes casos, a taxa de erro é equivalente à atribuição do mesmo custo para todos os erros de classificação, assim, por exemplo, no caso de uma classificação binária, os falsos positivos teriam o mesmo custo que os falsos negativos.

Accuracy: É comum, também, medir-se o desempenho dos classificadores por meio da taxa de acertos (*accuracy*). A taxa de acertos é a proporção dos acertos encontrados entre todas as instâncias analisadas e indica portanto a proporção dos exemplos corretamente classificados.

2.1.4 Algoritmos para as tarefas de DM

Nem todos os padrões possíveis podem ser extraídos já que estes dependem diretamente do método ou algoritmo usado. Alguns algoritmos podem ser mais completos e complexos que outros na busca de um padrão, o que requer do analista a seleção mais adequada do algoritmo em função dos dados a serem analisados, considerando a tarefa *Data Mining* e os resultados esperados [FPSSU96].

O projeto MetaL⁶ desenvolveu um sistema conhecido como *Data Mining Advisor* que consiste em sugerir um conjunto de algoritmos que podem ser utilizados conforme o conjunto de dados apresentado pelo utilizador. Os algoritmos são ordenados segundo seu potencial para solução de um novo problema conforme as características dos dados apresentados. Deste modo o utilizador interessado em aplicar DM a uma base de dados pode ter um ponto de partida para iniciar o trabalho evitando o risco de aplicar determinada técnica que seja incompatível com os dados que dispõe [BPK00, BS00].

2.1.4.1 Regras de Associação

As Regras de Associação (RA) são métodos que permitem identificar e representar por intermédio de regras “SE ... ENTÃO” correlações “significativas” entre diferentes itens (atributos) de um conjunto de dados. Ao contrário das regras de classificação não existe um atributo especial chamado “classe”. As regras possuem as seguintes propriedades:

- Se o resultado de uma operação booleana no antecedente causa (SE) for verdadeiro, muito provavelmente o consequente (ENTÃO) também será;
- São bastante compreensíveis e úteis para a tomada de decisões.

⁶URL do projeto MetaL: <http://www.metal-kdd.org>

O método *market basket*, por exemplo, considera que, dado um conjunto de transações de vendas, se pode encontrar uma regra que mostre o relacionamento entre os itens de venda. Assim, se um determinado *produto A* for vendido há, com certo grau de certeza, a possibilidade de se encontrar também um *produto B* na mesma compra, conforme demonstrado formalmente pela regra:

$$A \Rightarrow B$$

$$[\text{Suporte} = 2\%, \text{Confiância} = 60\%]$$

$$\text{Compra}(\text{Cliente}, \text{Produto_A}) \Rightarrow \text{Compra}(\text{Cliente}, \text{Produto_B})$$

O processo de descoberta de Regras de Associação é implementado em dois passos:

- (a) procurar os itens mais frequentes;
- (b) gerar regras de associação entre os itens mais frequentes, respeitando os valores das medidas de interesse especificadas pelos utilizadores para o suporte e a confiança⁷.

As regras de associação podem ser usadas em análises de mercado para estabelecer novas estratégias de venda. Diferentes algoritmos podem ser usados no processo de geração deste tipo de regras. Um dos mais utilizados é o algoritmo *Apriori* [AS94], que estabelece que se um conjunto de itens é frequente, esse mesmo conjunto de itens estará presente em regras de associação. É possível usar este algoritmo para análise particionada de uma extensa base de dados, conforme passos demonstrados na Figura 2.5 [Han01].

2.1.4.2 Classificação

O processo de classificação inclui geralmente 3 etapas: construção do classificador (aprendizagem); avaliação do classificador (verificação) e; uso do classificador (classificação). Estas etapas estão ilustradas na Figura 2.6 [Han01].

⁷Definidos na Secção 2.1.3

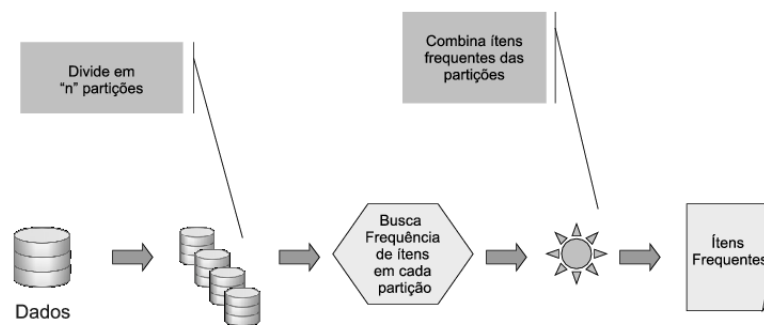


Figura 2.5: Etapas da aplicação do algoritmo Apriori conforme [AS94]

Durante a aprendizagem (a), o “algoritmo de classificação” utiliza uma amostra de dados para treino que lhe permite a geração de um classificador (um classificador pode ser visto como um modelo para os dados).

Na avaliação (b), o modelo encontrado é aplicado sobre uma amostra de dados diferente da de treino (conjunto de teste) e verificação nas medidas de interesse. Caso a avaliação não seja satisfatória um novo classificador deve ser construído voltando ao ponto (a).

Na última etapa classificação (c), a partir de um modelo gerado e avaliado, o classificador é usado para classificar dados reais.

2.1.5 Extração de Conhecimento - implementação

Em alguns casos, a aplicação de técnicas de *DM* deve ser revista para considerar principalmente os dados dispersos na Internet, por exemplo. A extensão e a alta dimensionalidade de algumas BD - muitos atributos versus muito valores, requerem

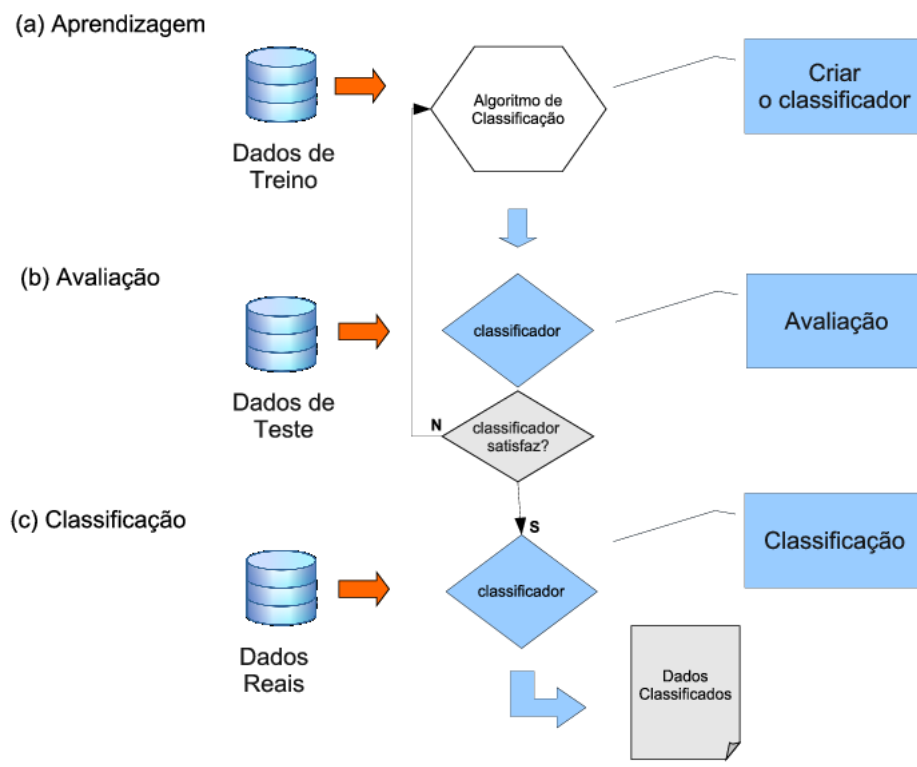


Figura 2.6: Etapas de um algoritmo de Classificação, adaptado de [Han01]

cada vez mais demanda por computação paralela e distribuída. A própria rede de comunicação que interliga diversos ambientes computacionais já é desenvolvida o suficiente para fazer parte de uma arquitetura de computação que permita a implementação de estrutura de *Data Mining Distribuído* [KC00].

Desenvolver técnicas de *DM* para arquiteturas distribuídas tem sido um grande desafio atual da comunidade científica de *Knowledge Discovery in Databases*. O desenvolvimento de um novo estágio do KDD denominado de *Distributed and Parallel Knowledge Discovery* (DPKDD) está ligado principalmente pelo avanço em tecnologias de armazenamento de dados e redes de telecomunicações [KC00] e uso de novas abordagens em computação distribuída como *Grid Computing* [FK99].

Além disso, dependendo do domínio de conhecimento a ser tratado por técnicas de *DM*, uma simples amostra pode não ser suficiente para se obter padrões a serem

analisados. Já que alguns comportamentos ditos anómalos podem surgir somente após grandes quantidades de dados serem analisados. Mesmo para algoritmos de *Aprendizagem Computacional*, a necessidade de registrar um padrão somente terá efeito se aumentarmos o tamanho da amostra de dados de aprendizado [KC00]. Isso conduz a necessidade de analisarmos quantidades significativas de dados e por conseguinte uma maior necessidade em termos computacionais.

Para que um processo de KDD possa beneficiar-se do modelo de computação distribuída, devemos adotar o modelo apresentado na Figura 2.7, que prevê um modelo de *Data Mining Distribuído*, onde uma determinada base de dados é fracionada em sub-bases. A cada fração da base principal podemos aplicar algoritmos de *Aprendizagem Computacional* garantindo a independência de processamento. Deste modo, um ou mais nós de um ambiente de computação distribuída podem executar processos de forma independente, cabendo a um nó central a tarefa de consolidação dos resultados e apresentação ao utilizador.

2.1.6 Questões de Pré-processamento em Extração de Conhecimento

Os dados oriundos de aplicações reais apresentam muitas vezes inconsistências porque simplesmente foram armazenados de forma incorreta ou incompleta [Han01].

De um modo geral, e conforme [Han01], os dados podem ser : (a) incompletos, sendo mais frequente o não preenchimento de alguns valores para determinados atributos; (b) apresentar interferências ou ruído, quando um determinado valor inserido está sob diferentes designações ou simplificações (abreviaturas), podendo gerar *outliers*; e (c) apresentar discrepâncias quanto a valores que deveriam estar presentes e entretanto foram substituídos por outros. Além disso, alguns atributos que pode-

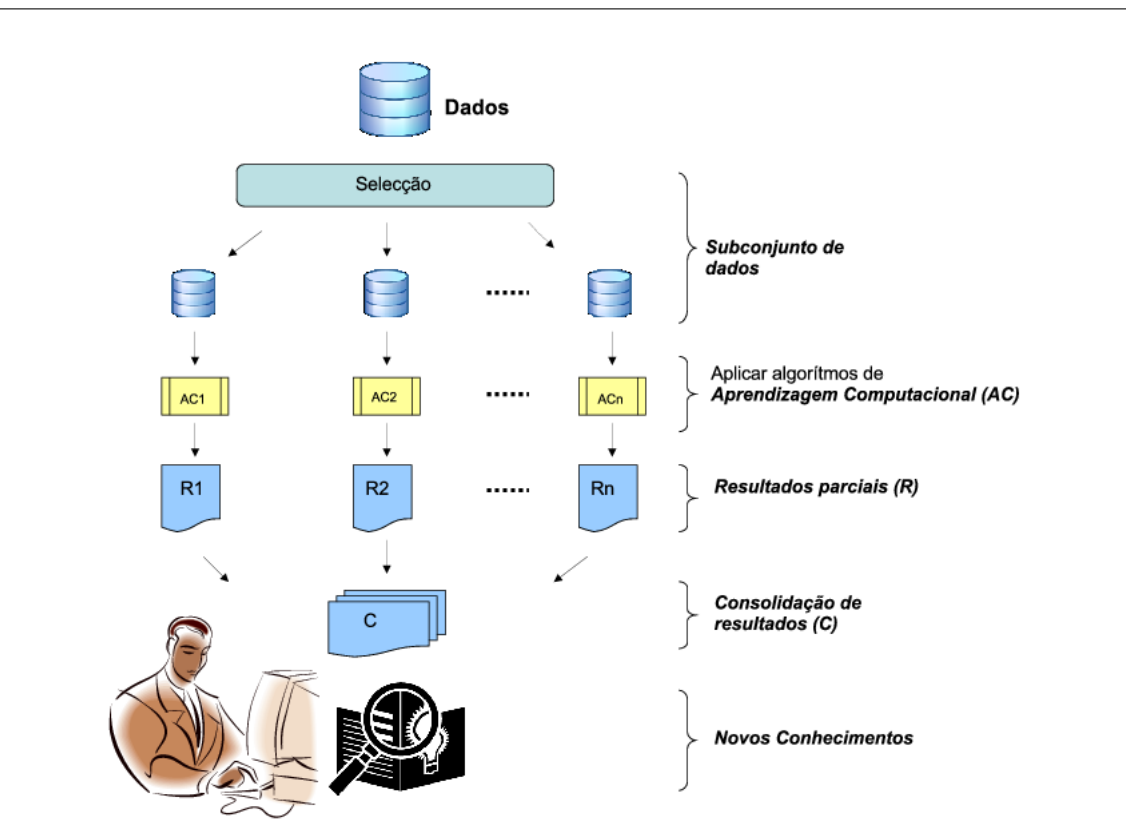


Figura 2.7: *Data Mining* Distribuído. Divisão dos dados inicialmente centralizados.

riam interessar ao processo de extração de conhecimento não foram planeados ou inseridos na Base de Dados para colecta e armazenamento.

Não havendo qualidade nos dados, a etapa de *Data Mining* fica dificultada e dificilmente teremos qualidade no processo de extração de conhecimento e por conseguinte a tomada de decisão será afetada. A adoção de *Data Warehouses (DW)* pode eliminar alguns dos problemas encontrados quanto à qualidade dos dados, mas nem sempre há a possibilidade de se ter um DW para aplicar algoritmos de *Data Mining* [FPSSU96, Han01].

Algumas técnicas que podem ser usadas para minimizar problemas com os dados reais, conforme proposto em [Han01] e que fazem parte do processo KDD, estão ilustradas na Figura 2.8. Entre elas podemos destacar:

- Limpeza dos Dados (*Data Cleaning*): consiste na aplicação de técnicas de limpeza tornando os dados uniformes e estruturados, detectando ausências de valores ou retirando eventuais discrepâncias. Atributos que não foram preenchidos ou apresentem dados inconsistentes podem ter valores substituídos por algo representativo que durante a etapa de *DM* possa ser compreendido e interpretado. Esses novos valores inseridos podem levar em consideração os demais dados, podem representar uma média ou o valor mais provável para aquele atributo. O fundamental é que nenhuma base de dados deve ser analisada por um algoritmo de *DM* sem antes ter sido devidamente uniformizada. Caso contrário, isso poderá distorcer os resultados esperados, ou ainda comprometer a eficácia e a eficiência do próprio algoritmo.
- Integração de dados (*Data Integration*): alguns conjuntos de dados podem ser originários de diferentes fontes (DBMS⁸, *ficheiro texto*, Web, entre outros) sendo necessário consolidar numa única base para a aplicação do processo de *DM*. Esta etapa permite ainda reconhecer eventuais redundâncias em termos de atributos e eliminar inconsistências com a manipulação de diferentes bases. A simples implementação de *Data Warehouse* é um exemplo típico de integração de dados de diferentes fontes e de diferentes formatos.
- Transformação de dados (*Data Transformation*): consiste, basicamente, na normalização dos dados por meio de agregação, generalização ou mesmo criação de novos atributos garantindo eficiência e precisão quando da aplicação dos algoritmos. O simples fato de se combinar dados provenientes de diferentes fontes pode requerer a simplificação de escalas, normalização de valores, fusão de atributos, consolidação de valores médios e ainda normalização para a escala decimal.
- Redução de dados (*Data Reduction*): extensas bases de dados com inúmer-

⁸*DataBase Management System*

ros atributos podem prejudicar ou mesmo inviabilizar a aplicação de alguns algoritmos de *Data Mining*. Pode-se reduzir significativamente o tamanho de uma base eliminando redundâncias, agregando determinados atributos ou agrupando determinados valores em intervalos. Aplicando técnicas de segmentação pode-se extrair uma amostra representativa de uma extensa base de dados e assim aplicar as técnicas da etapa de *Data Mining*.

A aplicação dessas técnicas pode contribuir significativamente para aumentar a qualidade dos padrões encontrados e reduzir sensivelmente o tempo de processamento [FPSSU96, Han01].

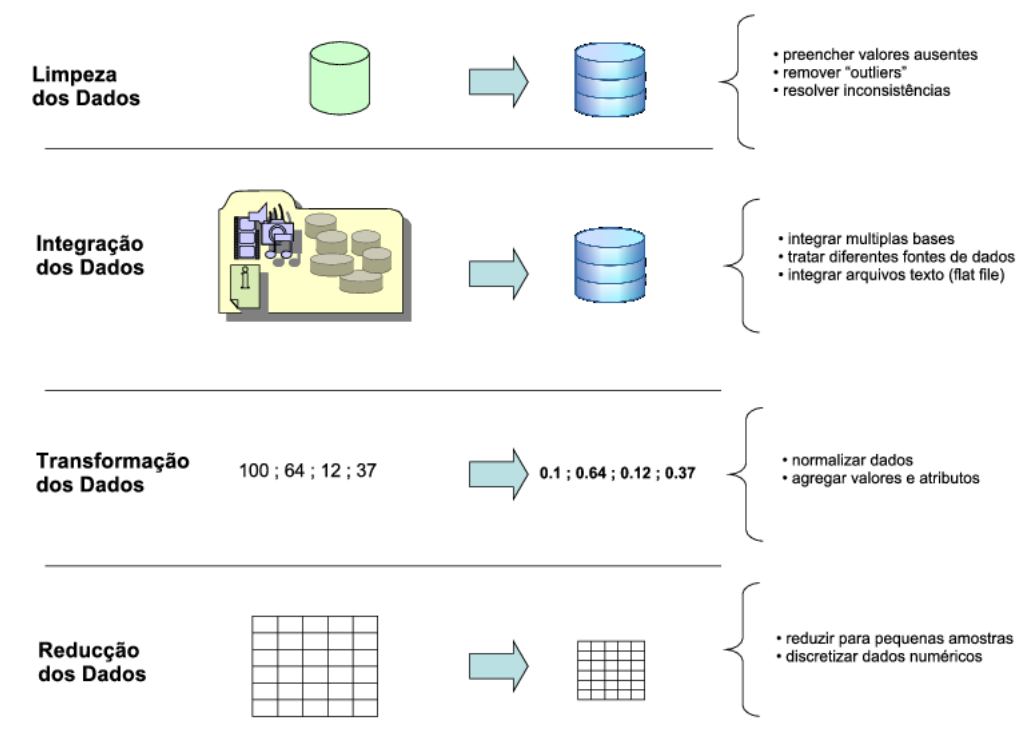


Figura 2.8: Pré-processamento dos dados

2.2 Extração de Conhecimento - métodos e ferramentas

O mercado dispõe atualmente de um conjunto de ferramentas direcionadas ao processo de Extração de Conhecimento completo ou ao *DM* em particular. A seguir apresentamos um resumo das principais características de algumas das opções disponíveis no mercado, com destaque para os algoritmos implementados e algumas das restrições que cada solução apresenta. Alguns dos produtos são licenciados sem qualquer custo, na forma de *GNU - General Public License*⁹, conhecida como Licença GNU-GPL, outros porém requerem o pagamento da licença de uso. É possível ter todo um ambiente de extração de conhecimento sem dispendios financeiros.

Cabe salientar que algumas tarefas e algoritmos têm nomes diferentes dependendo do fornecedor, dessa forma de modo a não prejudicar o entendimento optamos por apenas mencionar a tarefa de extração de conhecimento que é implementada sem nos preocuparmos com o rigor do detalhe sobre o algoritmo implementado.

Antes no entanto de discorrer sobre as diferentes ferramentas e implementações para uso em projetos de KDD, apresentamos a metodologia conhecida como *Cross Industry Standard Process for Data Mining*¹⁰ (CRISP-DM¹¹) [CCK⁺00] que por vezes é seguida como modelo de referência para alguns projetos e mesmo ferramentas de KDD.

⁹Licença de Software Livre promovida pela *Free Software Foundation* (FSF). A fundação FSF, criada em 1985, dedica-se a promover a liberdade dos utilizadores em geral quanto ao uso, cópia, modificação e re-distribuição de programas de computador sem pagamento de licenças comerciais. Mais informações na URL:<http://www.gnu.org/licenses/>

¹⁰O termo *Data Mining* usado na metodologia refere-se ao processo completo de KDD

¹¹URL:<http://www.crisp-dm.org>

2.2.1 A metodologia CRISP-DM

A metodologia CRISP-DM descreve um modelo de referência a ser utilizado em projectos de KDD. Foi inicialmente proposta 1996 por um consórcio formado pela DaimlerChrysler, SPSS e a NCR, e teve sua versão 1.0 tornada disponível em 1999. A metodologia CRISP-DM é composta por seis fases cada uma das quais contém um conjunto de tarefas específicas e com resultados esperados bem definidos. As fases que compõem a metodologia são:

- Definição dos objetivos de negócio
- Compreensão dos dados
- Preparação dos dados
- Modelização ou Modelagem
- Avaliação dos resultados
- Implementação ou desenvolvimento

Na Figura 2.9 são ilustradas as várias fases e as respectivas interações entre elas. É mostrado na figura o ciclo de vida de um projeto de KDD de acordo com a metodologia. Apresentamos de seguida uma breve descrição das fases:

Definição dos objetivos de negócio Consiste na identificação dos objetivos de negócio que o projeto de KDD deverá satisfazer. Estes objetivos de negócio, uma vez identificados, passarão a ser os objetivos do próprio projeto de KDD segundo a metodologia. Como tarefas principais desta fase temos: identificação dos objetivos de negócio, descrição do contexto (requisitos, recursos, riscos e custos/benefícios), identificação dos objetivos de KDD e produzir um plano do projeto.

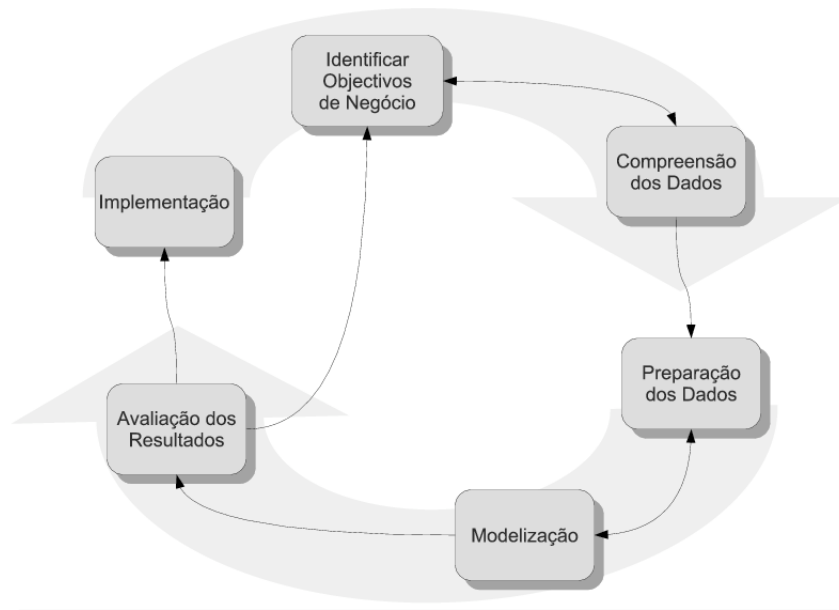


Figura 2.9: Metodologia CRISP-DM conforme [CCK⁺00]

Compreensão dos dados Esta fase caracteriza-se por uma análise aprimorada quanto à qualidade dos dados, identificando eventuais problemas que possam prejudicar os resultados do processo de KDD. Esta fase tem como principais tarefas: identificar os dados, descrever os dados, explorar e verificar a qualidade dos dados.

Preparação dos dados Nesta fase, os dados são devidamente identificados e tratados para posteriormente serem submetidos aos algoritmos de *DM*. Como tarefas principais destacamos: selecionar, limpar eventuais inconsistências ou valores ausentes, integrar e formatar os dados conforme os requisitos dos algoritmos a serem aplicados.

Modelização Consiste em aplicar os algoritmos de análise de dados e observar os resultados obtidos, e, eventualmente, efetuar ajustes, seja retornando a uma das etapas anteriores ou apenas ajustando eventuais parâmetros dos algoritmos ou ferramentas de *Data Mining* em uso. As principais tarefas são: selecionar técnicas de modelização, definir testes e aferições, construir e analisar modelos.

Avaliação dos resultados Esta fase permite aferir se os modelos obtidos atendem aos objetivos inicialmente propostos para o projeto de KDD em curso. As principais tarefas são: avaliar os resultados, rever o processo e identificar os próximos passos a partir dos resultados obtidos.

Implementação ou desenvolvimento Esta última etapa da metodologia prevê que os modelos e resultados encontrados sejam adaptados à organização, quer seja produzindo novas regras de negócio ou subsidiando novas estratégias de mercado. É necessário, portanto, estabelecer quais ações devem ser tomadas para que os resultados obtidos nas fases anteriores sejam aplicados à organização e seus respectivos sistemas. Como principais tarefas destacamos: planejar e monitorizar a aplicação dos resultados obtidos, produzir relatórios sobre os modelos e resultados obtidos e rever o projeto para incorporar novos dados e mesmo novas análises. Por se tratar de um ciclo de vida é previsível que retornemos as fases iniciais cada vez que os objetivos sejam aprimorados e novos modelos sejam encontrados.

2.2.2 Implementações em SGBD

Nesta seção, apresentamos alguns exemplos de produtos disponíveis no mercado que incorporam no Sistema de Gestão de Bases de Dados (SGBD) métodos e ferramentas comumente usadas no processo de Extração de Conhecimento. Na Tabela 2.2 apresentamos um quadro comparativo de alguns SGBD com as funcionalidades e algoritmos implementados.

A implementação dos algoritmos de Extração de Conhecimento no próprio SGBD traz algumas vantagens, segundo os próprios fornecedores. Entre as vantagens principais destacam-se:

- menor mobilidade dos dados: alguns dos algoritmos de extração de conhecimento estão implementados em software específico e dedicado (ferramentas de

Tarefa	Data Mining Oracle	SQL-Server Microsoft	Data Miner IBM DB2
Sumarização	X		X
Associação	X	X	X
Classificação	X	X	X
<i>Clustering</i>	X	X	X
Detecção de <i>Outliers</i>		X	
Regressão	X	X	X

Tabela 2.2: Classificação das funcionalidades de Extração de Conhecimento disponíveis em alguns SGBD

análise de dados), isso pode requerer constantes conversões ou transferências de dados de um lado para outro para satisfazer este requisito. Com a implementação dos algoritmos no próprio SGBD a análise é feita diretamente nas tabelas de dados;

- maior segurança dos dados: o facto dos dados estarem no SGBD facilita que determinados atributos mais sensíveis à privacidade possam ser protegidos de uso indevido ou mesmo de uma possível personalização do resultado obtido. Esta característica pode viabilizar, por exemplo, que uma base de clientes possa ser analisada omitindo-se a identificação dos nomes ou de outros atributos que possam identificar o próprio cliente;
- consistência e atualização: esta característica garante que a análise é processada sempre usando os dados mais atuais.

Oracle Data Mining As principais funções de extração de conhecimento implementadas neste SGBD são [TKH⁺05]:

- Data Mining Supervisionado:

-
- Classificação: Indução de Árvores de Decisão, Naive Bayes;
 - Regressão;
 - Data Mining Não Supervisionado:
 - Segmentação ou agrupamento (*Clustering*): *k-means*
 - Regras de Associação: *market basket*

Microsoft SQL-Server

- Data Mining Supervisionado:
 - Classificação: Indução de Árvores de Decisão, Naive Bayes;
 - Regressão;
 - Detecção de anomalias (*outliers*): OCSVM (One-Class Support Vector Machine);
- Data Mining Não Supervisionado:
 - Segmentação ou agrupamento (*Clustering*): *k-means*
 - Associação: *market basket*

IBM DB2 Intelligent Miner

- Data Mining Supervisionado:
 - Classificação: Indução de Árvores de decisão, Redes Neurais Artificiais;
 - Regressão: Redes Neurais Artificiais, Regressão Linear, predição RBF (Radial Basis Functions)
- Data Mining Não Supervisionado:

- Segmentação (*Clustering*): *distributed-based clustering*, *center based clustering*
- Indução de Regras de Associação;

2.3 Aprendizagem Computacional

Alan Turing em uma conferência realizada em Manchester (1951) intitulada “Intelligent Machinery, A Heretical Theory” reproduzida em [FMM99], ainda nos primórdios da Ciência da Computação, cita e descreve o termo “learn by experience” e teoriza todo um processo que sugere que a máquina pode ser “educada” e como consequência pode aprender, inclusive pode cometer erros e assim ser aperfeiçoada com a experiência. Muito provavelmente naquela época estava lançado o desafio à comunidade científica de criar uma máquina capaz de “pensar”, ou seja, “aprender com a sua própria experiência”.

Atualmente a área de *Machine Learning (ML)* ou *Aprendizagem Computacional (AC)* tem-se concentrado na essência já conceituada por Turing, ou seja, aperfeiçoando o conceito e os métodos (algoritmos) usados para que o computador possa “aprender” e assim dar respostas a determinados problemas sem necessidade de uma estruturação adequada ou mesmo completa. A partir de um conjunto de dados sobre um determinado domínio de conhecimento é possível criar um programa ou sistema capaz de “aprender” criando modelos a partir da identificação de padrões e regularidades dos dados. Isso faz com que em áreas em que o conhecimento ainda não está estruturado por completo se possa criar um modelo que permita simular determinados comportamentos e assim facilitar o entendimento do processo como um todo. Se nós fazemos algo inconscientemente e não conseguimos explicar como fazemos, certamente não conseguiremos escrever um programa para simular a mesma tarefa. É neste aspecto em que a *Aprendizagem Computacional* pode ajudar na criação de sistemas ou programas [Alp09].

Recentemente, Tom Mitchell [T.M97] apresenta a seguinte definição para aprendizagem automática:

Considera-se que um programa de computador aprendeu a partir de uma experiência (E) em relação a um conjunto de tarefas (T) segundo um desempenho (P), se o desempenho na execução das tarefas melhorar com a experiência.

Como exemplo:

T: jogar xadrez

P: percentagem de jogos ganhos aos oponentes

E: prática de jogadas passadas e regras do jogo

Embora diversas técnicas de AC possam ser usadas em DM em tarefas de classificação apresentamos agora um estudo sumário de algumas técnicas de AC usadas na tese.

2.3.1 Indução de Árvores de Decisão

Árvores de Decisão são estruturas que permitem a representação de um conjunto de decisões de forma simples. Os algoritmos relacionados à indução de árvores de decisão permitem gerar regras de classificação dos dados [Han05].

Na Figura 2.10, ilustramos os vários elementos que compõem uma estrutura de Árvore de Decisão: *nós*, *raiz*, *ramos* e *folhas*. O *nó de decisão* representa um teste num atributo qualquer; cada *ramo* ou *caminho* corresponde a um possível valor deste atributo e cada *folha* está associada a uma classe. O percurso a partir da *raiz* até as folhas permite interpretar uma *regra de classificação*, ou seja, cada folha da árvore dá origem a uma regra, sendo a sua conclusão registada na parte consequente da regra. No caso da Figura 2.10, como exemplo, podemos extrair a seguinte regra genérica: *Se* N1 = “c2” *e* N2 = “c5” *Então* “f3” .

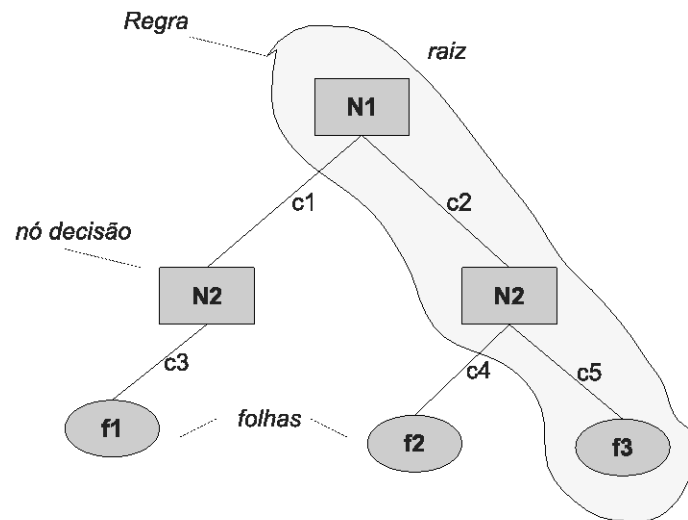


Figura 2.10: Representação de Árvore de Decisão

2.3.2 Indução de Programas em Lógica

A Indução de Programas em Lógica (IPL)¹² é uma área de investigação resultante da intersecção da Programação em Lógica (PL), da Aprendizagem Computacional (AC) e da Estatística. Por ter a base em PL, ILP utiliza a lógica de predicados para descrever tantos os dados como os modelos que constrói.

DM consiste em descobrir regularidades ou padrões nos dados através de uma linguagem formal de descrição desses mesmos padrões. A seleção da linguagem e do método para representar padrões de dados é um dos maiores desafios. Historicamente, métodos baseados em linguagens de descrição atributo-valor (*attribute-value language* - AVL) são as mais populares. Indução de Regras, Redes Neurais Artificiais e Árvores de Decisão são típicos exemplos de métodos baseados em AVL. São relativamente simples, eficientes e podem manipular dados com ruído. Estes métodos têm sérias limitações para representar determinados conhecimentos. O

¹²Tradução de *Inductive Logic Programming (ILP)*. Nesta tese usaremos sempre o termo *ILP* para referir esta área de investigação.

ILP permite superar estas limitações incorporando conhecimento prévio (*background knowledge*) sobre o domínio do problema e quando o conhecimento está guardado numa BD. Os métodos AVL obrigam a uma transformação de informação guardada em várias tabelas numa só, com consequente perda de informação. O ILP pode usar “diretamente” as várias tabelas sem transformação.

Uma regra do tipo *if-then* é representada em predicados de 1ª ordem e são formalmente representados por cláusulas *Horn*. Deste modo considerando que:

B: Conhecimento prévio (*background knowledge*) representando por um conjunto de cláusulas Horn;

P: conjunto de cláusulas Horn representando exemplos positivos;

N: conjunto de cláusulas Horn representando exemplos negativos;

Podemos ter uma hipótese (H) representada sob a forma de cláusulas Horn em que temos:

$\forall p \in P : H \cup B \models p$ (completude)

$\forall n \in N : H \cup B \not\models n$ (consistência)

Sistemas ILP permitem armazenar e processar conhecimentos ou informações prévias sobre um domínio de conhecimento na forma de programas lógicos. Um programa lógico é constituído de cláusulas. Uma cláusula consiste basicamente de regras de 1ª ordem, onde a conclusão é chamada de “cabeça” da regra. Um exemplo:

$pai(X, Y) \vee mae(X, Y) \leftarrow pais(X, Y)$

Se X é um dos *pais* de Y então X é *pai* de Y ou X é *mae* de Y , onde o símbolo $\vee$ indica a operação lógica “ou”.

Vantagens do ILP:

- elevado poder expressivo;

- permite fornecer conhecimento do domínio;
- fácil compreensibilidade dos modelos;
- aceita múltiplas relações (tabelas de dados de BD).

Entretanto, ILP não permite tratar grandes quantidades de dados; é limitado para manipular e processar dados numéricos, e é principalmente usado para tarefas de classificação em *DM*.

Um dos tipos de sistemas de ILP mais populares é chamado MDIE (Mode Directed Inverse Entailment) da autoria de Stephen Muggleton. São exemplos desta categoria de algoritmos os sistemas Progol [Mug95], Aleph [Sri03], April [FSC06] e IndLog [Cam00,Cam04]. Estes sistemas implementam um algoritmo ganancioso¹³ de cobertura¹⁴. Têm geralmente um ciclo principal em que, em cada iteração constroem a cláusula com maior cobertura, juntam essa cláusula ao modelo (conjunto de hipóteses/cláusulas) e removem os exemplos positivos cobertos pela cláusula construída. Para construir a cláusula com melhor cobertura efetuam dois passos designados por: saturação e redução. Na saturação é construída a cláusula mais específica do espaço de hipóteses que vai ser pesquisado. Esta cláusula mais específica é o limite inferior do espaço de procura e serve também de fonte para os literais que constituem as cláusulas construídas no passo de redução. No passo de redução é feita uma procura desde a cláusula mais geral em direção à cláusula mais específica. Isso é feito começando na cláusula mais geral e especializando-a. Aplicando este refinamento aos descendentes da cláusula mais geral e depois recursivamente aos descendentes dos descendentes o algoritmo pode percorrer o espaço das hipóteses do ponto mais geral para o mais específico. Sempre que é gerada uma cláusula a sua cobertura é avaliada, isto é, calcula-se quantos exemplos positivos e negativos a cláusula permite derivar. Os exemplos deriváveis por cada cláusula são guardados na sua lista

¹³Greedy algorithm.

¹⁴Uma hipótese cobre um exemplo quando, juntamente com o *background knowledge*, consegue deduzir (explicar) esse exemplo.

de cobertura (lista para os positivos e lista para os negativos). Uma cláusula pode ser aceite (pode ser uma possível cláusula resultado) se for consistente, isto é, se não cobrir nenhum exemplo negativo. Como o espaço de hipóteses é geralmente muito grande, os sistemas de ILP usam declarações de determinação para indicar que símbolos de predicado podem ser usados para construir as cláusulas e declarações de modo para indicar tipos para os argumentos dos predicados. A existência de tipos evita combinações inúteis de predicados. Além dos tipos, as declarações de modo indicam, para cada argumento, se este é de entrada, saída ou se vai ter uma constante. Os sistemas de ILP usam também parâmetros como o *clauselength* que indica um limite para o número de literais numa cláusula, o *nodes* que indica um limite para o número de cláusulas construídas e o *language* que indica um limite para o número de repetições de símbolos de predicado numa cláusula.

Representar a informação em ILP - ILP tem sido uma área que permite que análises sejam feitas considerando os dados nas suas tabelas de origem sem a necessidade de junção dos dados ou supressão de atributos.

A forma de representar os dados de uma base de dados relacional num sistema ILP é bastante simples, bastando identificar a tabela como sendo o predicado da cláusula e os atributos as variáveis, conforme ilustrado na Figura 2.11, e demonstrado a seguir:

$\langle \text{Nome_da_Tabela} \rangle (\langle v_1 \rangle \dots \langle v_n \rangle)$, onde $\langle \text{Nome_da_Tabela} \rangle$ é chamado de predicado e $\langle v_1 \rangle \dots \langle v_n \rangle$ as variáveis que correspondem as colunas da tabela.

Para representar uma informação disponível numa tabela que tem atributos incompletos ou que requeiram uma busca para completar usamos a notação:

Tabela 1 - Clientes

Id_Cliente	Nome	Idade	Sexo	Renda
11111	Keke	36	F	10000
22222	Domi	28	F	8000
33333	Gui	25	M	7000
44444	Ramos	41	M	10000



Clientes (Id_Cliente, Nome, Idade, Sexo, Renda).



Clientes (44444, Ramos, 41, M, 10000).

Figura 2.11: Representação de dados relacionais em cláusula de 1ª ordem em sintaxe Prolog

Clientes (-,-,-,F,-). onde o símbolo “-” representa que não há informação disponível.

Exemplos de representação de uma relação:

$Clientes(C, -, -, F, -) \wedge Compras(C, -, -, -, Dinheiro) \rightarrow BomConsumidor(C)$

ou em a sintaxe *Prolog*:

$BomConsumidor(C) : \neg Clientes(C, -, -, F, -), Compras(C, -, -, -, Dinheiro).$

ILP é um método que se aplica a tarefa de classificação, onde o *background knowledge* (B) é expresso como sendo:

- um conjunto de definição de predicados
- um conjunto de exemplos positivos (E+)

-
- um conjunto de exemplos negativos (E-)

Uma aplicação ILP segue 3 passos:

1. definir um conjunto eficaz e representativo de exemplos (positivos e negativos);
2. definir um “background knowledge” relevante;
3. utilizar um sistema ILP genérico para processar o “background knowledge” e os exemplos positivos e negativos de modo a elaborar uma teoria (hipótese). Esta teoria ou hipótese é apresentada na forma de regras que permitem validar o “background knowledge” usando os exemplos dados (positivos e negativos).

Na Figura 2.12 ilustramos as fases da aplicação do método, as quais descrevemos como:

- todos os exemplos positivos são logicamente derivados de B e H;
- todos os exemplos negativos são logicamente derivados de B e H;
- que a fórmula que descreve a hipótese (H) representa alguma regularidade dos exemplos positivos e é construída usando o *Background Knowledge (B)*
- a fórmula (H) pode ser descoberta por métodos ILP. Um conjunto de cláusulas pode ser construída pelo sistema ILP para representar a hipótese (H).
- a busca para satisfazer a regularidade (H) é organizada usando a lógica de inferência em PROLOG.
- numa aplicação de ILP em *Data Mining* o B, o E+, o E- e o H estão geralmente codificados em PROLOG.

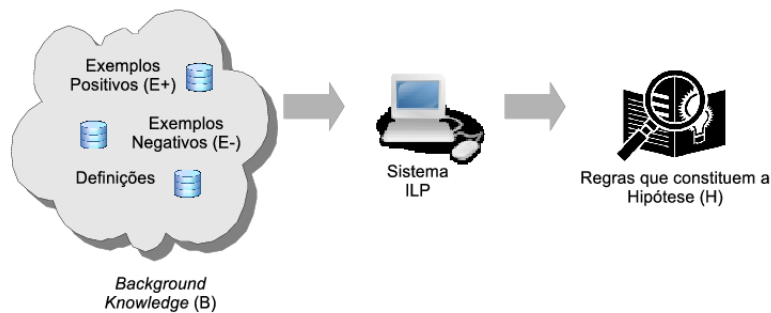


Figura 2.12: Uso de técnica ILP para Extração de Conhecimento

2.3.2.1 Implementações ILP em computação distribuída

Em [FSSC09], é descrito o estado da arte das principais formas de implementação de sistemas ILP utilizando técnicas de paralelismo.

Há basicamente duas formas de implementação do ILP em paralelo: *shared-memory* e *distributed memory*. O modelo *shared-memory* consiste de implementar o sistema ILP num ambiente de *cluster*, onde há a partilha da memória por vários processadores. Por outro lado, o modelo *distributed memory* consiste de aplicar o sistema ILP num ambiente computacional distribuído, ou seja, tanto o processador como a memória estão disponíveis em ambientes distintos e unidos por sistemas de comunicação (rede).

Do ponto de vista da aplicação dos algoritmos de implementação do ILP em paralelo, o autor [FSSC09] destaca ainda, que basicamente existem técnicas que simplesmente constroem modelos em paralelo em cada processador (*worker*) usando um subconjunto de dados e, em seguida, combinam os modelos encontrados em um único processador (*master*). Essa técnica costuma produzir os melhores resultados.

As estratégias de paralelismo são mostradas na Figura 2.13 e descritas em [FSSC05], [Fon06] e [FSSC09]. As abordagens de ILP paralelo baseam-se num tipo de algoritmo designado por MDIE (Mode-Directed Inverse Entailment) [Mug95]. São exemplos desta categoria de algoritmos os sistemas Progol [Mug95], Aleph [Sri03], April [FSC06] e IndLog [Cam00, Cam04].

Existem também estratégias de implementação de sistemas ILP paralelo que consideram os estágios de "busca", armazenamento dos dados (distribuído ou centralizado) e avaliação do modelo (cláusulas), de forma independente.

Observa-se uma forte tendência em tratar os algoritmos ILP-paralelo em único ambiente sem considerar que melhores resultados, como a redução do ciclo de processamento, por exemplo, pode ser otimizada. Para que isso ocorra são necessárias algumas estratégias no momento da implementação, dentre elas, considerar que cada estágio de execução de um sistema ILP possa ser tratado em ambientes distribuídos, ou seja, utilizando a abordagem *distributed memory*.

2.3.3 Data Mining Multi-relacional

Sob a perspectiva do KDD, podemos dizer que o ILP contribui para o desenvolvimento de técnicas e ferramentas de *Relational Data Mining (RDM)*. É designado assim por que os dados e os relacionamentos são alocados na forma de tabelas relacionais. Quando temos apenas uma tabela para representar os dados, dizemos que esta está na forma não normalizada. Enquanto as técnicas tradicionais de *Aprendizagem Computacional* buscam extrair ou encontrar padrões de dados em tabelas únicas, o RDM procura identificar novos padrões ou regularidades nos dados em Bases de Dados Relacionais. Os dados em Bases de Dados Relacionais estão distribuídos em múltiplas tabelas, deste modo a técnica de ILP consiste em identificar padrões que utilizam na sua definição relações existentes entre os dados distribuídos em múltiplas tabelas. A análise decorrente desta forma evita o problema causado pela agregação de dados, onde se junta os dados dispersos em múltiplas tabelas

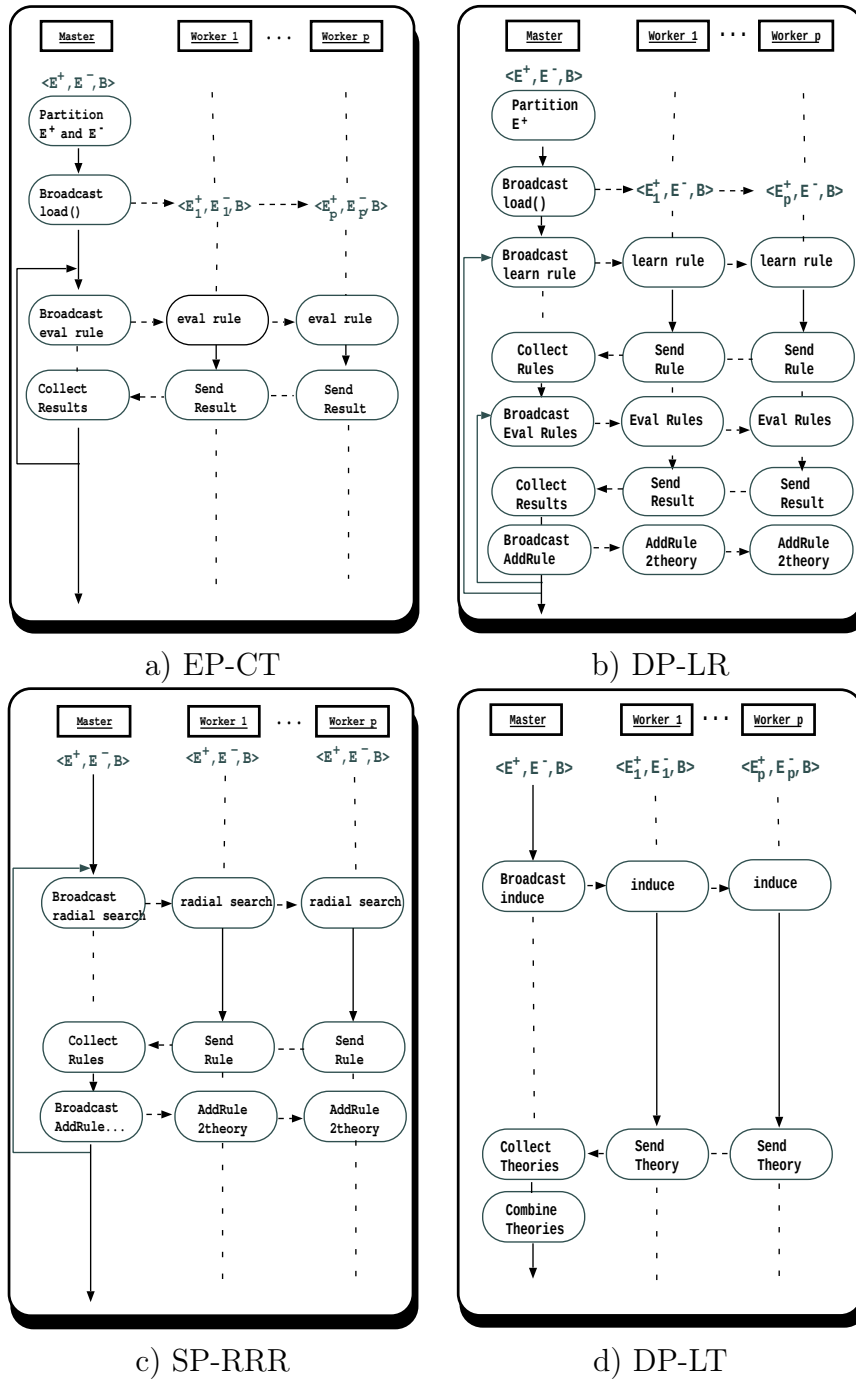


Figura 2.13: Esquemas simplificados de troca de mensagens dos diferentes tipos de algoritmos paralelos. (figura original de [FSSC09])

numa única, conforme ilustrado na Figura 2.14. Este processo de agregar dados de diferentes tabelas ocorre geralmente na etapa de pré-processamento e é requisito para algumas técnicas de *Data Mining*.

Suponhamos a seguinte situação ilustrada na Figura 2.15, onde na Tabela 1 - Clientes temos os atributos: *Id_Cliente*, *Nome*, *Idade*, *Sexo*, *Renda*; e na Tabela 2 - Compras temos os atributos: *Id_Cliente*, *Id_Prod*, *Data_Compra*, *Valor_Prod*, *Modo_Pagto*. Na etapa de pré-processamento, anterior a aplicação de técnicas de *Data Mining*, observa-se que decorre a junção de alguns atributos dando origem a outros, de modo a permitir uma melhor aplicação dos algoritmos. Neste caso ilustrado, temos a Tabela 3 - Transações Consolidadas com os atributos: *Id_Cliente*, *Idade*, *Renda*, *Qtde_Itens_Comprados*, *Valor_Total_Compras*.

Neste processo de agregar atributos, alguns padrões de dados deixam de ser encontrados, como por exemplo, a relação entre o atributo “Sexo” e a forma de pagamento preferida (“Modo_Pagto”). Contudo, preservando-se as tabelas 1 e 2 tal como estão definidas na base de dados relacional, e utilizando-se técnicas de ILP, o resultado já poderá ser outro.

Por exemplo, se desejamos identificar a preferência de pagamento dos clientes versus a renda ou outro atributo qualquer escolhido, não há como não ter que repetir o atributo “Modo_Pagto” também na tabela única, resultando na duplicação de outras informações. Neste caso em particular observamos a necessidade de tratarmos os dados na forma mais primitiva, sem agregação dos dados. Tal consideração representa que uma determinada compra pode dizer mais sobre o perfil do cliente do que o conjunto de todas as transações do cliente agrupadas.

A análise de dados via técnicas de *Data Mining* numa única tabela pode resultar em:

- ignorar informações relevantes dadas as condições de pré-processamento, principalmente na etapa de junção de atributos;

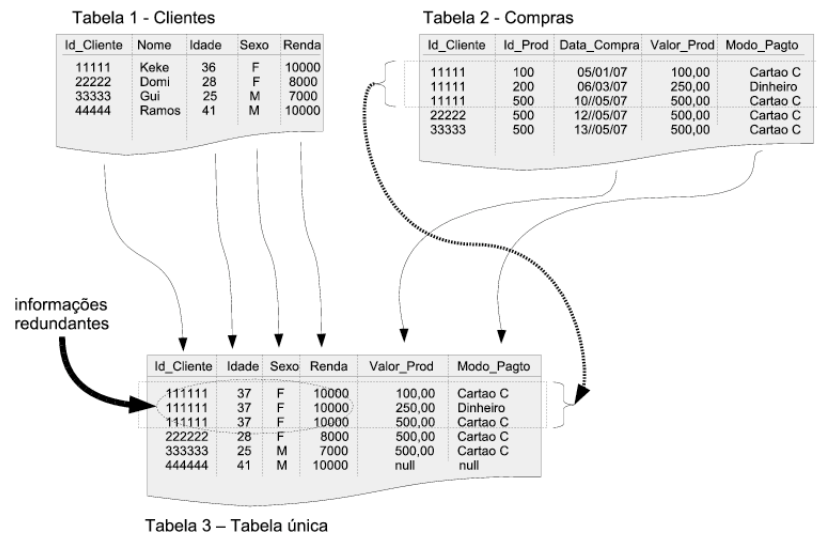


Figura 2.14: Tabela única com dados redundantes

- provocar redundância de informação e com isso prejudicar o desempenho de alguns algoritmos, influenciando em termos de eficiência e mesmo eficácia;

Vantagens de sistemas *Relational Data Mining*

O RDM permite reduzir a necessidade de pré-processamento já que não necessita que os dados dispersos em diferentes tabelas de uma BDR sejam transferidos para uma única tabela ou exportados para um ficheiro único. A junção dos dados numa única tabela pode omitir eventuais padrões de dados. Outra vantagem relevante do RDM é descobrir regularidades e padrões de dados diretamente em várias tabelas de uma BDR e assim, identificar padrões nas relações existentes entre as tabelas. Com isso, evita-se a redundância de dados decorrente do processo de junção de tabelas.

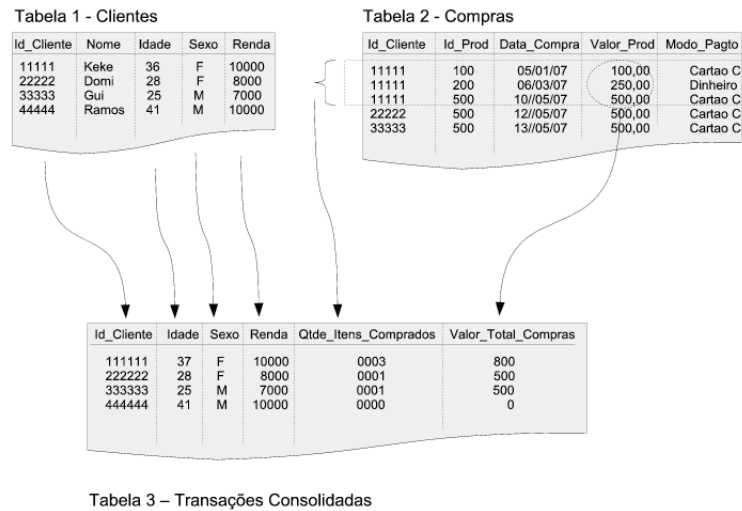


Figura 2.15: Tabelas de Clientes, de Compras e de Transações Consolidadas representando um processo de junção de dados.

2.3.4 Ferramentas para Aprendizagem Computacional

Apresentamos a seguir alguns dos produtos disponíveis no mercado utilizados em processos de *Data Mining*. Estas ferramentas podem ser tanto usadas isoladamente como podem aceder às bases de dados implementadas num SGBD.

2.3.4.1 Linguagem e ambiente R

O ambiente e linguagem para análise estatística denominada de *Projeto R* foi desenvolvido por Ross Ihaka e Robert Gentleman [IG96] da Universidade de Auckland na Nova Zelandia, a partir da linguagem para análise estatística *S* criada pela Bell Laboratories (Lucent Technologies), cuja denominação comercial é *S-plus*.

A linguagem de programação *R* é destinada à modelização de cálculos estatísticos e geração de gráficos. É uma linguagem bastante poderosa, flexível e modular. E,

com a utilização de pacotes extras, é viável implementar novas funções para áreas de estudo específicas.

O código fonte do *R* está disponível sob a licença GNU-GPL e as versões binárias fornecidas para diferentes sistemas operativos (Windows, Linux e Mac OS).

Há uma série de pacotes incluídos com a instalação do *R* e muitos outros disponíveis na rede de distribuição do *R* (CRAN - *Comprehensive R Archive Network*)¹⁵ que permite dar boa expansibilidade de aplicação à linguagem e ao ambiente.

O pacote RATTLE (*the R Analytical Tool To Learn Easily*)¹⁶ [Wil07], por exemplo, é destinado exclusivamente a adaptar o *R* para execução de tarefas *Data Mining*. O RATTLE tem uma interface simples e intuitiva que facilita a aplicação de técnicas *Data Mining* usando as funções estatísticas implementadas no ambiente *R*.

2.3.4.2 Weka

O WEKA (*Waikato Environment for Knowledge Analysis*) [Han01] é um ambiente de software que reúne um conjunto de algoritmos de *Aprendizagem Computacional* aplicáveis a tarefas de *Data Mining*. Todo o ambiente está programado em linguagem Java o que permite a interoperabilidade entre diversos sistemas operativos. Os algoritmos podem ser aplicados diretamente sobre os dados utilizando para isso a própria interface gráfica do WEKA, ou por intermédio de uma aplicação Java. O WEKA contém funcionalidades de pré-processamento e as principais tarefas de *Data Mining* implementadas entre as quais: classificação, regressão, segmentação, *clustering*, regras de associação e também a visualização de resultados. Na Figura 2.16 ilustramos a interface do *Explorer* do WEKA, com demonstração de um resultado de processamento.

¹⁵A URL disponível para acesso é <http://cran.r-project.org/>

¹⁶Disponível sob a licença GNU GPL (General Public License)

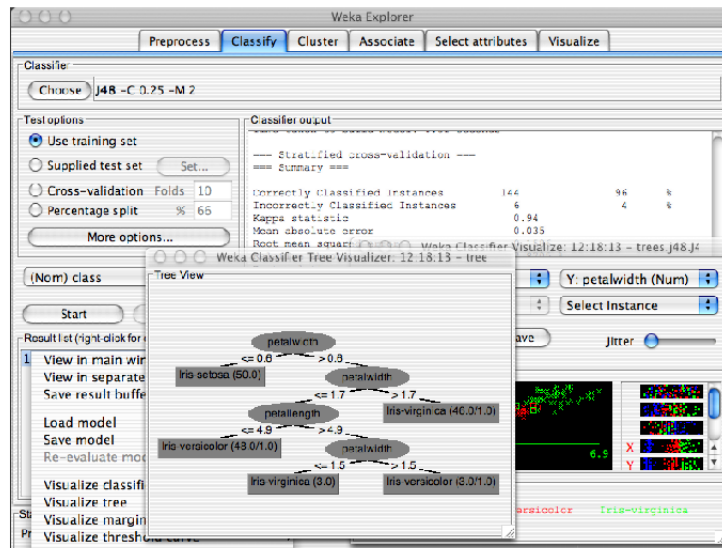


Figura 2.16: Interface *Explorer* do WEKA

O WEKA está disponível¹⁷ sob a licença GNU-GPL e as versões fornecidas para diferentes sistemas operativos (Windows, Linux e Mac OS X).

2.3.4.3 Yale

O *Yet Another Learning Environment* (YALE) [MWK⁺06], é também um ambiente destinado a aplicações de *Data Mining* usando algoritmos de *Aprendizagem Computacional*. Contém alguns algoritmos do próprio WEKA, e ainda outros mais recentes. Tem uma interface gráfica mais elaborada que o WEKA, e incorpora um conceito de modelização de todo o processo de KDD por intermédio de operadores e diagramas. Cada operador representa uma ou mais etapas do processo de extração de conhecimento. Assim, o utilizador escolhe um operador para tratar os dados, um ou mais operadores para a aplicação dos algoritmos, e por fim, um operador para integrar e visualizar os resultados. O fluxograma do processo vai sendo implementado

¹⁷URL: <http://www.cs.waikato.ac.nz/ml/weka/>

graficamente e posteriormente é possível ainda ser representado em formato XML, o que permite a esta ferramenta uma grande versatilidade de integração com outros ambientes similares.

O YALE é desenvolvido em Java e está disponível¹⁸ sob a licença GNU-GPL, sendo as versões fornecidas para diferentes sistemas operativos (Windows, Linux e Mac OS). Há uma versão comercial chamada de *Rapidminer*¹⁹.

2.3.4.4 Rapidminer

O sistema *Rapidminer*, uma versão aprimorada do YALE [MWK⁺06], é um produto bastante robusto para uso mais profissional em negócios empresariais, e incorpora uma série de funcionalidades advindas do YALE, notadamente conhecido como uma ferramenta de modelagem do processo KDD. O *Rapidminer* é suportado pela empresa Rapid-i e atualmente está na versão 5.1. em 2 modalidades de licenciamento: *Community Edition* e *Enterprise Edition*. Nas versões mais recentes o Rapidminer permite integração, além do WEKA, também com o ambiente *R*.

O *Rapidminer* incorpora ainda cerca de 15 *extensões* que acrescentam funcionalidades ao processo KDD, tais como: *text mining*, *visual mining*, *image processing*, *time series processing* e *web mining*. A Figura 2.17 demonstra a interface de visualização de resultados do *Rapidminer*.

2.3.4.5 SPSS Clementine

O SPSS (*Statistical Package for the Social Sciences*)²⁰, é um software que originalmente foi desenvolvido para aplicação de estatística nas ciências sociais. É atualmente o programa de análise estatística mais usado nas ciências sociais e estendido a outros domínios do conhecimento.

¹⁸URL: <http://sourceforge.net/projects/yale>

¹⁹URL do fornecedor do produto: <http://www.rapid-i.com>

²⁰URL do fornecedor do produto: <http://www.spss.com/>

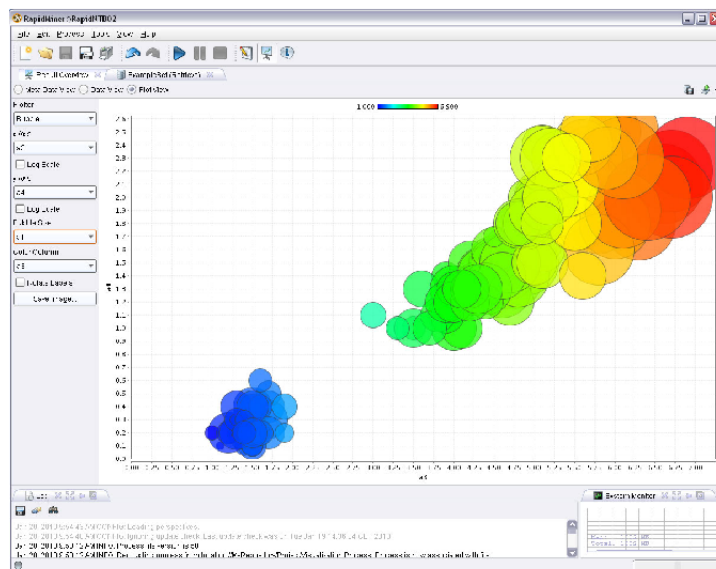


Figura 2.17: Visualização de resultados no Rapidminer

A versão SPSS *Clementine* é um produto derivado do original SPSS que implementa funcionalidades exclusivamente de *Data Mining* e segue rigorosamente as fases e tarefas definidas pela metodologia CRISP-DM. As principais tarefas *Data Mining* como classificação, regressão e associação fazem parte do produto.

2.3.4.6 SAS Enterprise Miner

O SAS²¹ é um sistema integrado de aplicações para análise de dados, que consiste na: recuperação de dados, gestão de arquivos, análise estatística, acesso a BD, geração de gráficos, e geração de relatórios. Implementa um conceito de ciclo de análise de dados que considera: *aceder, manipular, analisar e apresentar* os dados. Pode ser instalado em diversos ambientes operacionais disponíveis no mercado, sendo os programas e arquivos portáteis para qualquer um desses ambientes. O SAS *Enterprise Miner* é um produto exclusivamente destinado a aplicação de técnicas de *Data Mining*.

²¹URL: <http://www.sas.com/technologies/analytics/datamining/miner/>

O SAS dispõe uma vasta biblioteca com vários algoritmos clássicos de DM que podem ser organizados para execução de tarefas DM nas várias etapas do processo de extração de conhecimento, desde a sumarização até a exploração e simulação de resultados obtidos. Os resultados podem ser tratados no próprio ambiente SAS e também transferidos ou processados por outros ambientes computacionais.

2.3.4.7 KNIME

KNIME (Konstanz Information Miner)²² é um ambiente com uma interface bastante amigável capaz de processar, integrar e analisar dados com objetivo de extrair conhecimento útil e aplicável.

Segundo o fornecedor, é uma ferramenta bastante robusta capaz de ser instalada em ambientes corporativos para uso em aplicações de *business intelligence*. Tem como características principais a execução de tarefas, de forma remota ou não, integrado a ambientes SOA e com possibilidade de acesso via *web browser*. Pode ser utilizado em ambientes de *cluster* para execução de tarefas em paralelo. A Figura 2.18 apresenta uma visão geral da interface, com destaque para a representação do fluxo de análise de dados.

O KNIME pode ainda ser integrado aos outros ambientes de análise de dados, como o *Mathlab* e o *R*.

complexidade

2.4 Computação Distribuída

2.4.1 Computação em *Cluster*

A demanda por processamento computacional tem exigido computadores cada vez mais rápidos e eficientes. Entretanto, a aquisição de super-computadores pode ser

²²URL: <http://www.knime.org/>

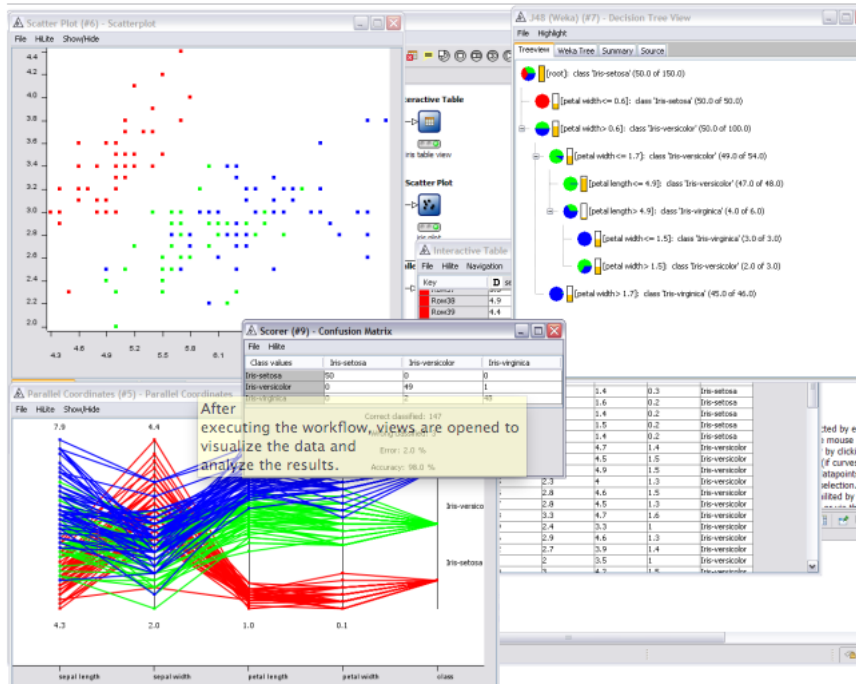


Figura 2.18: Visão de um fluxo de análise de dados no *KNIME*

um desafio bastante oneroso e não acessível para qualquer empresa ou universidade. Por outro lado, o aperfeiçoamento das redes de computadores permitiu o surgimento de um novo paradigma computacional: a *computação em Cluster* [BBH99, BB99].

A *computação em cluster* consiste em reunir um conjunto de computadores (> 2) de configuração heterogênea numa rede local única, conforme apresentado na Figura 2.19. Para o utilizador, a impressão é que trata-se de um único computador virtual de elevado desempenho capaz de processar as mais exigentes tarefas computacionais. Cada computador pertencente ao conjunto chama-se nó de *cluster*, sendo que há um nó principal responsável pelo gerenciamento, controle e distribuição das tarefas para os demais. Um requisito relevante aos computadores pertencentes ao mesmo *cluster* é que todos os nós estejam configurados para rodar o mesmo sistema operativo.

A aplicação da *computação em cluster* pode ter finalidades específicas, como:

- necessidade de elevado desempenho computacional (*high performance*);
- necessidade de alta disponibilidade de recursos (*high availability/failover*); ou

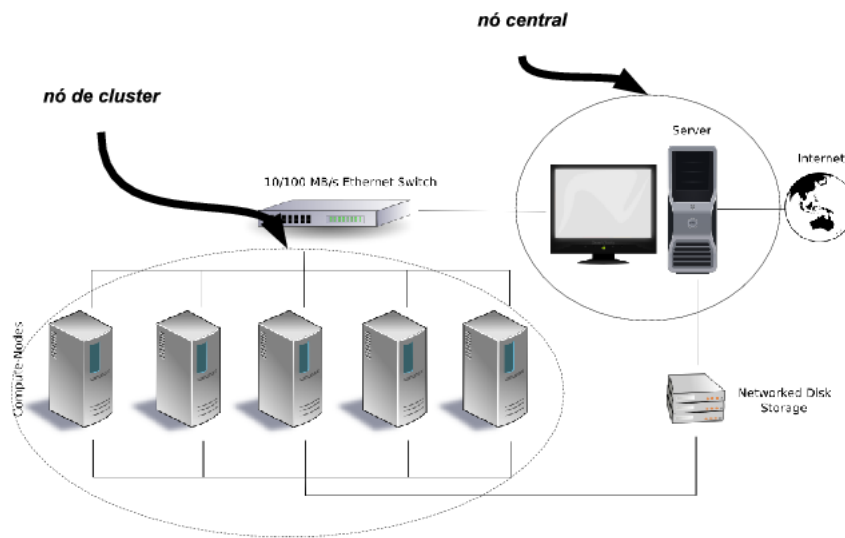


Figura 2.19: Representação de um *Cluster*

ainda,

- necessidade de balanceamento de carga (*load balancing*) requerida para processamento contínuo.

Em quaisquer desses casos é necessário que o sistema gerenciador do *cluster* seja capaz de detectar erros ou falhas, permitir ajustes e providenciar correções. Todas estas atividades devem ser executadas sem interromper o funcionamento do cluster.

Um dos sistemas gerenciadores de *cluster*, muito conhecido no meio acadêmico, é o *Beowulf* [SBS99]. Este sistema inicialmente proposto por Thomas Sterling²³ é composto por uma biblioteca de funções de comunicação, chamada de *Message Passing Interface (MPI)*, que permite aos programadores desenvolver programação

²³<http://www.cct.lsu.edu/~tron/Welcome.html>

paralela em linguagens como: C/C++ e FORTRAN, e também *Python*, *Java* e *Perl*. Com o uso da MPI, os programas podem ser executados em diferentes CPUs com a troca de mensagens e partilhamento de dados entre os *nós*. O *Beowulf* é um grande exemplo de um sistema de computação paralela com a implementação do conceito de PVM (Parallel Virtual Machine).

O *Beowulf* é composto de um nó central ou mestre denominado de *front-end*, cuja função principal é controlar o *cluster*, monitorando e distribuindo as tarefas, atua ainda como servidor de arquivos e permite a ligação entre os utilizadores e o *cluster*. Os demais *nós* são conhecidos como *clientes* ou *backends* (ou ainda nós escravos), e são exclusivamente dedicados para processamento das tarefas enviadas pelo *nó* central.

2.4.2 Grid Computing

A procura de computação de elevado desempenho para tratar problemas que requeiram intenso processamento numérico ou o processamento de grande quantidade de dados em áreas como as ciências, as engenharias e os negócios têm crescido nos últimos tempos [FK99]. Alguns requisitos de processamento vão além dos disponíveis em centros de computação disponíveis nas universidades ou nas empresas, requerendo muitas vezes avultados investimentos para se ter acesso a super-computadores.

Por outro lado, integrar centros de processamento dispersos geograficamente para tirar proveito de um processamento cooperativo e paralelo não é tarefa trivial, e envolve desde questões técnicas até sociais [FK99]. Surge um novo conceito na computação distribuída com o advento da Internet, a chamada meta-computação (*metacomputing*) [FK97]. A meta-computação ou computação omnipresente integrando diferentes recursos dispersos geograficamente tornou-se realidade a partir de conceitos implementados via *Grid Computing*, que consiste em integrar recursos computacionais distribuídos como sendo um único computador virtual capaz

de processar os mais rigorosos processos. Um meta-computador acaba por ser um *super-computador virtual* à disposição do utilizador em qualquer lugar em qualquer tempo [BBL02].

A ideia original de *Grid Computing* era a de integrar centros de computação de elevado desempenho, mas com a Internet este conceito passou a ser amplamente utilizado para integrar diferentes computadores distribuídos pela Internet para resolver problemas comuns ou de interesse geral. Com um ambiente de *Grid Computing* recursos computacionais heterogêneos reais (físicos) passam a ser um único recurso computacional virtual [FK99].

Um ambiente de *Grid Computing* inclui toda a infra-estrutura de *software* e *hardware* que permita ao utilizador ter acesso às facilidades de um ambiente de computação avançada a custos reduzidos. Deve ser totalmente transparente ao utilizador o facto de se estar usando vários computadores simultaneamente espalhados por uma rede na execução de determinada tarefa que exija HPC (*High Performance Computing*) [FK99].

De acordo com *Foster et al* [FK99], o conceito de *Grid Computing* pressupõe algumas características:

- não há um controle centralizado rígido como em grandes centros de processamento, pois os recursos devem estar dispersos numa rede. Entretanto, deve atender a requisitos de segurança, disponibilidade, integridade, confidencialidade no processamento de alguma tarefa;
- deve ser concebido em plataformas abertas e protocolos não proprietários de modo a que se possa facilmente escalar a sua esfera de atuação. Conectar duas universidades num único *Grid* pode exigir esforço adicional se determinados padrões e protocolos não forem cumpridos;

-
- a qualidade de serviço deve ser garantida, ou seja, tempo de resposta, disponibilidade, segurança e desempenho adequados;
 - cada centro de processamento é autónomo na sua administração;
 - não se pode comprometer a segurança dos utilizadores ou dos centros remotos;
 - não é necessário alterar a configuração do sistema operativo ou protocolos de rede;
 - os elementos distribuídos podem deixar de colaborar ao ambiente *Grid* deliberadamente;
 - é tolerante a falhas e seguro;
 - suporta componentes de *hardware* e *software* heterogéneos;
 - utiliza padrões e tecnologias existentes e pode interagir com aplicações legadas;
 - proporciona adequada sincronização entre os componentes da aplicação;

Ainda de acordo com *Foster et al* [FK99] os princípios que um ambiente *Grid Computing* devem seguir são:

- Escalabilidade: disponibilizar mais processamento com o acréscimo de mais recursos computacionais à medida do necessário.
- Dinamismo e adaptabilidade: num ambiente Grid, o recurso não estar disponível pode ser a regra e não a exceção. Temos portanto, um ambiente dinâmico e flexível capaz de atender as demandas adaptando-se as diferentes configurações e disponibilidades de recursos.
- Heterogeneidade de recursos: tratar diferentes recursos de diferentes características e funcionalidades dispersos numa rede;

Foster [FK99] e Baker [BBL02] advogam quando o *Grid Computing* deve incluir as seguintes funcionalidades:

- (a) administração hierárquica: cada centro gere os seus próprios recursos mas permite o uso pelo ambiente como todo;
- (b) serviços de comunicação: tudo que se refere a QoS^{24} , banda de comunicação instalada, latência e tolerância a falhas em termos de comunicação é garantido por essa camada;
- (c) serviços de informação: gerir recursos e serviços disponíveis e o estado de utilização;
- (d) serviço de nomes: cada recurso disponível num ambiente *Grid* deve ter um nome único que permita a sua identificação por todo o ambiente;
- (e) sistema de arquivos distribuído: o acesso aos dados distribuídos deve ser transparente aos utilizadores, não importando onde os dados estão e estar sempre disponíveis;
- (f) controle de segurança e autorização: requisitos de segurança como confidencialidade, autenticação, integridade e disponibilidade da informação são fundamentais;
- (g) controle de disponibilidade e tolerância a falhas;
- (h) sistema de gestão de recursos e agendamento (programação de eventos);
- (i) economia computacional e negociação de recursos;
- (j) novas ferramentas e paradigmas de programação num ambiente Grid requerem bibliotecas, interfaces, API (*Application Program Interface*);

²⁴*Quality of Service*

-
- (k) interface com o utilizador deve ser amigável, intuitiva e apresentar todos os recursos disponíveis. Deve estar disponível em qualquer plataforma em qualquer altura. Pode-se usar a *Web* para submeter e acompanhar uma tarefa;

Um processo *Grid Computing* inclui as seguintes etapas [FK99]:

1. integrar recursos de *software* e *hardware* numa única rede como um recurso virtual único à disposição do utilizador;
2. implementar uma camada intermédia (*middleware*) para gerir recursos disponíveis e torná-los disponíveis de forma transparente;
3. desenvolver ferramentas para gerir as aplicações e a infra-estrutura disponível;
4. adaptar as aplicações de forma a usufruírem dos recursos *Grid*;

Segundo [FK99], um Grid Computing pode ser sistematizado em quatro componentes principais:

- a) ***Grid Fabric***: detectar e configurar recursos disponíveis numa rede (*lan*, *wan*, *man*) tais como: computadores pessoais, *server*, *clusters*, *mainframes* em diferentes sistemas operativos.
- b) ***Middleware***: gerir a qualidade de serviço (QoS), alocação de recursos remotos, gestão de segurança e armazenamento;
- c) ***Grid development environment and tools***: dispor de bibliotecas para desenvolver aplicações distribuídas;
- d) ***Grid Applications and Portals***: desenvolver aplicações distribuídas em linguagens específicas, uso de interfaces com o utilizador via *Web*, uso de bibliotecas.

2.4.3 Iniciativas *Grid Computing*

Embora um ambiente *Grid Computing* deva ser usado para reduzir investimentos computacionais com a partilha de recursos computacionais, algumas iniciativas atualmente não têm previsto o lucro ou não remuneram o utilizador pelo uso de sua estação conectada à Internet. O SETI@home (SETI - *Search for Extraterrestrial Intelligence*) procura inteligência extra-terrestre a partir da análise de informações colectadas pelo radiotelescópio Arecibo [KWA⁺01]. Este aparato situado em Porto Rico analisa constantemente o espaço via sinais de rádio, gerando e armazenando extensas bases de dados para posterior análise.

Outro projeto é o *distributed.net* que conecta o equivalente a 160.000 computadores (Pentium II 266MHz) trabalhando 24 horas por dia, 7 dias por semana, 365 dias por ano para simplesmente decifrar o algoritmo de criptografia de dados RSA²⁵, sendo que a versão de RC5-64 foi decifrada em Julho de 2002 decorridos 1757 dias de intenso processamento [KWA⁺01].

A demanda por computação avançada é crescente, principalmente para tratar grandes quantidades de dados. O potencial da abordagem *Grid* demonstra que num futuro não muito distante teremos a disposição o fornecimento de unidades de processamento como temos na rede de distribuição de energia eléctrica. Bastará conectar nossa estação à Internet num site de fornecimento e poderemos executar complexos cálculos matemáticos ou proceder extensas análises de dados pagando alguma taxa de uso. É possível até que em sendo nós fornecedores de um ciclo de processamento possamos ser remunerados pela iniciativa [FK99].

²⁵RSA é o acrónimo dos nomes dos criadores do algoritmo: Ronald Rivest, Adi Shamir e Leonard Adleman

2.4.3.1 Globus

É considerado um *framework* de meta-computação e abrange todos os serviços definidos para um ambiente *Grid Computing* padrão [FK99] conforme descrito na secção anterior. É composto por um elemento central chamado de *Globus Metacomputing Toolkit* (GMT) que fornece uma série de ferramentas para o desenvolvimento das aplicações distribuídas. Há um conjunto de módulos que implementam os diferentes serviços de um ambiente *Grid* tais como: gestão de recursos e serviços, detecção de falhas, comunicações, segurança, gestão de ficheiros, entre outros. Cada uma dessas funcionalidades está implementada em módulos específicos, entre os quais: GRAM, GSI, MDS, HBM, GASS, GEM, GARA. Cada um desses módulos é baseado num conjunto de API que implementam os respectivos serviços de um *Grid* [FK97].

O *framework* Globus²⁶ é flexível, multiplataforma e modular e segue a arquitectura padronizada pelo *Open Grid Services Architecture (OGSA)* do *Global Grid Forum* (GGF), permitindo assim que infra-estruturas tecnológicas diferentes possam partilhar recursos de forma transparente ao utilizador. É um produto que está disponível sob a licença GNU *General Public License* (GPL), sendo as versões binárias e respectivos códigos fontes fornecidos para diferentes sistemas operativos.

2.4.3.2 Condor

O ambiente Condor [LLM88] é um projecto de pesquisa da Universidade de Wisconsin-Madison²⁷ e caracteriza-se por gerir um ambiente composto de vários computadores servidores ou desktop, dedicados ou não, e/ou também ambientes computacionais que utilizam sistemas em *cluster*, como o *Beowulf*²⁸. É basicamente um ambiente

²⁶URL:<http://www.globusconsortium.org/>

²⁷URL:<http://www.cs.wis.edu/condor>

²⁸URL: <http://www.beowulf.org/>

Grid Computing para gerir a capacidade de processamento de recursos computacionais de um “campus” ou de uma organização, permitindo executar tarefas que exigem alta disponibilidade e alto desempenho computacional. Tem como características principais a gestão de *jobs*, definição de políticas de uso do ambiente distribuído e gestão e monitoramento dos recursos computacionais. Há uma variante denominada Condor-G [FTL⁺01] que facilita a integração com ambientes *Grid Computing* baseado no *Globus Toolkit* [FK97].

É um produto que está disponível sem custo de licença, sendo as versões binárias e respectivos códigos fontes fornecidos para diferentes sistemas operativos (Linux, Windows).

2.4.3.3 BOINC

O BOINC (Berkeley Open Infrastructure for Networking Computing)²⁹ [And04] é uma arquitectura computacional distribuída que utiliza recursos computacionais registados voluntariamente a partir da Internet. Pode ser implementado num ambiente de intranet. É uma plataforma que permite aos cientistas gerir recursos computacionais públicos disponíveis na Internet em benefício de projetos científicos de grande dimensão. A arquitectura é composta por servidores de projetos que operam suas próprias aplicações e bases de dados, e partilham processamento voluntário disponível a partir de estações ligadas à Internet que “doam” ciclos de processamento por intermédio de software cliente instalado para esta finalidade.

Alguns projetos como o *SETI@home*³⁰, *Folding@home*³¹ utilizam a plataforma BOINC. Um exemplo de iniciativa regional é o projeto Ibercivis³². O Ibercivis é uma plataforma de computação voluntária que permite a colaboração dos cidadãos na investigação científica. Neste projeto, o utilizador é convidado a partilhar ciclos de

²⁹URL: <http://boinc.berkeley.edu/>

³⁰URL: <http://setiathome.berkeley.edu/>

³¹URL: <http://folding.stanford.edu/>

³²Ibercivis - www.ibercivis.pt

processamento para uma investigação científica. O Ibercivis foi lançado na Espanha em 2008, com o alargamento da iniciativa a Portugal com o objetivo de assumir uma dimensão Ibérica.

O BOINC está disponível sob a licença GNU-GPL, sendo as versões binárias e respectivos códigos fontes fornecidos para diferentes sistemas operativos (Linux, Mac OS X e Windows).

2.4.3.4 Hadoop

O Projeto *Hadoop*³³ é coordenado pela Fundação Apache. Esta arquitetura é inspirada em modelos criados por engenheiros da Google, o Google File System (GFS) [BDH03], o *Bigtable* [CDG⁺06] e o modelo de processamento de dados de forma distribuída, conhecido como MapReduce [DG08].

É, portanto, basicamente uma arquitetura de processamento distribuído baseada em cluster que consiste de dois módulos: o componente de gerenciamento de ficheiros, conhecido como HDFS (hadoop file system) e o componente de processamento de dados, baseado no conceito do MapReduce [DG08] apresentado pelos engenheiros da Google. O HDFS é um sistema de ficheiros, similar ao sistema de ficheiros do sistema operativo Linux, em termos de funcionalidades e utilidades. O HDFS possui um conjunto de comandos básicos, inclusive com a mesma sintaxe do Linux, para manipular ficheiros em várias funções essenciais como copiar, listar, apagar ou visualizar conteúdo, entre outras. O componente MapReduce é o responsável pelo processamento distribuído dos programas (algoritmos) que devem seguir algumas regras e utilizar-se de API Java do HADOOP. Na Figura 2.20 é apresentado os componentes e a relação entre eles.

O *Hadoop* é desenvolvido em Java, e por enquanto, os componentes HDFS e MapReduce só operam em sistemas operativos do tipo Linux, mas isso não impede que um utilizador possa obter os benefícios de uso de uma arquitetura baseada em

³³URL: <http://hadoop.apache.org/>

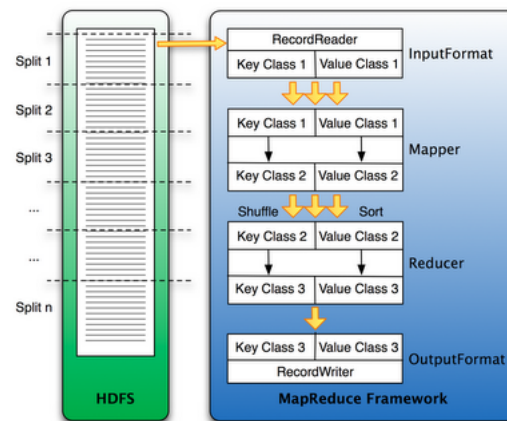


Figura 2.20: Arquitetura do Projeto *Hadoop* [Whi09]

Hadoop, a partir de um browser.

A arquitetura Hadoop implementa o conceito de *Mestre-escravos* na forma de um *Cluster Hadoop* e opera em máquinas de baixa confiabilidade, visto que cada nó cliente ou escravo atua de forma independente executando pequenas tarefas sob pequenas porções de dados (dataset). Requer a definição de um ou mais nós na função de mestre para o gerenciamento e distribuição das tarefas.

Outros projetos da própria Fundação Apache são complementares ao Hadoop como o Apache *MAHOUT*³⁴, que trata de problemas específicos na área de Aprendizagem Computacional mantendo uma biblioteca de algoritmos aplicáveis ao Data Mining, agregando novas funcionalidades ao Hadoop originalmente implementado.

2.4.3.5 MapReduce

O conceito de *MapReduce* [DG08] foi primeiramente aplicado pelo Google para tratar o desafio de processar a grande quantidade de dados coletados por conta do sistema de buscas [CDG⁺06]. O princípio é o de *dividir para conquistar*, ou seja, uma extensa lista de dados é dividida em porções (tarefa e dados) e cada qual proces-

³⁴URL: <http://mahout.apache.org>

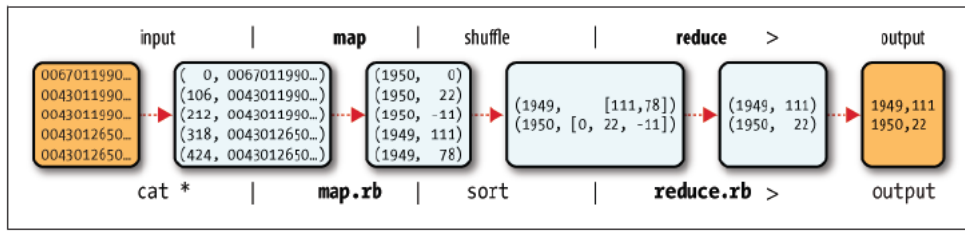


Figura 2.21: Fluxo exemplo de execução das funções de mapeamento (*Map*) e redução (*Reduce*) de um conjunto de dados [Whi09]

sada individualmente por uma máquina. Ao final de cada processamento individual da tarefa, os resultados são consolidados. É um modelo distribuído e paralelo de processamento de dados, e tende a ser mais eficiente a medida que mais tarefas são criadas e distribuídas para diferentes máquinas e assim serem tratadas em paralelo num ambiente computacional distribuído.

O *MapReduce* aplica duas funções sobre um conjunto de dados: a função Map (mapeamento) e a função Reduce (redução). Cada uma das funções processa uma tupla (chave, valor) como entrada e saída do processamento, conforme ilustrado na Figura 2.21. Estas funções derivam de linguagens de programação funcionais e permitem assim simplificar o tratamento dos dados, pois cada porção é processada de forma independente [Whi09].

A estrutura do MapReduce permite que as funções de mapeamento e redução sejam aplicadas ciclicamente ou recursivamente de modo a atingir os objetivos da análise de dados. Na Figura 2.22 é apresentado um diagrama ilustrativo do processo.

2.4.4 Computação Peer-to-peer(p2p)

Embora os termos computação oportunista [BP01] ou computação voluntária [Sar01] não possam ser ainda tão populares, o fato é que convivemos com computadores pessoais ou computadores organizações que estão disponíveis num período de tempo, e em alguns casos literalmente desligados. Obviamente nós não podemos estar em frente ao computador para fazer uso do recurso 24 horas por dia. E mesmo

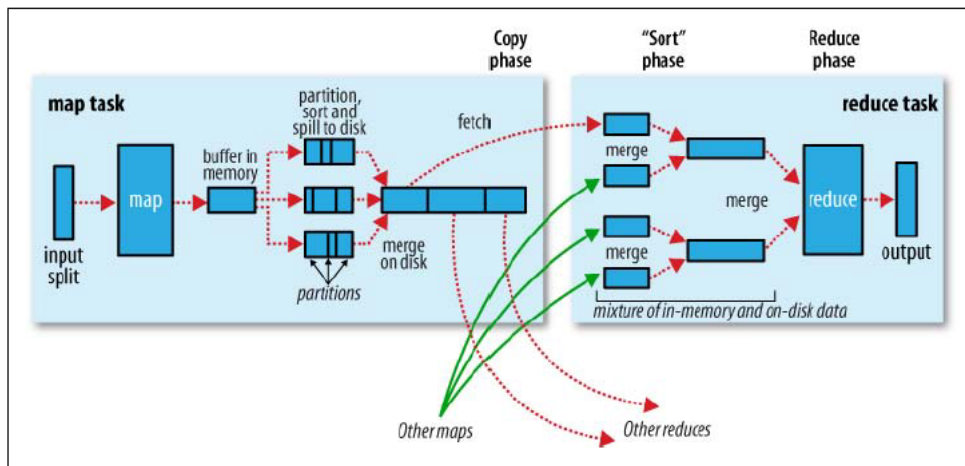


Figura 2.22: Diagrama de execução das funções de mapeamento (*Map*) e redução (*Reduce*) de um conjunto de dados de forma recursiva [Whi09]

computadores de uso comum, como em faculdades e escritórios mantêm-se ociosos uma significativa parcela de tempo, nem que seja durante a madrugada. O sucesso das redes *peer-to-peer* para compartilhamento de ficheiros demonstram que recursos computacionais podem ser compartilhados beneficiando uma parcela significativa de utilizadores quando há cooperação entre as partes envolvidas.

2.4.5 Computação em Nuvem (*Cloud Computing*)

O recente termo *Cloud Computing* é bastante amplo e abrangente para caracterizar uma nova forma de acessar, armazenar e processar os dados. O que importa nesta nova forma de *computar* é que as aplicações e os dados estão sempre disponíveis e acessíveis ao utilizador, não tendo qualquer importância onde estão guardadas as informações, ou ainda, qual dispositivo está sendo utilizado para acesso [Mil08].

Nesta nova modalidade de computação, o utilizador não precisa se preocupar com funções básicas da computação, tais como: *back-up*, *memória disponível*, *capacidade de processamento* ou mesmo *configurar um hardware ou periférico*. Enfim, nada disso parece ser preocupação do utilizador mas das empresas que disponibilizam os serviços para esta nova modalidade de computação.

Ao utilizador cabe escolher o dispositivo de acesso (*PC, Tablet, ou um Smartphone*) e o serviço desejado, sem a preocupação de quanto espaço ocupar. A infra-estrutura é gerida por um fornecedor qualquer, que pode estar a milhares de quilómetros de distância do utilizador. Esta é a nova geração da computação que têm atraído mais e mais utilizadores dos serviços baseados em *Cloud Computing*. Já são inúmeras as empresas que fornecem serviços, desde *Webmail* até serviços de armazenamento virtual com facilidades de sincronismo para diferentes dispositivos. Alguns serviços são gratuitos e outros já exigem o pagamento de alguma quantia para dispor de mais espaço, aplicações ou serviços adicionais.

Os sistemas de *Cloud Computing* pode ser categorizados [DKFY10] em:

- a) ***Infrastructure-as-a-Service (IaaS)***: este tipo de infra-estrutura caracteriza-se por fornecer serviços de processamento e/ou armazenamento, e são baseados em grandes *datacenter*, onde as funções de redundância, *back-up* e alta disponibilidade são garantidas. Os principais clientes deste modelo são geralmente empresas interessadas em não imobilizar grandes recursos financeiros e pagar por demanda de serviço.
- b) ***Platform-as-a-Service (PaaS)***: este modelo caracteriza-se por dispor de recursos computacionais para hospedar aplicações e também fornecer toda a infra-estrutura necessária para que as empresas possam construir suas próprias aplicações sob medida.
- c) ***Software-as-a-Service (SaaS)***: este modelo já bem mais amplo fornece serviços sob medida aos utilizadores. É o caso de aplicações que oferecem espaços de armazenamento, serviços de mensageria (*webmail*), entre outros que podem estar disponíveis em diferentes dispositivos de acesso.

Qualquer que seja a necessidade do contrante ou utilizador, estes modelos de *Cloud Services* tem em comum a oferta e garantia de serviços voltados: a segu-

rança, o desempenho, a escalabilidade, a virtualização, ao armazenamento e a alta disponibilidade.

Na era *Cloud Computing* não há necessidade de investir em recursos computacionais complexos e de alto investimento, os interessados contratam o serviço e não mais a infra-estrutura e tudo o mais fica por conta do *provedor de serviços* que se encarrega em manter o ambiente computacional.

2.5 Conclusões

Neste capítulo apresentamos o *Estado da Arte* de alguns dos temas tratados no âmbito desta dissertação e que subsidiam a proposta de trabalho apresentada nos Capítulos 3 e 4. Os temas discutidos e apresentados nas áreas de *Extracção de Conhecimento*, *Aprendizagem Computacional* e *Computação Distribuída* não esgotam o assunto das respectivas áreas de conhecimento, mas permitem uma adequada contextualização para subsidiar o trabalho desenvolvido nesta tese.

Observamos que algumas técnicas e métodos para KDD mais complexos podem exigir um maior poder computacional para tratamento de grande quantidade de dados. Constatamos ainda, que algumas ferramentas disponíveis para KDD não necessariamente endereçam o problema de tratar grande quantidade de dados, e quando o fazem, são produtos comerciais de tradicionais fornecedores de mercado como: IBM, Oracle, SAS Institute, entre outros. E portanto, exigem um razoável investimento de recursos, seja tanto na compra de software como hardware para suportar as aplicações de KDD. Por outro lado, notamos alguma oferta de soluções de software, como o *WEKA*, *Rapidminer*, *R*, *KNIME*, em licenciamento GNU/GPL, mas que vão também exigir investimentos em hardware, e/ou ainda, adaptações para suportar adequadamente um ambiente em computação distribuída, como é o caso dos ambientes *Grid Computing* (*Condor* e *Globus*). E não menos importante, os serviços baseados em *Cloud Computing* estão se multiplicando e trazendo cada vez mais adeptos interessados em usufruir da computação baseada em serviços, dispensando

a necessidade de manter infra-estruturas computacionais, custosas e complexas.

Capítulo 3

Sistema HARVARD

No Capítulo 2 tivemos a oportunidade de discorrer sobre os principais avanços nas áreas de *Extração de Conhecimento, Aprendizagem Computacional e Computação Distribuída*. A sinergia destas áreas permite propor soluções para questões de análise de grandes quantidades de dados com objetivo de extrair informações úteis à tomada de decisão, ou seja extrair conhecimento a partir de dados brutos.

Uma análise de dados eficaz, e também eficiente, requer uma infra-estrutura capaz de suportar técnicas e algoritmos avançados definidos para o KDD. Entretanto, se de um lado temos o desafio de aprimorar o processo de análise de dados nas atuais organizações, temos também o desafio de viabilizar economicamente este mesmo processo. A utilização de recursos computacionais distribuídos para aplicações complexas tem sido uma tendência principalmente em campus universitários para apoiar projetos de investigação. Com isso reduz-se a necessidade de investimentos adicionais em termos de recursos computacionais ou aquisição de super computadores a custos elevados.

O sistema HARVARD cujo o acrónimo significa *HARVesting Architecture of idle machines foR Data mining* é um ambiente computacional distribuído preparado para suportar tarefas de extração de conhecimento de grandes quantidades de dados utilizando técnicas de *Data Mining* com recurso aos algoritmos de *Aprendizagem*

Computacional. Com o HARVARD podemos ter disponível como que um “*super-computador virtual*” usando computadores convencionais dispersos e ociosos numa rede. E assim, como diz o ditado popular: “a união faz a força”, já que a natural ociosidade de computadores numa rede local, em determinados períodos de tempo, pode ser convertida em processamento útil, gerando a possibilidade de novas aplicações. Alguns autores, entre eles [FK99], defendem que quaisquer aplicações que necessitem de computação de elevado poder computacional, onde determinado problema possa ser subdividido em pequenas porções para a busca de uma solução integrada, pode usufruir dos benefícios de um “*super-computador virtual*”.

Com esta abordagem desenhamos uma plataforma computacional que tem como características essenciais, quanto ao uso de recursos, a escalabilidade e a adaptabilidade. É escalável enquanto puder fazer uso de recursos disponíveis, gerando um poder computacional compatível com a exigência das aplicações de KDD. Em complemento, é adaptativo por fazer uso de recursos computacionais em diferentes sistemas operativos (Linux e Windows) com a adoção da plataforma Java como ambiente de integração e portabilidade.

Além disso, o HARVARD é uma plataforma de baixo custo pois utiliza recursos computacionais disponíveis e ociosos de uma organização, e portanto não perturba o normal uso dos recursos na rotina diária. É uma plataforma flexível e versátil pois utiliza diferentes ferramentas pré-existentes de análise de dados sem re-programações ou adaptações, sendo independente quanto à ferramenta requerida para a tarefa de KDD. É também uma plataforma fiável já que incorpora características de segurança e controle das operações e com esquemas de tolerância a falhas. E é ainda fácil de utilizar já que dispõe de uma linguagem de especificação de tarefas de KDD simultaneamente poderosa e de fácil compreensão por não especialistas.

3.1 Arquitetura do Sistema HARVARD

A infra-estrutura básica do HARVARD é composta por dois tipos de *nós*: um "*nó servidor*" e vários "*nós clientes*". Para além destes "componentes principais" a arquitetura pode aceder a um servidor *Web* para obter programas de análise de dados e ainda a um sistema gestor de BD para obter os dados e armazenar os resultados e efetuar um *back-up* atualizado de informação sobre os nós clientes e o estado geral de desenvolvimento da execução das tarefas. Uma visão global da arquitetura da infra-estrutura básica pode ser vista na Figura 3.1.

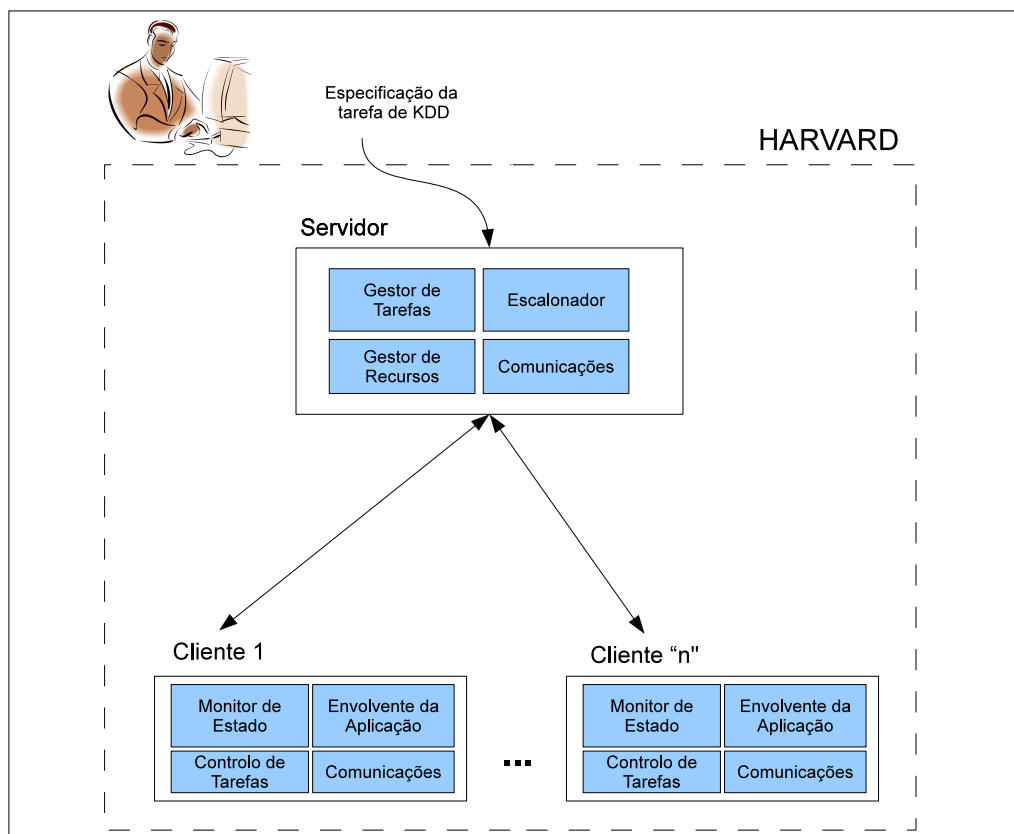


Figura 3.1: Arquitetura HARVARD

A arquitetura inclui as seguintes características:

- Plataforma baseada numa arquitetura *Servidor/Cliente* (*Master/Worker*);

-
- Os nós *Cliente* estão sujeitos a uma política de utilização que permite a sua ativação/desativação sempre que essa política o determinar;
 - A arquitetura é modular, o que facilita a inclusão de novas funções;
 - A linguagem de programação da infra-estrutura dos módulos do *Servidor* e *Cliente* é o *Java*, o que concede elevada portabilidade à arquitetura;
 - A infraestrutura básica é independente do(s) algoritmo(s) de análise de dados;
 - Qualquer *Cliente* ou o *Servidor* pode ser executado tanto em sistemas operativos *Linux* como *Windows*;
 - Em caso de falha de um qualquer nó a tarefa em execução nesse nó é reiniciada num outro nó (tolerância a falhas dos *Clientes*);
 - Em caso de falha do *Servidor* um dos *Clientes* (previamente designado) assume o papel de servidor (tolerância a falhas do *Servidor*);
 - Permite acesso direto a Bases de Dados e a repositórios *Web* para transferência de dados ou de programas

3.2 Processo de Análise de Dados no HARVARD

O HARVARD inicia o processo de Análise de Dados lendo, de um ficheiro, a especificação do fluxo do processo (*workflow*) juntamente com a descrição de cada uma das tarefas do processo KDD. O fluxo do processo é representado por um grafo com dois tipos de nós: sequenciais e paralelos. Cada nó regista o conjunto de tarefas a serem executadas, representadas e designadas como UT (unidades de trabalho). Na Secção 3.5, que descreve a arquitectura do HARVARD, as UT são descritas em maior detalhe.

No caso de um nó sequencial, as tarefas deverão necessariamente ser executadas em ordem de precedência devido à sua interdependência. Por outro lado, os nós paralelos podem ter tarefas executadas em simultâneo.

Na Figura 3.2 é apresentado um exemplo de descrição de um processo KDD. A figura mostra ainda em detalhe um conjunto de tarefas definidas $T1 \sim T15$ que abrangem todo o processo, desde o pré-processamento, seleccionando os atributos mais relevantes, até à consolidação dos resultados ($T15$). A descrição de cada tarefa Tn está codificada em XML¹ em ficheiros distintos que serão processados pelo módulo Gestor de Tarefas (GT) do HARVARD, conforme detalhado na Secção 3.5.1.

¹*Extensive Markup Language*

```
# This is the Task control description
# using the Task Control Language (.tcl file)

seq # execute tasks sequentially
  T1 # choose a 70%/30% train/test set

  par # execute the tasks in parallel
    seq
      T2 # generate a dataset without Att1
      par
        T3 # eval dataset without Att1 using m = 10
        T4 # eval dataset without Att1 using m = 50
      endpar
    endseq

    seq
      T5 # generate a dataset without Att5
      par
        T6 # eval dataset without Att2 using m = 10
        T7 # eval dataset without Att2 using m = 50
      endpar
    endseq

  barrier T[3-4], T[6-7] # wait for all of the tasks to finish

  T8 # choose the best set of attributes
  T9 # produce a 5-fold CV blocks

  par
    T[10-14] # do each CV i

  barrier T[10-14] # wait for all CV folds

  T15 # run with all data to produce the final theory

endseq
```

Figura 3.2: Descrição do fluxo do processo KDD em tarefas sequenciais e paralelas.

3.3 A especificação das tarefas

A linguagem de especificação das tarefas, conforme ilustrado na Figura 3.3 é XML. A partir deste contexto são definidas variáveis com seus respectivos parâmetros que posteriormente serão interpretadas pelo HARVARD num dos seus módulos que trata da especificação da tarefa - o Gestor de Tarefas (GT).

Na especificação de cada tarefa, o utilizador do HARVARD deverá informar em detalhe toda a informação relevante. Isso inclui descrever entre outros itens:

- local e nome(s) do(s) ficheiro(s) que contém os dados para análise;
- algoritmo(s) a ser(em) usado(s) com respectivos parâmetros de análise;
- nome da aplicação que implementa o algoritmo escolhido (p.ex: C4.5)
- ferramentas de pré-processamento a serem usadas (p.ex: aplicação de *script* “Perl” para eliminar ou consolidar atributos);
- local e formato de representação dos resultados.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE tasks SYSTEM ''tasks.dtd'' >
# Run C4.5 on a data set stored in a Data Base and return the
# result in a "results table" of the same DB
##### Task Description Language (.tdl) file #####
<workunit>
  <id> T1 </id>

  ##### data #####

  <fetch-data>
    <method> jdbc </method>
    <server> dbserver </server>
    <user> dmuser </user>
    <db> kdd99 </db>
    <password> _____ </password>
    <db-access>
      <source-data>
        <query> select * FROM data LIMIT 5000 OFFSET 100 </query>
        <file> kdd99.data </file>
      </source-data>
      ...
    </db-access>
  </fetch-data>

  ##### source code #####

  <fetch-code>
    <source-code> # C4.5 code to construct the Decision Tree
    <getMethod> http </getMethod>
    <url> http://www.fe.up.pt/~rcamacho/c4.5 </url>
    </source-code>
    ...
  </fetch-code>

  ##### sub-tasks execution #####

  <execution>
    <results-storage>
      <method> jdbc </method>
      <server> dbserver </server>
      <user> dmuser </user>
      <db> kdd99 </db>
      <password> _____ </password>
    </results-storage>
  ##### sub-task 1 #####
  <subtask>
    <exec-mode> noninteractive </exec-mode>
    <exec-command> c4.5 -f kdd99 -m 100 -u </exec-command>
    <results-file> c45.output </results-file>
    <exec-time> 30 </exec-time>
  </subtask>
  ##### sub-task 2 #####
  ...
  </execution>

  ##### required resources #####

  <resources>
    <hd> 10 </hd>
    <ram> 0.5 </ram>
    <opsystem> linux </opsystem>
  </resources>
</workunit>

```

Figura 3.3: Descrição em detalhe de uma tarefa de KDD específica.

3.4 O uso da linguagem XML no HARVARD

A linguagem XML usada no HARVARD permite que o utilizador possa especificar claramente o processo de KDD pois "faculta um conceito para descrever, armazenar, permutar e manipular dados estruturados" [Hei01, Bra06].

Isso é possível porque o XML permite estruturar a informação de forma hierárquica e independente, facilita a edição devido à sintaxe simples (qualquer editor de texto pode ser usado), e permite a fácil compreensão dos dados estruturados, já que não requer nenhuma sofisticada ferramenta de tratamento ou visualização. No caso do HARVARD os dados estruturados em XML permitem a fácil interpretação pelos diferentes módulos da plataforma, e também a geração dos resultados para posterior integração com outras ferramentas de análise de dados que importem formato XML, e facilita o armazenamento em bases de dados.

Considerando que o HARVARD é desenvolvido em Java, há completa facilidade de processamento de ficheiros em XML devido às bibliotecas (classes) disponíveis. E considerando as características do XML, alterações em ficheiros XML não significam necessariamente alterações em código de programação Java. Pode-se, por exemplo, acrescentar novas "tags" como requisito de um novo módulo do HARVARD, ou simplesmente para melhor apresentar os resultados.

O utilizador pode, através de um simples editor ou de qualquer outra ferramenta de edição/visualização para XML, especificar o processo de análise de dados bastando para isso carregar um "template" de tarefa do HARVARD que define todos os atributos passíveis de especificação com seus respectivos parâmetros. Uma vez escrito o ficheiro de especificação de tarefas este é inserido como parâmetro de ativação do HARVARD.

Cabe destacar que os resultados do HARVARD por serem representados em XML, não necessitam de tratamento especial para apresentação ao utilizador, sendo visualizados em qualquer *browser* compatível.

3.5 A estrutura do HARVARD

3.5.1 O Servidor

O nó servidor “lê” o conjunto de tarefas a realizar, descritas como um conjunto de Unidades de Trabalho (UT). Para cada UT é associada uma máquina onde vai ser executada. O servidor atribui essa UT à máquina indicada. Quando uma UT termina, associa o resultado recebido à UT e devolve-o ao utilizador.

O *Servidor* é constituído por quatro módulos: o gestor de tarefas (GT), o escalonador (ESC), o gestor de recursos (GR) e o módulo de comunicação (COM).

Módulo Gestor de Tarefas (GT)

O módulo GT orienta todo o processo KDD representado através de um grafo de *workflow* com todas as tarefas pertinentes ao processo de análise de dados. As tarefas por sua vez são convertidas em uma ou mais UT (unidades de trabalho) que serão processadas pelo ambiente de forma independente. À medida que cada uma das tarefas do processo termina, os resultados são associados e armazenados no respectivo nó do grafo. Isso permite o acompanhamento do processo de análise de dados passo-a-passo pelo utilizador e facilita a consolidação final dos resultados para apresentação. O módulo GT interage com o módulo ESC provendo todas as especificações necessárias para a distribuição e execução das tarefas. Deve recolher os resultados e devolvê-los ao utilizador.

Módulo Escalonador (ESC)

O módulo ESC recebe do módulo GT uma lista com as tarefas que no momento podem ser executadas pelos clientes. As tarefas são especificadas em uma ou mais unidades de trabalho (UT) de acordo com o formato mostrado na Figura 3.4. Para cada UT, o escalonador solicita ao módulo de gestão de recursos (GR) uma máquina

adequada à execução da tarefa. O módulo GR devolve a especificação de uma máquina disponível e adequada para a tarefa ou a indicação de que não há máquinas disponíveis para essa tarefa. O escalonador compõe a mensagem de requisição da tarefa e entrega-a ao módulo de comunicação para ser enviada ao cliente escolhido. O escalonador recebe ainda mensagens que indicam a conclusão de tarefas.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<workunit>

  <identification> T1 </identification>
  <apllication>
    <urlapl>www.fe.up.pt/ilp/IndLog/indlog.tgz</urlapl>
    <script>www.fe.up.pt/ilp/IndLog/script-ilp.scp</script>
    <parameters>www.fe.up.pt/ilp/IndLog/parameters.txt
    </parameters>
  </apllication>
  <data>
    <dataset>kdd99</dataset>
    <DBserver>www.fe.up.pt/mysql</DBserver>
    <DB>kdd99</DB>
    <translationscript>toilp.scp</translationscript>
  </data>
  <requirements>
    <memory unit="Mb">100MB</memory>
    <processor>Pentium</processor>
    <harddisc unit="Mb">1000</harddisc>
  </requirements>
  <estimatedtime unit="s"> 30</estimatedtime>
  <results>
    <filename>kdd99.out.gz</filename>
    <DBserver>www.fe.up.pt/mysql</DBserver>
    <DB>kdd99</DB>
  </results>
</workunit>
```

Figura 3.4: Exemplo de especificação de uma tarefa em UT usando a linguagem XML.

Módulo Gestor de Recursos (GR)

Este módulo mantém informação sobre os recursos computacionais (máquinas e estado da rede²) de acordo com a especificação indicada na Figura 3.5, e que inclui: informação estática e informação dinâmica que é regularmente atualizada. Entre

²Informação ainda não usada na versão atual do HARVARD.

a informação de cada máquina está o estado dela: indisponível, disponível ou em utilização. Este módulo utiliza 3 filas (uma para cada estado possível). Este módulo recebe ainda pedidos do módulo escalonador no formato da especificação de uma máquina e devolve uma máquina com características mais próximas das restrições ao pedido ou a indicação de que não há máquinas disponíveis para esse pedido. O módulo GR recebe periodicamente mensagens de cada cliente indicando a carga de trabalho no cliente e quantos utilizadores estão a usar a máquina.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<machine>
  <identifier>
    <ip>0.0.0.0</ip>
    <hostname>tau4</hostname>
  </identifier>
  <hardware>
    <cpu>Intel</cpu>
    <clockspeed unit="Gh">1.6</clockspeed>
    <ram unit="Mbyte">50</ram>
    <hd unit="Mbyte">250</hd>
  </hardware>
  <opsystem>Linux</opsystem>
  <availability>
    <static> </static>
    <workload> </workload>
    <login> </login>
  </availability>
</machine>
```

Figura 3.5: Exemplo da codificação em XML de descrição da máquina cujo o *hostname* é “tau4”.

Módulo de Comunicações (COM)

O módulo de comunicação é o único ponto de interação direta entre Servidor e Cliente. Implementa processos de comunicação por *RMI*, *socket* ou utilizando o protocolo *HTTP*. Este módulo é igual para o servidor e clientes e está detalhado na secção a seguir onde são descritos os módulos do cliente.

3.5.2 Cliente

Um *Cliente* é constituído por quatro módulos: o módulo de monitorização do estado da máquina (MON), o módulo de execução de tarefas (TRAB), o módulo envolvente

do programa aplicação (WRAP) e o módulo de comunicação (COM).

Um *Cliente* recebe uma UT, realiza as operações especificadas nessa UT e devolve o resultado ao servidor. O *Cliente* realiza em simultâneo a monitorização do estado da máquina em que executa. É da responsabilidade do *Cliente* ativar e monitorizar o programa aplicação. Toda a comunicação entre o *Servidor* e *Cliente* é feita pelo módulo de comunicação.

Módulo Controlo de Tarefas ou Trabalhador (TRAB)

Este módulo interpreta as mensagens vindas do servidor e que são de três tipos: *execução de uma unidade de trabalho, terminação da tarefa actual ou terminação de toda a atividade do Cliente*. No caso de uma mensagem para execução de uma tarefa este módulo interpreta os campos da especificação da UT. Vai buscar o programa aplicação, e também buscar os dados e guarda-os localmente no diretório onde colocou a aplicação; lança a aplicação. A aplicação é carregada através de *HTTP* no servidor *Web* de aplicações. Os dados são carregados de uma BD usando *JDBC*. Na especificação da UT estão as *queries* SQL para acesso à BD e o *script* de execução da aplicação. O ficheiro resultado da execução da tarefa é colocado numa tabela da BD e comunicado ao servidor o fim da tarefa.

Módulo Envolvente da Aplicação (WRAP)

Este módulo lê e interpreta o *script*. Esta interpretação pode ser com ou sem interações. Se for com interações o módulo WRAP interpreta linha a linha do *script* de controlo da execução da tarefa, e coloca cada linha no *stdin* da aplicação e recolhe o seu *stdout* e *stderr*. Coloca o *stdout* num ficheiro como resultado da execução da tarefa. Este módulo descomprime automaticamente ficheiros com extensões *tgz*, *tar*, *gz*, *zip* e *rar* antes de eles serem utilizados na execução da tarefa.

Módulo de Monitorização de Estado dos Recursos (MON)

Este módulo verifica o número de utilizadores e carga da máquina, onde o *Cliente* executa e comunica regularmente essa informação ao *Servidor*. A ausência da recepção destas mensagens regulares permite ao *Servidor* saber quando o *Cliente* não está a executar.

Módulo de Comunicações (COM)

A comunicação é feita de acordo com o destinatário e a natureza da mensagem. O tipo de protocolo está indicado num campo XML da mensagem. As mensagens trocadas entre clientes e servidor utilizam RMI ou *sockets*. Existe mais um tipo de mensagem que requer o uso do HTTP e é utilizada para trazer a aplicação do servidor Web de aplicações para o local do cliente. Um último tipo de mensagens indica a utilização de JDBC para trazer os dados a serem processados pela aplicação. O módulo de comunicações é constituído por quatro sub-módulos e duas filas de mensagens: uma de mensagens a enviar e outra de mensagens recebidas. Cada sub-módulo é implementado por *threads* independentes que consultam filas de mensagens, identificam, e processam as mensagens que lhes competem. Cada um destes sub-módulos implementa um protocolo: sub-módulo RMI; sub-módulo sockets; sub-módulo HTTP e; sub-módulo JDBC. Os sub-módulos HTTP e JDBC respondem a mensagens dirigidas especificamente ao módulo de comunicações e são utilizados para buscar aplicações e dados, respectivamente.

3.5.3 Funcionamento guiado por eventos

O HARVARD é concebido num esquema *event-driven* e passagem de mensagens entre os diferentes módulos. Na prática isso resulta numa grande versatilidade do sistema visto que todos os módulos, sejam do *Servidor* ou do *Cliente* do HARVARD, recebem repetidamente informação para processar e disparam uma função de resposta de acordo com o evento recebido. Esta característica é muito útil quando aplicada num

ambiente onde as estações configuradas para operar como nó cliente do HARVARD são na verdade equipamentos de uso comum aos utilizadores do mesmo ambiente organizacional. Isso significa que toda a vez que o utilizador necessitar operar uma estação, que tenha um módulo cliente do HARVARD em execução, haverá necessariamente a geração de um evento de "interrupção" ou "pausa" a partir do módulo de Monitorização de Estado dos Recursos (MON) e o envio de uma mensagem ao módulo TRAB informando sobre a situação do recurso em utilização. Por sua vez, o módulo Controle de Tarefas (TRAB), ainda na estação cliente, encaminha mensagem ao Módulo Escalonador (ESC) do Servidor, informando que a tarefa está "pendente", cabendo ao módulo ESC a decisão sobre as futuras ações a serem tomadas em relação à UT inicialmente encaminhada para processamento no módulo cliente. Para cada módulo do HARVARD há um conjunto pré-definido de eventos possíveis com as respectivas mensagens e ações a serem tratadas. O funcionamento dos módulos por eventos tem ainda como vantagem uma carga reduzida no CPU da máquina local. Só é usado CPU quando de fato é necessário, isto é, quando chega um evento que necessita de processamento. Logo após o tratamento do evento, o *thread* liberta o CPU até que apareça novo evento.

3.6 Extensões ao Sistema HARVARD

3.6.1 Macro Tarefas

Existem inúmeras situações no processo de KDD envolvendo tarefas constituídas por sub-tarefas independentes que podem ser executadas em paralelo. Exemplos destas situações encontram-se desde as fases iniciais de pré-processamento até à fase de *Data Mining* (aplicação de um algoritmo de análise de dados). Incluem nesta categoria as operações frequentes de escolhas dos melhores atributos (*feature subset*

selection), sintonização de parâmetros, validação cruzada, execução dos algoritmos numa abordagem de Ensemble, etc. Estas oportunidades de paralelizar/distribuir a execução podem ainda ocorrer em situações menos frequentes mas não menos interessantes/importantes. Alguns algoritmos, por exemplo, têm um carácter estocástico. Para estes uma possibilidade seria efetuar várias execuções em paralelo e reportar resultados médios. Sistemas em que a semente inicial, por exemplo, determina a solução podem ser lançados em paralelo, por meio de várias execuções com sementes diferentes. Ainda algoritmos cuja solução seja dependente da ordem de processamento de exemplos podem usufruir de computação distribuída em que se faz *N shufflings* aos dados antes de executar o algoritmo e se apresentam no final as médias dos resultados individuais. Algoritmos de *clustering* como o *k-means* em que o utilizador tem que especificar o valor de *K* pode ser automatizado executando em paralelo o mesmo algoritmo com vários valores de *k* e depois num nó final escolhemos o melhor *k* usando uma medida de avaliação de clusters como por exemplo a *silhouette*.

Embora estas situações, acabadas de referir, tirem facilmente vantagem de um sistema distribuído como o HARVARD em alguns casos o utilizador pode ter bastante trabalho a especificar todos as sub-tarefas em XML necessárias para distribuição da computação. Para agilizar esta tarefa de especificação do workflow de computação foi desenvolvido e incluído no HARVARD o conceito de **macro tarefa** que definiremos em seguida.

Uma macro tarefa é uma tarefa com um nome reservado e que aceita um conjunto de parâmetros de acordo com a sua utilização. As macro tarefas são expandidas num passo de pré-processamento antes de o sistema ler o ficheiro de workflow das tarefas que o utilizador especificou. Associado a cada macro tarefa está um conjunto de programas e scripts que geram o grafo de tarefas necessário à implementação da macro tarefa. Este grafo pode conter um número grande de nós (tarefas a serem diretamente executadas pelo HARVARD). Por exemplo, numa macro tarefa de vali-

dação cruzada, o pré-processamento recebe o conjunto de dados original e o número CV de partições (*folds*) pretendido e gera CV+1 tarefas básicas do HARVARD em que CV delas contêm cada uma um par treino/teste típico da validação cruzada bem como o script para executar o algoritmo nesse par. Estas CV tarefas são executadas em paralelo. E tem uma última tarefa que espera pelo término de todas as tarefas iniciais e utiliza os resultados da execução delas para calcular o resultado final da validação cruzada.

Este esquema permite ainda uma extensão fácil do número de macro tarefas disponibilizadas pelo HARVARD. Para tal é preciso que o investigador responsável pelo HARVARD desenvolva os scripts necessários à super tarefa, defina a sua sintaxe e atualize o pré-processador.

As situações referidas no início desta sub-secção podem facilmente ser implementadas como macro tarefas.

Nesta secção descrevemos de modo abstrato algumas das macro tarefas implementadas. Exemplos concretos de implementação destas macro tarefas para vários tipos/classes de algoritmos são ilustradas no Capítulo 5.

Cross validation - uma das macro tarefas implementadas é a validação cruzada. Os scripts que implementam estabelecem as N partições requeridas pelo utilizador e produzem N tarefas HARVARD que vão executar em paralelo correndo o algoritmo nos dados de cada partição. Após esse N nós existe uma barrier para que no nó N+1 que se segue este possa recolher os resultados de todas as N execuções anteriores e produzir o resultado.

Feature subset selection - a versão de seleção dos melhores atributos implementada realiza uma procura primeiro-em-largura começando com todos os atributos e terminando no limite de profundidade especificado pelo utilizador. Na implementação desta estratégia são primeiro calculadas, nível a nível, todas as combinações de atributos a remover. E no final desta contagem são produzidas tarefas HARVARD

um número igual ao número total de combinações mais uma tarefa final de recolha do resultado. Estas tarefas (menos a final) são executadas em paralelo e existe uma barrier para que a tarefa final espere pela conclusão de todas as anteriores para produzir o resultado final.

Sintonização de parâmetros - esta macro tarefa recebe o nome do algoritmo a avaliar e um ficheiro com os valores a experimentar para cada parâmetro do algoritmo. Num passo inicial calcula todas as combinações desses parâmetros e de modo semelhante às macro tarefas anteriores cada combinação dá lugar a uma tarefa HARVARD que vai executar em paralelo com todas as restantes combinações. Existe também, uma barrier que antecede a tarefa final de cálculo do resultado global.

3.6.2 HARVARD e o ambiente *Grid Computing*

Para a integração do sistema HARVARD com ambientes de processamento em Grid, criamos uma extensão à estrutura básica do próprio sistema. "-g" é a versão do HARVARD com a ligação a *Grid*. A conexão com um ambiente *Grid* permite estender as capacidades e funcionalidades do sistema HARVARD com a utilização de recursos computacionais adicionais. A vantagem deste modelo, ilustrado na Figura 3.6, é distribuir tarefas além do ambiente local do HARVARD. Algumas tarefas podem ter alguma restrição, seja pela complexidade ou mesmo pelo acesso a determinados dados remotos; e assim não podem ser executadas no ambiente local. A integração com um ambiente remoto *Grid* permite executar tarefas KDD usando um conjunto muito vasto de recursos computacionais. Entretanto, caberá ao HARVARD a consolidação dos resultados para apresentação ao utilizador. O importante neste modelo é agregar maior poder de processamento a partir da integração do HARVARD com ambientes *Grid Computing*.

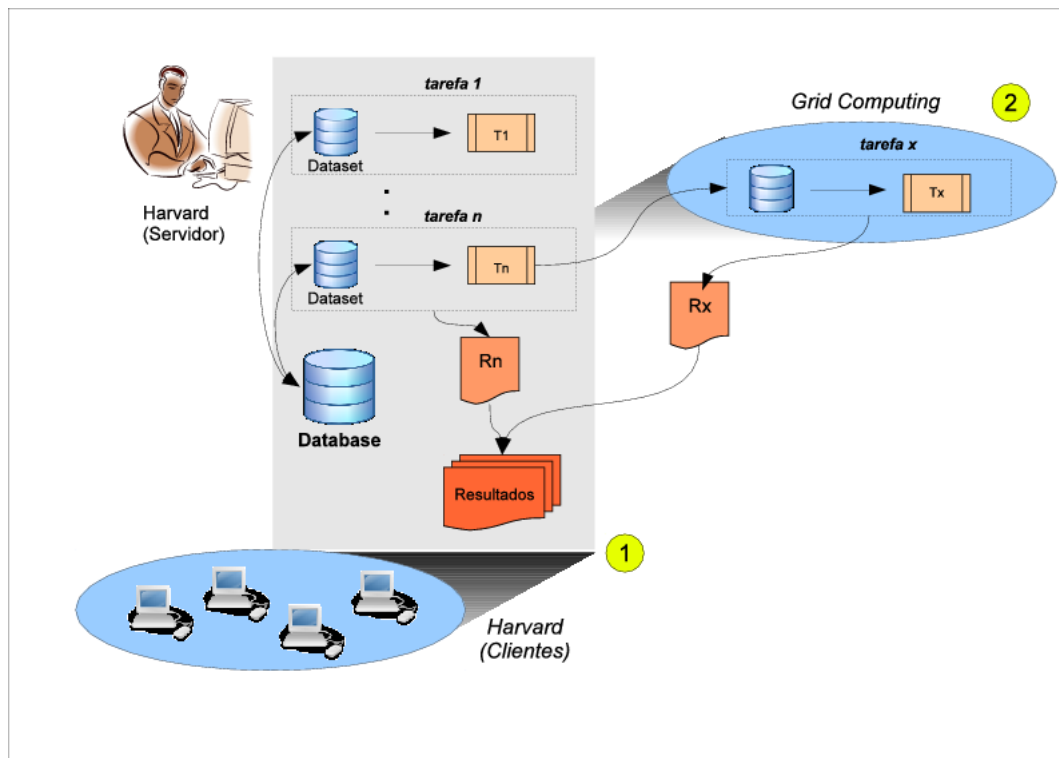


Figura 3.6: Arquitetura HARVARD-g: (1) processamento local do HARVARD; (2) execução de tarefas em ambiente *Grid Computing* remoto

Módulo de Conexão ao *Grid* na estrutura HARVARD

Este módulo é agregado à estrutura do Sistema HARVARD apresentada na Secção 3.5. O módulo de Conexão ao Grid (CGrid) agrega, para além das funcionalidades básicas da implementação *Grid* compatível com o HARVARD (discutidas na Secção 3.6.2), as funcionalidades de tratamento das UT (Unidades de Trabalho) a serem encaminhadas para processamento em *Grid*. O módulo CGrid está baseado inicialmente na mesma máquina que hospeda o Servidor do HARVARD e tem interação com todos os demais módulos. Conforme a Figura 3.7 mostra, o módulo CGrid comporta-se de forma similar a um nó cliente e também tem alguns sub-módulos seja para monitorizar o estado da Grid (MONg), acompanhar a execução das tarefas recebidas do servidor (TRABg), monitorizar o estado dos recursos da *Grid* (GRg) e intermediar a comunicação entre o *Grid* (COMg) e o HARVARD. O CGrid interpreta as mensagens vindas do servidor e que são de três tipos: *execução de uma unidade de trabalho, terminação da tarefa atual ou terminação de toda a atividade junto ao Grid*. No caso de uma mensagem para execução de uma tarefa este módulo interpreta os campos da especificação da UT, codifica na forma compatível com a implementação Grid e submete a UT para processamento. A UT codificada na forma compatível do Grid deve especificar previamente a aplicação a executar. Caso o Grid não disponha da aplicação a executar, vai buscar o programa aplicação num repositório de algoritmos *Data Mining* implementados em *Java*, e também buscar os dados em uma Base de Dados usando JDBC. O resultado da execução da tarefa no ambiente Grid é colocado numa tabela da BD e comunicado ao servidor e aos demais sub-módulos do CGrid a conclusão da tarefa.

Implementações *Grid*

Neste caso, desenvolvemos conexão com sistemas compatíveis com o *Globus Toolkit 4.0 (GT4)* que dispõe de ferramentas de conectividade, entre elas o *Commodity Grid (CoG) Kit* [vLFGL01] [vLH05] que facilita a interconexão de ambientes heterogê-

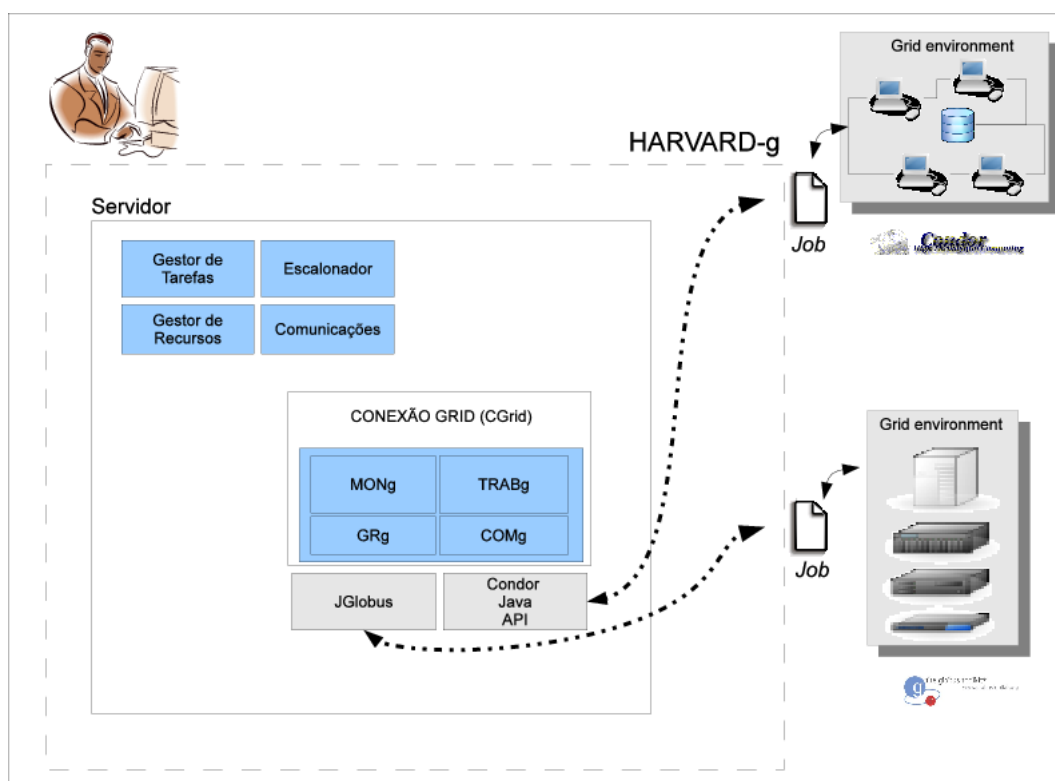


Figura 3.7: Módulo Conexão Grid (CGrid) e submódulos

neos, neste caso particular utilizamos a biblioteca compatível com Java chamada de *jGlobus*. A fim de estender o uso a outros ambientes *Grid Computing* também incluímos a versão HARVARD-g a conexão ao *Condor* [FTL⁺01, TWML01, TTL05] com a utilização da biblioteca Condor-API-Java³. A biblioteca Condor-API-Java permite submeter, monitorizar e controlar tarefas submetidas um *Grid Computing* baseado em *Condor* a partir de uma interface Java. Como o HARVARD é todo desenvolvido em Java a adaptação torna-se facilitada a partir do uso dessa API.

As bibliotecas *jGlobus* e *Condor-API-Java* fazem parte da implementação do módulo CGrid conforme o caso da plataforma *Grid* a ser utilizada.

Conexão ao *Globus* O Sistema HARVARD-g, na versão de conexão Globus, conforme representado na Figura 3.8, inclui conexão com ambientes *Grid Computing* configurados para a plataforma *Globus Toolkit* (versão GT-4) [Fos05]. A biblioteca *jGlobus* para aplicações em Java contém uma API básica que permite o acesso remoto de dados (gridFTP), a submissão e monitorização de tarefas (GRAM), a implementação de recursos de segurança (GSI), e ainda, inclui o cliente *myProxy* (certificate store).

O *CoG Kit na versão Java* facilita a integração de programas escritos em *Java* com o *Globus Toolkit*. Como o HARVARD é escrito em Java, para o HARVARD-g foi adicionado um módulo extra chamado de *Conexão-Grid*, junto ao nó *Mestre* do HARVARD. Este módulo utiliza basicamente a biblioteca (API) *jGlobus* [vLFGL01], e permite que uma tarefa originalmente definida para o HARVARD seja adaptada aos requisitos de submissão de tarefas num ambiente *Grid Computing*.

O módulo *Conexão Grid* implementa as seguintes classes:

- (a) *harvard.grid.autentica*: sistema de autenticação e uso dos recursos do ambiente Grid, com uso das classes *org.globus.myproxy* e *org.globus.gsi*,
- (b) *harvard.grid.recursos*: especificação dos recursos a serem utilizados ("stdin",

³<http://code.google.com/p/condor-java-api/>

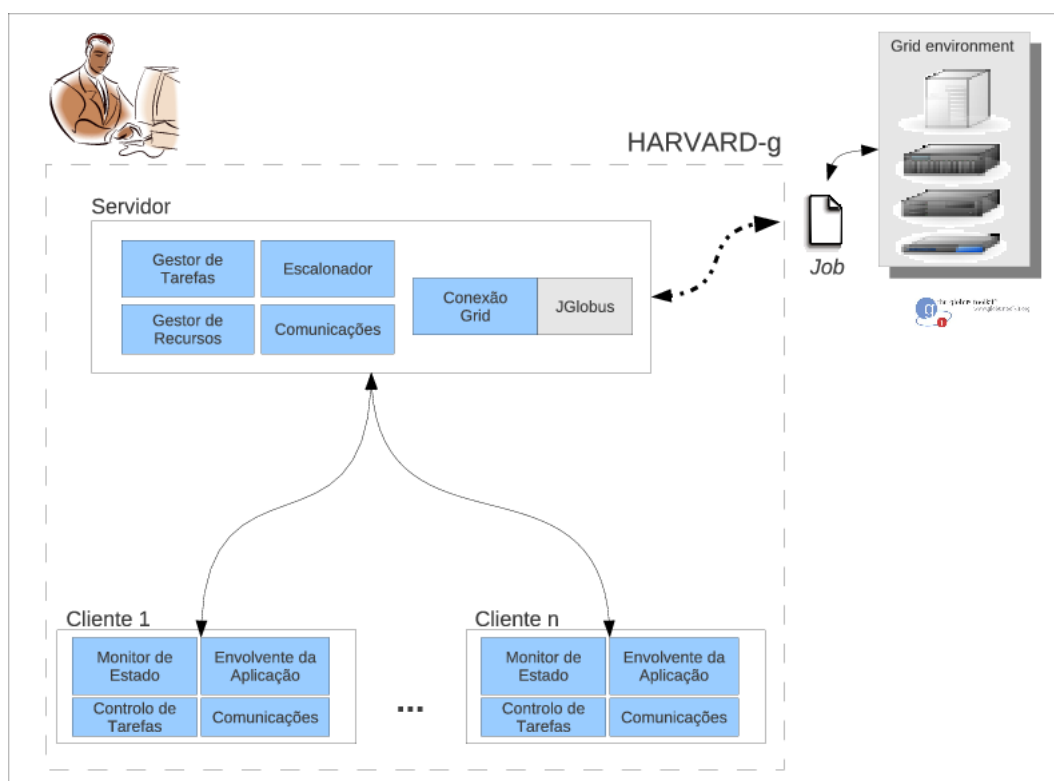


Figura 3.8: Representação dos módulos do HARVARD-g (versão Globus)

"stdout", "stderr", programas a executar e diretório de trabalho), com uso da classe *org.globus.rsl*,

(c) *harvard.grid.ftp*: transferência de ficheiros de dados e/ou programas a executar, com uso da classe *org.globus.ftp*, e

(d) *harvard.grid.tarefa*: submissão de tarefas no ambiente Grid, com uso da classe *org.globus.gram*.

Os resultados de submissão de tarefas num ambiente Grid são encaminhados ao Módulo Escalonador com status “finalizado”. O módulo Conexão Grid utiliza o certificado padrão x.509 do utilizador para permitir a submissão de tarefas ao ambiente Grid na plataforma *Globus*.

Como o GT4, uma vez adequadamente configurado, aceita programas escritos em linguagem JAVA, o HARVARD-g encaminha previamente os algoritmos *Data Mining* codificados em “.jar” necessários a execução de tarefas KDD. Como exemplo, temos utilizado os algoritmos codificados para o ambiente WEKA [Han01, SR08, KFR08, WF05] e Yale [MWK⁺06]. Assim, uma vez escolhido o algoritmo, este é transferido e executado no ambiente *Grid Computing* previamente configurado para operar com Globus, de forma transparente ao utilizador.

Para além do requisito dos algoritmos Data Mining estarem codificados em Java, o utilizador do sistema HARVARD deve ter necessariamente um certificado x.509 [CSF⁺08] válido e aceito no ambiente GT4 conectado ao HARVARD-g.

Conexão ao *Condor* O sistema HARVARD-g, na versão de conexão ao *Condor*, conforme representado na Figura 3.9, inclui conexão com ambientes *Grid Computing* que utilizam-se da arquitetura e solução *Condor* [TTL05].

A API Condor-Java viabiliza a integração de programas escritos em *Java* com um ambiente *Grid Computing* baseado em *Condor*. Nesta versão de integração ao *Condor* foi adicionado um módulo extra chamado de *Conexão-Grid-Condor*, junto ao nó *Mestre* do HARVARD. Este módulo é composto das classes, conforme ilustrado na Figura 3.10, da biblioteca Condor-API-Java, e permite que uma tarefa

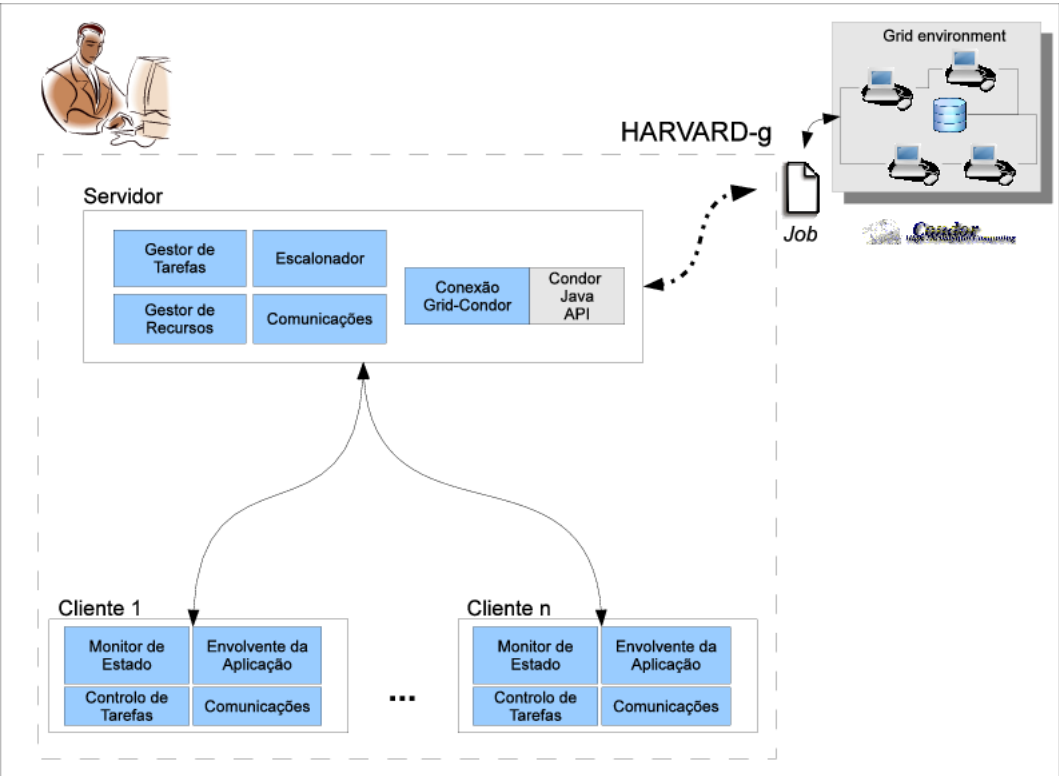


Figura 3.9: Representação dos módulos do HARVARD-g (versão Condor)

Package condorAPI

Interface Summary	
EventType	Condor Event Type constants
Handler	Interface for event handlers.
Status	Condor Job Status constants

Class Summary	
Cluster	stands for the job cluster
Condor	This class abstracts the condor system
Event	Event object
Job	this class represent a Job.
JobDescription	This class stands for a submit file for condor.
JobId	
Node	this class meant to represents a node in parallel jobs.

Exception Summary	
CondorException	

Figura 3.10: Classes disponíveis na Condor-API-Java, descrição original

```

import harvard.*
import condorAPI.Cluster;
import condorAPI.Condor;
import condorAPI.CondorException;
import condorAPI.Event;
import condorAPI.Handler;
import condorAPI.JobDescription;

public class harvard.gridCondor.Tarefa {

    /**
     * @param args
     */
    public static void main(String[] args) throws CondorException{
        Condor.setDebug(true);
        Condor condor = new Condor();

        JobDescription jd2 = new JobDescription();
        jd2.addAttribute("executable", "/harvard/data");
        jd2.addAttribute("universe", "Java");
        jd2.addQueue();
        jd2.setHandlerOnSuccess(new Handler(){
            public void handle(Event e){
                System.err.println("success " + e);
            }
        });

        Cluster c2 = condor.submit(jd2);
        System.out.println("submitted");
        c2.waitFor();
        System.out.println("done");

        ...
    }
}

```

Figura 3.11: Exemplo de utilização da Condor-API-Java

originalmente definida para o HARVARD seja adaptada aos requisitos de submissão aum ambiente *Condor*.

Neste caso particular, o módulo *Conexão-Grid* versão *Condor* implementa basicamente uma classe:

(a) *harvard.gridcondor.tarefa*: este módulo trata da submissão de uma tarefa proveniente do HARVARD para os requisitos de execução no ambiente Condor com uso das classes *condorAPI.Cluster*, *condorAPI.Job* e *condorAPI.JobDescription*. Esta classe submete uma tarefa inicialmente definida para execução no HARVARD, conforme ilustrado na Figura 3.11 . Os resultados de submissão de tarefas ao ambiente Condor são encaminhados ao Módulo Escalonador com status “finalizado”.

O uso do Condor permite estender a aplicabilidade do HARVARD para uma intranet, onde os recursos estão sob um único domínio, o que certamente facilita a configuração quanto aos requisitos de processamento do HARVARD. No caso de uso

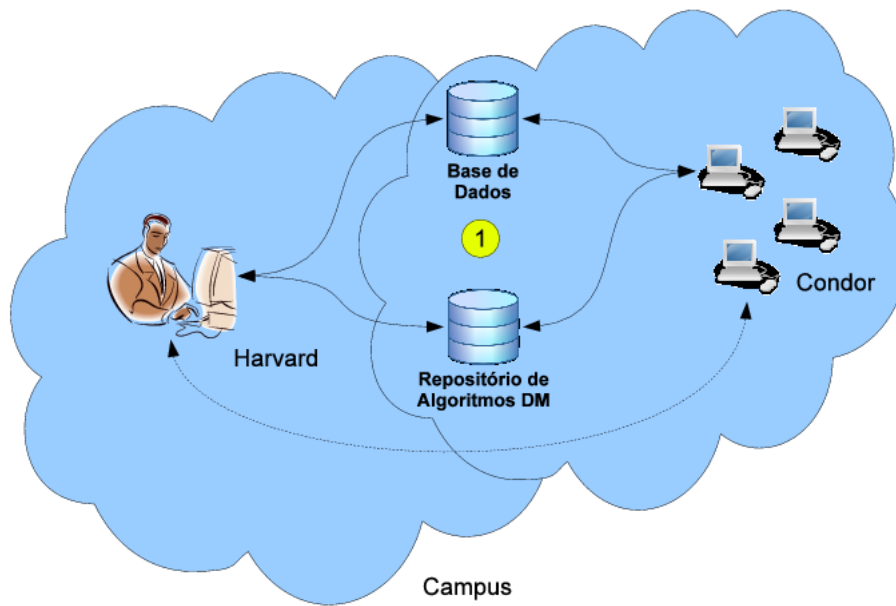


Figura 3.12: Diagrama de compartilhamento de recursos: (1) Base de Dados e o Repositório de Algoritmos DM acessíveis para o HARVARD e o Condor

do HARVARD combinado ao Condor alguns requisitos devem ser estabelecidos, entre os quais a prévia configuração do ambiente Condor no campus ou intranet onde o HARVARD será processado. Isso inclui, entre outros requisitos, a configuração do Condor para prévio acesso a um repositório contendo os algoritmos Data Mining e os dados a serem processados conforme ilustrado na Figura 3.12 .

3.7 Conclusões

O HARVARD é um ambiente computacional distribuído capaz de extrair conhecimento de grandes quantidades de dados utilizando técnicas de *Data Mining* com uso de algoritmos de *Aprendizagem Computacional*.

É uma plataforma modular de baixo custo pois utiliza recursos computacionais

disponíveis e ociosos de uma organização, e portanto não perturba o normal uso dos recursos na rotina diária. É uma plataforma flexível e versátil pois utiliza diferentes ferramentas pré-existentes de análise de dados sem re-programações ou adaptações, sendo independente quanto à ferramenta requerida para o processo de KDD.

Neste capítulo descrevemos a arquitetura e estrutura dos módulos (*Cliente e Servidor*), e também os respectivos sub-módulos responsáveis pelo funcionamento do HARVARD para tratar tarefas de *Data Mining*. Descrevemos ainda algumas características operacionais de funcionamento do HARVARD para aceder os dados e algoritmos de DM em BD, e do próprio uso da linguagem XML para formatação das mensagens trocadas entre os diferentes módulos do sistema.

Descrevemos ainda um conjunto de extensões à arquitetura básica do HARVARD de modo a aumentar a capacidade de computação e a facilitar a especificação das tarefas de KDD com recurso a macro tarefas.

O desenvolvimento de um módulo de interface com uma Grid permite alargar de forma substancial os recursos computacionais disponíveis. As Grids, em geral integram um elevado número de máquinas com recursos (memória RAM, arquitetura de CP, etc) diversificados. Este número de máquinas e a diversidade permitem processar em menos tempo mais tarefas e aumentam a probabilidade de satisfazer um leque mais vasto de restrições das tarefas.

O módulo de ligação ao Condor permite alargar o número de máquinas e, utilizar de modo fácil, uma infra-estrutura pré-existente.

A implementação de macro tarefas poupa muito tempo ao utilizador, que caso contrário, teria de especificar cada uma das muitas tarefas que constituem a macro tarefa. Esta poupança é notória, como se verá no Capítulo 5, onde tarefas com *feature subset selection* e sintonização de parâmetros podem envolver um grande número de "sub-tarefas" em paralelo que, no caso de não estarem disponíveis macro tarefas, seriam especificadas pelo utilizador.

Capítulo 4

Contribuição para Data Mining Relacional

Como foi referido anteriormente, o ILP é uma abordagem de Aprendizagem Computacional utilizada em *Data Mining* Multi Relacional. Algumas das suas vantagens em relação principalmente a algoritmos de Aprendizagem Computacional proposicionais incluem: i) elevado poder expressivo que permite a construção de modelos complexos; ii) a possibilidade de usar com facilidade conhecimento prévio (*background knowledge*) relevante para o processo de construção dos modelos; iii) compreensibilidade dos modelos construídos; e iv) codificação fácil de dados com estrutura. Estas características são importantes para aplicações tanto industriais como científicas que requeiram uma complexa análise a partir de dados com estrutura.

No entanto, a eficiência dos sistemas de ILP é uma das suas grandes limitações na aplicação a problemas complexos ou com grande número de exemplos. Como a maioria dos sistemas MRDM (*Multi Relational Data Mining*), um sistema ILP efetua uma procura num vastíssimo espaço de hipóteses. Controlar o tempo de execução é, portanto, uma questão fundamental e tornou-se cada vez mais importante à medida que aumenta o número de exemplos a analisar. Como consequência e para o caso das execuções sequenciais, os investigadores de ILP têm que reduzir a

busca usando procuras estocásticas (eficiência conseguida à custa da qualidade dos modelos), restrições à linguagem das hipóteses e limites explícitos ao número de hipóteses construídas. Uma alternativa a estas abordagens sequenciais para tornar os sistemas de ILP mais rápidos é recorrer a técnicas de paralelismo [Fon06]. A execução paralela de sistemas de ILP é uma abordagem muito promissora para superar a limitação da eficiência. Pode, não só reduzir substancialmente o tempo de execução, mas também ser usada para melhorar a qualidade dos modelos construídos. Acresce ainda que o paralelismo para execução de tarefas é hoje uma escolha possível, dada a ampla disponibilidade e o baixo custo das plataformas paralelas (clusters).

Várias contribuições têm sido dadas para a execução paralela de sistemas de ILP. Veja-se os exemplos referidos no Capítulo 2. No entanto, grande parte dessas contribuições descrevem algoritmos que são executados em máquinas de memória partilhada ou em ambientes em que a comunicação entre os vários processos é relativamente rápida. Isto é admissível quando há uma considerável quantidade de comunicação entre os processos que executam em paralelo. Estas implementações, no entanto, não são adequadas para execução em ambientes como o Grid, em que as sub-tarefas devem ser completamente independentes.

Propomos por isso um novo algoritmo que tem um conjunto de características que permitem a sua execução tanto em ambientes de memória partilhada (*shared memory*) como em ambientes de computação distribuída tipo Grid. A principal preocupação no seu desenvolvimento foi definir um conjunto de sub-tarefas que sejam processadas de forma completamente independente.

4.1 Uma nova abordagem para uma execução paralela de ILP

Em Programação em Lógica (PL), e em particular na abordagem de paralelismo-E (AND-paralelism), tira-se partido das partições efetuadas numa cláusula em que

nenhum dos literais de cada partição partilha qualquer variável com qualquer outro literal de outra partição. Podem no entanto partilhar variáveis com o literal da cabeça da cláusula. Veja-se este exemplo¹:

$$C = h(X, Y) \leftarrow l1(X, X1), l2(X1, X2), l3(X1, X3), l4(Y, Y1), l5(Y1, Y2)$$

Consideremos a seguinte partição $\{\{l1, l2, l3\}, \{l4, l5\}\}$. Em paralelismo-E cada subconjunto da partição pode ser executado em paralelo pois a cláusula C só sucede se sucederem ambos os subconjuntos, e falha se falhar pelo menos um deles. Os valores usados na unificação de variáveis num subconjunto nunca podem alterar a computação no outro subconjunto.

O algoritmo que propomos é uma generalização deste resultado conhecido em paralelismo-E e a sua adaptação para o contexto de ILP.

Com base no resultado de LP em que literais que não partilham variáveis podem ser executados em "nós" independentes, transpomos esse resultado para o sistema ILP dizendo que uma cláusula com dois subconjuntos de literais que só partilham variáveis com a cabeça da cláusula a sua cobertura pode ser calculada com mais eficiência. Se avaliarmos os dois subconjuntos resultantes em separado podemos combinar a cobertura da cláusula completa fazendo a intersecção das listas de cobertura das suas "sub-cláusulas" e a intersecção de duas cláusulas de componentes tratadas de forma independente.

Consideremos o exemplo com duas cláusulas:

$$C_1 = \text{ativa}(\text{Farmaco}) \leftarrow \text{atomo}(\text{Farmaco}, A), \text{pesoAtómico}(A, 12).$$

com cobertura positiva $[1-2, 10-20]^2$ (cobre os exemplos 1, 2 e todos entre 10 e 20 inclusive e cobertura negativa $[1-40]$ e

$$C_2 = \text{ativa}(\text{Farmaco}) \leftarrow \text{logp}(\text{Farmaco}, \text{LogP}), \text{menorQue}(\text{LogP}, 0.2).$$

¹A vírgula representa a conjunção.

²codificação usada pelo Aleph

com cobertura positiva [2-2,15-25] e cobertura negativa [10-60]. Se definimos C_3 pela combinação de C_1 e C_2 obtemos:

$C_3 = \text{ativa}(\text{Farmaco}) \leftarrow \text{atomo}(\text{Farmaco}, A), \text{pesoAtomico}(A, 12), \text{logp}(\text{Farmaco}, \text{LogP}), \text{menorQue}(\text{LogP}, 0.2)$

com cobertura positiva [2-2,15-20] e cobertura negativa [10-40]

A cobertura de C_3 pode ser obtida pela intersecção das coberturas positivas e negativas de C_1 e C_2 .

Isto permite uma maior eficiência quanto à avaliação de cláusulas, que é a parte mais dispendiosa de um sistema ILP. O sistema pode gerar separadamente (em paralelo) cláusulas que não partilham variáveis entre si e, em seguida, calcular novas cláusulas combinando as previamente geradas. A cobertura de uma cláusula posterior poderá ser calculada pela intersecção das listas de cobertura das cláusulas iniciais e, portanto, sem usar chamadas ao Prolog para derivar os exemplos. Uma vez que é demorado analisar em cada cláusula a identificação desses grupos independentes de literais em runtime, precisamos de um processo eficiente. Em vez de analisar os literais, analisamos as declarações de modo. Declarações de modo que têm informações sobre os tipos de argumentos e, portanto, se dois conjuntos de literais não compartilham entre eles qualquer tipo de argumento, então eles não podem ter variáveis comuns e, portanto, eles podem ser considerados como duas ilhas separadas. Isto é a base para o algoritmo proposto.

Por simplicidade chamamos *ilha* a um conjunto de declarações de modo que partilham tipos entre si e não partilham tipos com outras ilhas. O conjunto de declarações de modo de um conjunto de dados é dividido em ilhas em que uma declaração de modo de um ilha não partilha tipos com qualquer outra declaração de outra ilha. Isso garante que os literais gerados usando declarações de modo de ilhas diferentes não partilhem variáveis.

Resultados para esta implementação inicial parecem muito promissores. O ponto-chave para uma execução distribuída é que cada "nó" (ilha) funciona como um sistema MDIE (*Mode-Directed Inverse Entailment*) completamente independente, que permite a saturação/redução da operação de busca em menor tempo. Não há necessidade de comunicação entre "nós" ou "ilhas" de processamento.

Uma possível explicação para esta abordagem produzir *speedups* em relação à execução sequencial reside no fato de na execução sequencial todas as cláusulas são avaliadas usando um demonstrador de teoremas (o *Prolog engine*) para calcular quais os exemplos (positivos e negativos) que a nova cláusula (a hipótese) cobre. Este processo é o ponto mais oneroso em termos computacionais do ciclo principal de um sistema de ILP. Com a divisão em ilhas, só as cláusulas construídas com as declarações de modo de cada ilha são avaliadas usando o processo "tradicional" de derivação dos exemplos. A cobertura de todas as cláusulas que resultem da combinação de cláusula entre ilhas diferentes são eficientemente calculadas pela operação de intersecção das listas de cobertura. Quanto maior for o número deste último tipo de cláusulas maior será o *speedup* em relação a uma execução sequencial.

Como exemplo para a implementação do algoritmo, desenvolvemos uma etapa de pré-processamento que calcula o conjunto de declarações de modo e de determinação para cada ilha uma única vez para cada conjunto de dados e então podemos realizar qualquer análise utilizando um algoritmo paralelo como o Algoritmo 1.

A vantagem do Algoritmo 1 é que o espaço de busca de toda a execução sequencial é particionada entre as *ilhas*, o que vai exigir menos requisitos de memória para fazer a computação.

O pseudo-código do Algoritmo 1 efetua um ciclo (ciclo *while* da função *CalculaIlhas()*) em que escolhe como semente uma declaração que não tenha tipos de entrada e chama a função *CompletaIlha()* para recolher todas as declarações que tenham tipos comuns a qualquer declaração que esteja na ilha em construção. O

Algoritmo 1 Calcula as *ilhas* a partir das declarações de modo

```
function CALCULAILHAS(TodosOsModos)
  ConjuntoDeIlhas  $\leftarrow \emptyset$ 
  Modos  $\leftarrow$  removeTiposEntradaCabeca(TodosOsModos)  $\triangleright$  passo de
  pré-processamento
  while Modos  $\neq \emptyset$  do  $\triangleright$  processa todos os modos
    Modo  $\leftarrow$  semTipoDeEntrada(Modos)
    Modos  $\leftarrow$  Modos  $\setminus \{ \text{Modo} \}$ 
    Ilha  $\leftarrow$  CompletaIlha( $\{\text{Modo}\}$ , Modos)
    ConjuntoDeIlhas  $\leftarrow$  ConjuntoDeIlhas  $\cup \{ \text{Ilha} \}$ 
  end while
  return ConjuntoDeIlhas
end function

function COMPLETAILHA(Ilha, Modos)
  repeat
    Modo  $\leftarrow$  ConectaSemIODeTipo(Modos)  $\triangleright$  retorna  $\emptyset$  se nenhum foi
    encontrado
    Modos  $\leftarrow$  Modos  $\setminus \{ \text{Modo} \}$ 
    Ilha  $\leftarrow$  Ilha  $\cup \{ \text{Modo} \}$ 
  until Modo =  $\emptyset$ 
  return Ilha  $\triangleright$  Ilha com conjunto de modos
end function
```

ciclo termina quando todas as declarações forem utilizadas. A função `CompletaIlha()` recebe uma declaração semente e vai acrescentando declarações até que não haja declarações que partilhem tipos com as declarações do conjunto em construção.

O pseudo-código do Algoritmo 2 implementa um algoritmo tipo Cobertura Gananciosa³. Basicamente, o algoritmo executa um ciclo principal até não haver exemplos positivos cobertos. Em cada iteração deste ciclo é construída a cláusula que cobre o maior número de exemplos positivos (ainda não cobertos na altura). Antes de passar a uma nova iteração, os exemplos positivos cobertos pela cláusula construída são removidos da lista atual de exemplos positivos, a cláusula é adicionada ao modelo e o ciclo repete-se (até não haver mais exemplos positivos para cobrir). A busca pela cláusula com maior cobertura é realizada evocando todas as ilhas para efetuarem a sequência tradicional de saturação/redução (característica dos sistemas MDIE em que é feita a busca da melhor cláusula no espaço de hipóteses). No final de cada ciclo principal as ilhas atualizam também a lista dos exemplos positivos ainda não cobertos.

4.2 Implementação rápida

Outra vantagem do Algoritmo 1 é que pode ser fácil e rapidamente implementado em sistemas MDIE existentes como Progol [Mug95], Aleph [Sri03] ou IndLog [Cam04]. Com poucas alterações, é possível re-utilizar estas implementações para reduzir o ciclo de processamento das cláusulas, reduzindo os passos que visem atingir a saturação/redução dos objetivos. Para que a implementação seja possível, é necessário um sistema de ficheiros para armazenar (em um ficheiro, por exemplo) todas as cláusulas úteis mas ainda inconsistentes e ainda considerar a melhor e mais consistente cláusula que tenha sido encontrada quando da saturação/redução do processo de

³Tradução de Greedy Cover Induction algorithm

Algoritmo 2 Algoritmo de Indução do tipo Cobertura Gananciosa

```
function INDUZTEORIA(ConjuntoDeDados, Clientes)
  Ilhas  $\leftarrow$  CalculaIlhas(GetModes(ConjuntoDeDados))     $\triangleright$  GetMode() retorna
  uma lista de declarações de modos definida para o conjunto de dados
  Teoria  $\leftarrow \emptyset$ 
  Exemplos  $\leftarrow$  ExemplosPositivos(ConjuntoDeDados)     $\triangleright$  conjunto inicial de
  exemplos positivos
  broadCast(Clientes, loadIslandsConjuntoDeDados)     $\triangleright$  cada cliente carrega os
  dados sem os modos
  while Exemplos  $\neq \emptyset$  do                                 $\triangleright$  cobertura de todos os exemplos positivos
    IlhasTmp  $\leftarrow$  Ilhas
    while IlhasTmp  $\neq \emptyset$  do                                 $\triangleright$  todas as ilhas processados no ciclo
      if Clientes  $\neq \emptyset$  then
        W  $\leftarrow$  cliente(Clientes)                                 $\triangleright$  verifica cliente disponível
        Clientes  $\leftarrow$  Clientes  $\setminus \{ W \}$ 
        I  $\leftarrow$  island(IlhasTmp)                                 $\triangleright$  seleciona ilha não processada
        IlhasTmp  $\leftarrow$  IlhasTmp  $\setminus \{ I \}$ 
        enviaMsg(W, I)                                 $\triangleright$  cliente processará ilha I
      end if
      if (W  $\leftarrow$  ClienteTerminou)  $\neq \emptyset$  then
        Clientes  $\leftarrow$  Clientes  $\cup \{ W \}$                                  $\triangleright$  cliente disponível
      end if
    end while
    h  $\leftarrow$  ResultadosDasIlhas()                                 $\triangleright$  retorna o melhor h das ilhas
    Cobertos  $\leftarrow$  Cobertura(h, Exemplos)                                 $\triangleright$  calcula exemplos positivos
    cobertos por h
    Exemplos  $\leftarrow$  Exemplos  $\setminus$  Cobertos                                 $\triangleright$  nó mestre remove os exemplos
    positivos cobertos por h
    if Exemplos  $\neq \emptyset$  then broadcast(Clientes, removeExemplos(Cobertos))  $\triangleright$ 
    nós clientes removem exemplos positivos cobertos
    end if
    Teoria  $\leftarrow$  Teoria  $\cup \{ h \}$ 
  end while
  return Teoria
end function
```

análise. Todo o resto da análise pode ser feito com um programa Prolog e alguns poucos comandos reunidos em um *script*.

1. No primeiro passo, usamos um script e/ou um programa Prolog para gerar as ilhas, mais especificamente os ficheiros que contenham as conclusões para cada uma das cláusulas tratadas nas ilhas. Não há nenhuma alteração nos arquivos de exemplos.
2. Neste passo, utiliza-se um pequeno programa em Prolog que executa o sistema MDIE em cada ilha e reúne os conjuntos de cláusulas geradas (consistentes e inconsistentes). Este programa, em seguida, produz todas as cláusulas consistentes realizando a intersecção das listas de cobertura das cláusulas combinadas e retorna a melhor cláusula consistente. Nesta etapa, pode-se utilizar o código de intersecção de listas de cobertura de cláusulas implementadas em sistemas MDIE existentes.
3. No passo final, precisamos de um programa ou *script* que implemente a etapa que busque a saturação/redução das cláusulas, a atualização dos exemplos positivos, e ainda removendo os exemplos cobertos até que não haja exemplos mais positivos em aberto.

4.3 Avaliando o algoritmo

O Algoritmo 1 foi testado em cinco conjuntos de dados que estão caracterizados na Tabela 4.1. Os resultados são apresentados na Tabela 4.2 e Tabela 4.3 para os conjuntos mutagenesis (descrito em [SMKS94]) e carcinogenesis (descrito em [SKMS97]). Todas as ilhas foram corretamente identificadas em todos os conjuntos de dados. As ilhas identificadas para o conjunto CPDBAS (descrito em [CPC⁺11]) é mostrada nas Tabelas B.2 a B.5 do Apêndice B, para o conjunto DBPCAN (descrito em [CPC⁺11]) é mostrada na Tabela B.6 a B.9 do Apêndice B e para o conjunto *triazines* (descrito em [KMLS92]) na Tabela B.1 do Apêndice B.

conjunto de dados	número de exemplos	número de declarações de modo	número de <i>ilhas</i>
mutagenesis	125/63	22	5
carcinogenesis	162/136	34	4
DSSTox-CPDBAS	843/966	218	37
DSSTox-DBPCAN	80/98	218	37
triazines	17063/17063	40	2

Tabela 4.1: Caracterização dos conjuntos de dados. Os dois valores nas células da coluna "número de exemplos" indicam o número de exemplos positivos e negativos respectivamente.

modeh(active(+drug))	
modeb(lumo(+drug,-energy)) modeb(lteq(+energy,#energy)) modeb(gteq(+energy,#energy))	modeb(methyl(+drug,-ring)) modeb(nitro(+drug,-ring)) modeb(ring_size_5(+drug,-ring)) modeb(ring_size_6(+drug,-ring)) modeb(hetero_aromatic_5_ring(+drug,-ring)) modeb(hetero_aromatic_6_ring(+drug,-ring)) modeb(carbon_6_ring(+drug,-ring)) modeb(carbon_5_aromatic_ring(+drug,-ring)) modeb(benzene(+drug,-ring))
modeb(logp(+drug,-hydrophob)) modeb(lteq(+hydrophob,#hydrophob)) modeb(gteq(+hydrophob,#hydrophob))	modeb(bond(+drug,-atomid,-atomid,#int)) modeb(atm(+drug,-atomid,#element,#int,-charge)) modeb(lteq(+charge,#charge)) modeb(gteq(+charge,#charge))
	modeb(ball3(+drug,-ringlist)) modeb(phenanthrene(+drug,-ringlist)) modeb(anthracene(+drug,-ringlist))

Tabela 4.2: As 5 ilhas identificadas pelo Algoritmo 1 no conjunto de dados *mutagenesis*. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

modeh(active(+drug))	
modeb(five_ring(+drug,-ring)) modeb(connected(+ring,+ring)) modeb(non_ar_hetero_5_ring(+drug,-ring)) modeb(non_ar_5c_ring(+drug,-ring)) modeb(six_ring(+drug,-ring)) modeb(non_ar_hetero_6_ring(+drug,-ring)) modeb(non_ar_6c_ring(+drug,-ring)) modeb(ar_halide(+drug,-ring)) modeb(alkyl_halide(+drug,-ring)) modeb(ester(+drug,-ring)) modeb(phenol(+drug,-ring)) modeb(alcohol(+drug,-ring)) modeb(sulfide(+drug,-ring)) modeb(ether(+drug,-ring)) modeb(ketone(+drug,-ring)) modeb(amine(+drug,-ring)) modeb(amine(+drug,-ring)) modeb(methoxy(+drug,-ring)) modeb(methyl(+drug,-ring)) modeb(sulfo(+drug,-ring)) modeb(sulfo(+drug,-ring)) modeb(nitro(+drug,-ring)) modeb(nitro(+drug,-ring)) modeb(ashby_alert(#alert,+drug,-ring))	modeb(mutagenic(+drug)) modeb(has_property(+drug,#property,#propval)) modeb(ames(+drug))
modeb(ind(+drug,#alert,-nalerts)) modeb(gteq(+nalerts,#nalerts)) modeb(lteq(+nalerts,#nalerts))	modeb(atm(+drug,-atomid,#element,#integer,-charge)) modeb(gteq(+charge,#charge)) modeb(lteq(+charge,#charge)) modeb(symbolbond(+drug,+atomid,-atomid,#integer))

Tabela 4.3: As 4 ilhas identificadas pelo Algoritmo 1 no conjunto de dados *carcinogenesis*. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

conjunto de dados	clauselength = 6, nodes=1000000	
	language=2	language=3
mutagenesis	2.9(2.2)	1.4(1.0)

Tabela 4.4: *Speedups*⁶ para diferentes valores de parâmetro language no conjunto de dados *mutagenesis*. Os valores indicados são a média e desvio padrão de 5 sequências de saturação/redução.

O Algoritmo 2 foi ensaiado num dos conjunto de dados mais emblemáticos do ILP, o *mutagenesis*⁴. O algoritmo foi testado sobre as etapas básicas do algoritmo de cobertura MDIE, especificamente os passos centrais do ciclo de “cobertura gananciosa”, que são também os passos onde é dispendido mais tempo de computação: a saturação seguida da redução. As experiências foram realizadas numa máquina com dois processadores do tipo Xeon quad Core com 32 GB de RAM. Nestas experiências foi variado o limite máximo de repetições de um símbolo de predicado numa cláusula (parâmetro *language*) e os resultados são a média de saturar e reduzir 5 exemplos. Embora o nosso interesse seja a independência das ilhas, que permite a execução de forma completamente independente, os resultados mostram-se promissores para máquinas de memória partilhada.

4.4 Melhoramentos para a implementação

A implementação foi realizada utilizando, tanto quanto possível código já disponível, de modo a obter um protótipo rapidamente. Para esta implementação devemos considerar que podem haver melhorias na eficiência, incluindo uma estrutura de dados mais elaborada. Observando a operação do sistema, podemos perceber dois pontos de melhoria: otimização do espaço de armazenamento das cláusulas inconsistentes e otimização do tempo dispensado na intersecção dos resultados intermédios. Para comprovar estas melhorias observamos que cada “nó” cliente gerou em um ficheiro as cláusulas inconsistentes (cobrindo mais exemplos positivos do que o necessário)

⁴É também designado humoristicamente como a Drosófila do ILP.

e também a melhor cláusula consistente. Podemos ainda observar que há uma grande quantidade de estruturas comuns nas cláusulas produzidas por um sistema ILP. Como sugerido em [CtCR08], o conjunto de cláusulas pode ser armazenado de forma eficiente numa Trie (*prefix tree*) poupando uma quantidade considerável de espaço em termos de armazenamento. Esta economia é conseguida sobretudo porque entre cada cláusula e as suas especializações varia no máximo um literal que está localizado no final da cláusula. Como numa Trie os prefixos comuns das estruturas são guardados uma única vez consegue-se uma enorme economia de espaço de armazenamento

Se observarmos que a principal atividade do "nó" Mestre é realizar interseções entre os intervalos de cobertura, para gerar a cobertura de novas cláusulas, podemos acelerar a operação de interseção se codificarmos as listas de cobertura das cláusulas como vetores de bits (bit vectors).

4.5 Comparando com outras implementações

Das estratégias de implementação do ILP paralelo proposto por Fonseca [FSSC09] somente a abordagem designada por DP-LT (Data Parallel Learn Theory) é adequada para computação distribuída desde que as tarefas (ou sub-tarefas) possam ser processadas de forma independente. A principal desvantagem é que cada sub-tarefa representa uma parte dos exemplos utilizados. No modelo proposto pelo algoritmo paralelo todas as ilhas podem referenciar todos os dados necessários e a execução da ilha é completamente independente, sem nenhuma interação entre elas.

4.6 Conclusões

O algoritmo proposto neste capítulo tem a vantagem de processar sub-tarefas de forma independente, o que o torna adequado para sistemas distribuídos em máquinas independentes, e o uso de ambientes de computação como o Condor ou Compu-

tação em Grid. Como foi descrito a implementação é bastante fácil e rápida usando um sistema MDIE existente para o qual tenhamos acesso ao código fonte. Este algoritmo também pode ser implementado, ou embutido no código de um sistema MDIE existente. Este recurso torna útil para sistemas ILP em aplicações *Multi Relational Data Mining*.

Capítulo 5

Utilização e Avaliação do Sistema

HARVARD

Muitas tarefas de DM que podem ser decompostas em sub/tarefas independentes, são adequadas para serem eficientes e automaticamente implementadas como um conjunto de tarefas HARVARD.

Neste capítulo descrevemos um conjunto de experiências que foram concebidas para avaliar as funcionalidades do sistema HARVARD e mostrar que os objetivos propostos foram alcançados. Foram utilizados conjuntos de dados de domínios bastante diversificados mostrando a generalidade e em diversas tarefas de DM, como a classificação e a regressão.

Para além do seu uso geral como sistema de computação distribuída, as experiências procuram evidenciar as facilidades de execução de tarefas frequentes em *Data Mining* como a sintonização de parâmetros dos algoritmos de análise de dados, escolha de um conjunto adequado de atributos (*feature subset selection*) ou *cross validation*. Estas tarefas têm como característica comum o fato de serem “trivialmente paralelizáveis”. Esta característica faz com que possam usufruir das vantagens de um sistema distribuído tanto para acelerar a sua realização como para poder suportar maior volume de dados, ou obter melhores resultados.

Na Secção 5.1 são descritas experiências onde algumas das tarefas de pré-processamento em DM são facilmente realizadas com recurso ao sistema HARVARD. Na Secção 5.2 são descritas as experiências em que o sistema pode ajudar no processo de análise de dados propriamente dito. Na Secção 5.3 é ilustrado como o algoritmo proposto no Capítulo 4 foi adaptado para ser executado no sistema HARVARD. Na última secção são retiradas conclusões sobre as experiências relatadas no capítulo. As experiências foram realizadas em 10 máquinas da LAN da Faculdade de Engenharia da Universidade do Porto (FEUP).

5.1 Utilização em pré-processamento de *Data Mining*

É reconhecido que grande parte do tempo de uma tarefa de KDD é dispendido no pré-processamento. É portanto importante o desenvolvimento de ferramentas que possam facilitar e acelerar esta fase do processo de KDD. A linguagem de especificação das tarefas para o sistema HARVARD disponibiliza um conjunto de *macro tarefas* que permitem ao utilizador evitar escrever em XML a decomposição da operação num conjunto grande de sub-tarefas, que são executadas em paralelo ou sequencialmente. No uso desta facilidade (referida no Capítulo 3), o utilizador apenas indica o nome da macro tarefa e os correspondentes parâmetros que cada uma necessita e é gerado automaticamente o XML para o conjunto de execuções paralelas e/ou distribuídas que implementam a operação. Para efeitos ilustrativos são agora descritas algumas implementações úteis para pré-processamento de dados.

Escolha de um bom conjunto de atributos

O desempenho de qualquer algoritmo de Aprendizagem Computacional depende decisivamente do conjunto de atributos que lhe são disponibilizados. Um conjunto

inadequado de atributos faz com que o algoritmo não consiga construir bons modelos ou não consiga construir nenhum modelo mesmo. Existem por vezes situações em que o número de atributos é elevado. Nessas situações, o sistema pode demorar demasiado tempo a construir o modelo. Pode acontecer que os atributos escolhidos pelo algoritmo não sejam os mais adequados. Para minimizar estas situações foram propostas várias abordagens, designadas globalmente como *feature subset selection*, que permitem estimar qual o subconjunto de atributos iniciais mais adequado.

Neste caso de estudo é mostrado uma das macro tarefas do sistema HARVARD que implementa uma procura em largura no “espaço de atributos”. Nesta procura sistemática, o algoritmo começa com todos os atributos iniciais e prossegue numa pesquisa em largura-primeiro (por níveis) em que no primeiro nível considera todas as combinações em que se retira um atributo ao conjunto inicial. No segundo nível se retiram 2 atributos e assim sucessivamente até às combinações em que só é usado um atributo ou (por questões pragmáticas) ser atingido um nível máximo especificado pelo utilizador. Para problemas de elevada dimensionalidade esta procura requer um elevado poder computacional e pode ser muito demorada, pelo que podem/devem ser adotadas alternativas conhecidas mais expeditas.

Esta macro tarefa recebe como parâmetros o conjunto de dados, o limite para o nível de profundidade e o algoritmo de Aprendizagem Computacional usado na avaliação das combinações de atributos¹. Descrevemos agora a implementação da macro tarefa para processar ficheiro em formato ARFF com algoritmos do Weka e depois a sua implementação para um caso de ILP.

Utilização com ficheiros ARFF e Weka

A implementação foi realizada com o script principal (Tabela C.1 do Apêndice C) que gera os nós, chamando por vezes scripts auxiliares, como o *script* “getResults”

¹No caso do ILP o sistema Aleph é utilizado pelo que não é necessário especificá-lo

número de atributos	número de classes	número de exemplos
42	2 ('normal'/'anomaly')	125973

Tabela 5.1: Caracterização do conjunto de dados NSL-KDD usado para ilustrar a macro tarefa de *feature subset selection* implementada para conjuntos de dados no formato ARFF.

(Tabela C.2 do Apêndice C), que analisa os resultados dos nós que avaliam cada combinação e devolve o resultado final. Este resultado inclui as medidas avaliadas pelo Weka: accuracy, true positives, false positives, precision, recall, F measure , e ROC.

Nesta experiência foi utilizado o conjunto de dados NSL-KDD² que é uma atualização do conjunto de dados sobre deteção de intrusão lançado como desafio no KDD'99, caracterizado na Tabela 5.1.

O comando executado para criar o código e dados para cada tarefa HARVARD foi

```
mkFeatureSelection weka.classifiers.trees.J48 NSL - KDDTrain.arff NSL - KDDTune.arff 2
```

em que se indica o algoritmo usado para avaliar os diferentes conjuntos de atributos (primeiro argumento), o conjunto de treino (segundo argumento), o conjunto de avaliação (turnig set – terceiro argumento) e o nível máximo de profundidade. Com este comando foram gerados 861 nós, cada um com uma combinação diferente de atributos, que foram, executados em paralelo e um nó final que calcula o melhor resultado e que é executado depois de uma barrier esperar pelas execuções paralelas. Por este exemplo podemos ver a facilidade que representa para o utilizador a sua utilização e o tempo que lhe é poupado.

A execução teve o seguinte resultado:

²disponível em <http://www.iscx.ca/NSL-KDD/>

número de declarações de determinação	número de exemplos positivos/negativos
30	162/136

Tabela 5.2: Caracterização do conjunto de dados relacional *carcinogenesis* usado para ilustrar a macro de *feature subset selection* implementada para conjuntos de dados no formato usado pelo Aleph.

Best accuracy = 99.86

@attribute 'num_compromised' real

@attribute 'dst_host_same_src_port_rate' real

removed

Utilização do Aleph

Está também disponível uma macro para *feature subsection selection* implementando a abordagem descrita anteriormente mas desta vez para o sistema de Indução de ILP Aleph [Sri03].

A implementação foi realizada com o *script* principal (Tabela C.3 do Apêndice C) que gera os nós, chamando por vezes *scripts* auxiliares como o *script* “computeResult” (Tabela C.4 do Apêndice C) que analisa os resultados dos nós que avaliam cada combinação e devolve o resultado final. Este resultado inclui a medida avaliada pelo Aleph (Accuracy).

Nesta experiência foi utilizado o conjunto de dados *carcinogenesis* descrito no Capítulo 4. As características deste conjunto de dados são mostradas na Tabela 5.2

Usando o comando

```
mkFeatureSelection carcinogenesis 2
```

foram gerados 465 nós com as combinações de declarações de determinação e um nó final que combina os resultados dos anteriores que correm em paralelo.

A execução teve o seguinte resultado:

Best accuracy = 0.59

Remove the following determinations to get the best result:

determination(active/1,has_property/3).

determination(active/1,non_ar_6c_ring/2).

Escolha de um bom conjunto de valores para os parâmetros de um algoritmo

O desempenho de um algoritmo de Aprendizagem Computacional depende, em muitos casos, fortemente da combinação de valores dos seus parâmetros para analisar determinado conjunto de dados. Os valores por omissão nem sempre são os melhores para todos os conjuntos de dados. O HARVARD disponibiliza uma macro tarefa para sintonização de parâmetros de algoritmos de classificação disponíveis no Weka.

Utilização do Weka

A implementação foi realizada com o script principal (Tabela C.5 do Apêndice C) que gera os nós, chamado por vezes scripts auxiliares como o *script* “getResults” (Tabela C.6 do Apêndice C) que analisa os resultados dos nós que avaliam cada combinação e devolve o resultado final. Este resultado inclui as medidas avaliadas pelo Weka: accuracy, true positives, false positives, precision, recall, F measure , e ROC.

Nesta experiência foi utilizado o conjunto de dados CPDBAS referido já no Capítulo 4 (descrito em [CPC⁺11]) e que contém um conjunto de moléculas e seus resultados em estudos de toxicidade, caracterizado na Tabela 5.3.

O comando executado para criar o código e dados para cada tarefa HARVARD foi

```
mkParameterTuning j48.conf 1D2D_CPDBAS_v5c_1547_29Apr2008_md_Train.arff 1
D2D_CPDBAS_v5c_1547_29Apr2008_md_Test.arff 4
```

número de atributos	número de classes	número de exemplos
564	2 (activo/inactivo)	2292

Tabela 5.3: Caracterização do conjunto de dados CPDBAS usado para ilustrar a macro tarefa de sintonização de parâmetros implementada para conjuntos de dados no formato ARFF.

em que se indica um ficheiro com o nome do algoritmo a usar e os valores para cada parâmetro do algoritmo (primeiro argumento), o conjunto de treino (segundo argumento), o conjunto de avaliação (turnig set – terceiro argumento) e a posição da classe (quarto argumento). Com este comando foram gerados 72 nós, cada um com uma combinação diferente de parâmetros, que foram, executados em paralelo e um nó final que calcula o melhor resultado e que é executado depois de uma barrier esperar pelas execuções paralelas.

A execução teve o seguinte resultado:

Best accuracy = 62.81 [-M 25 -C 0.2 -A]

Best true positives = 0.63 [-M 25 -C 0.2 -A]

Best false positives = 0.37 [-M 25 -C 0.2 -A]

Best precision = 0.62 [-M 25 -C 0.2]

Best recall = 0.63 [-M 25 -C 0.2 -A]

Best F measure = 0.62 [-M 25 -C 0.2]

Best ROC = 0.67 [-M 25 -C 0.2 -A]

Indicando valores para as várias medidas disponibilizadas pelo Weka e o correspondente conjunto de parâmetros.

5.2 Utilização em tarefas de análise de dados

Validação Cruzada

Validação cruzada é uma técnica largamente utilizada para estimar a qualidade de um classificador. Para conjunto de dados grandes o processo pode ser acelerado correndo cada um dos componentes da validação cruzada em paralelo e calculando

número de atributos	número de exemplos
10	189

Tabela 5.4: Caracterização do conjunto de dados lowbwt usado para ilustrar a macro tarefa de validação cruzada implementada para conjuntos de dados no formato ARFF.

no final o resultado. É esta abordagem que descrevemos para o caso de ficheiros ARFF usando algoritmos do Weka e para ILP.

Utilização do Weka

A implementação foi realizada com o script principal (Tabela C.7 do Apêndice C) que gera os nós, chamado por vezes scripts auxiliares como o *script* “getResults” (Tabela C.8 do Apêndice C) que analisa os resultados dos nós que avaliam cada combinação e devolve o resultado final. Este resultado inclui as medidas avaliadas pelo Weka: accuracy, true positives, false positives, precision, recall, F measure , e ROC.

Nesta experiência foi utilizado o conjunto de dados de regressão [KCJ98] sobre estimação do risco de “dar à luz” crianças com baixo peso, caracterizado na Tabela 5.4.

O comando executado para criar o código e dados para cada tarefa HARVARD foi

mkCVweka lowbwt.arff weka.classifiers.functions.SMOreg 3 10

em que se indica um ficheiro de dados (primeiro argumento), o algoritmo a usar treino (segundo argumento), o número de “folds”(terceiro argumento) e a posição da classe (quarto argumento). Com este comando foram gerados 4 nós, cada um com um par treino/teste correspondente a um “fold”, que foram, executados em paralelo e um nó final que calcula o melhor resultado e que é executado depois de uma barrier esperar pelas execuções paralelas.

A avaliação desta macro tarefa utilizou a ligação ao Condor com a configuração um nó servidor e três clientes.

A execução teve o seguinte resultado:

$$\text{RMSerr} = 448.026(43.146)$$

Que representa o *Root Mean Square Error*.

Utilização do Aleph

A implementação foi realizada com o script principal (Tabela C.9 do Apêndice C) que gera os nós, chamado por vezes scripts auxiliares como o *script* “getResults” (Tabela C.10 do Apêndice C) que analisa os resultados dos nós que avaliam cada combinação e devolve o resultado final. Este resultado inclui a medida avaliada pelo Aleph (Accuracy).

Nesta experiência foi utilizado o conjunto de dados carcinogenesis referido já no Capítulo 4.

O comando executado para criar o código e dados para cada tarefa HARVARD foi

$$mkCV \text{ carcinogenesis } 10$$

em que se indica o conjunto de dados e o número de folds, Com este comando foram gerados 10 nós, cada um com um par treino/teste correspondente a um “fold”, que foram, executados em paralelo e um nó final que calcula o melhor resultado e que é executado depois de uma barrier esperar pelas execuções paralelas.

A execução teve o seguinte resultado:

$$\text{accuracy} = 0.597(0.109)$$

5.3 Executando um sistema de ILP no HARVARD

O algoritmo de ILP proposto no Capítulo 4 foi adaptado para poder utilizar o sistema HARVARD. Para realizar a adaptação foi feita uma alteração do ciclo principal

característico (de cobertura gananciosa) de sistemas como Aleph. A questão é que a linguagem de workflow do HARVARD não tem a definição de ciclos pelo que foi preciso pensar um processo alternativo. Esta foi a principal questão para a adaptação conforme descrevemos a seguir.

A implementação do Algoritmo 1 de identificação das *ilhas* não sofreu qualquer alteração pois as ilhas são independentes da parte de execução do algoritmo. A parte relativa aos Clientes consistiu em torná-los autónomos. Isto é, correm do princípio ao fim sem precisarem de receber mensagens para trabalhar. O código original de um cliente recebe o conjunto de dados em que o ficheiro de *background knowledge* só tem as declarações de determinação e modo correspondentes a uma ilha e recebe como parâmetro o número de exemplos positivos que vai usar na saturação/redução. Corre do princípio ao fim de forma independente e devolve as cláusulas consistentes e inconsistentes aceitáveis. As cláusulas aceitáveis satisfazem as restrições de cobrir um número mínimo de exemplos positivos (parâmetro *minpos*), ter um número de literais inferior ao limite (parâmetro *clauselength*) e não excederem o limite de repetições do mesmo símbolo de predicado (parâmetro *language*). O ponto mais elaborado do processo de adaptação deriva, como se disse, de não haver ciclos na linguagem de workflow do HARVARD. Para contornar este problema o ciclo principal do processo de “cobertura gananciosa” foi implementado no código do processo final (antigo nó Mestre). Usando o código inicial do nó Mestre da implementação MPI descrita no Capítulo 4 foi retirado todo o processo de troca de mensagens pois agora este nó corre de modo independente e quando é executado já tem todos os dados (cláusulas produzidas por todas as ilhas) para produzir o modelo final. É construída inicialmente uma lista “global” dos exemplos positivos. O processo implementado é um ciclo em que cada iteração é calculada a melhor cláusula consistente. Esta cláusula é adicionada ao modelo e os exemplos positivos cobertos por essa cláusula são retirados à lista “global” de positivos. Se a lista “global” ficar vazia, o processo termina pois não há mais exemplos positivos por cobrir. No caso contrário simula-

se a remoção destes exemplos positivos fazendo a intersecção da lista “global” de positivos com a lista de positivos da cada cláusula individual. O resultado desta intersecção é uma lista idêntica à avaliação da cláusula pelo método tradicional de “teste contra os exemplos” com os exemplos positivos, já cobertos, removidos do conjunto de dados.

Esta implementação foi avaliada no conjunto de dados mutagenesis tendo sido executados (em 10 máquinas da LAN da FEUP) 625 nós clientes em paralelo e um nó final que construiu o resultado. O resultado de *accuracy* foi semelhante à execução sequencial.

Esta implementação tem como principal limitação a necessidade de efetuar a saturação/redução em todos os exemplos. Num sistema como o Aleph é definida uma amostra limitada (3 a 5 exemplos), os exemplos desta amostra são usados para a saturação/redução e depois a melhor cláusula é logo aceite, os exemplos cobertos removidos e o ciclo continua até não haver positivos por cobrir. Neste processo geralmente são efetuados muito menos passos de saturação/redução do que na implementação HARVARD que tem que usar todos os positivos.

No entanto, o fato de termos os resultados da saturação/redução para todas as cláusulas permite implementar (o que não foi ainda feito) uma procura ao nível da teoria (descrita como *theory-level search* no manual do Aleph). Isto é evitar a procura gananciosa habitual e encontrar uma teoria que, por exemplo, cubra todos os exemplos positivos com o menor número de cláusulas.

5.4 Conclusões

Neste capítulo utilizamos uma gama variada de conjuntos de dados com características distintas de modo a explorar e demonstrar a utilidade das funcionalidades disponíveis no HARVARD.

Foi mostrada a utilidade das macro tarefas codificando tarefas frequentes em análise de dados utilizando algoritmos de Aprendizagem Computacional, facilitando

enormemente a tarefa do utilizador.

A utilização do HARVARD com sistemas de ILP torna mais viável a utilização deste tipo de algoritmos de Data Mining Multi Relational possibilitando tanto análises mais complexas em tempo útil, uma vez que a linguagem de descrição tanto de dados como dos modelos é, no ILP, bastante poderosa, como tempos de resposta mais rápidos.

Capítulo 6

Conclusões

A quantidade de dados disponíveis em suporte digital é imensa e cresce a taxas exponenciais, e portanto, extrair informação de grandes quantidades de dados é hoje em dia uma tarefa virtualmente impossível de realizar manualmente e que requer, quando realizado por máquinas, elevada capacidade computacional.

O sistema HARVARD (*HARVesting Architecture of idle machines foR Data mining*) apresentado nesta tese, mostra como o processo de *Extração Automática de Conhecimento* pode ser viável economicamente em qualquer organização, e pode ser feito usando recursos computacionais distribuídos numa rede local. Na verdade com o HARVARD temos à disposição um "*super-computador virtual*" usando computadores convencionais dispersos por uma organização e que por vezes, em alguns períodos de tempo, estão ociosos. Esta ociosidade pode ser convertida em processamento útil em benefício das organizações interessadas em utilizar KDD para análise de grandes quantidades de dados e assim obter alguma vantagem competitiva.

A arquitetura proposta, juntamente com a proposta que foi feita nesta tese para uma implementação paralela de um algoritmo de ILP, aumenta as possibilidades de KDD pelo menos em três vertentes. O processo de análise de grandes quantidades de dados passa a ser economicamente viável pois não requer hardware especial e caro. O processo de KDD pode ser exequível para grandes quantidades de dados

pois pode ser realizado em tempo útil. Pode ser viável a utilização de algoritmos como os de ILP em análises de MRDM construindo modelos bastante complexos e, conseqüentemente, realizar análises mais profundas nos dados.

6.1 Contribuições

O HARVARD é um ambiente de baixo custo pois utiliza recursos computacionais disponíveis e ociosos de uma organização, e portanto não perturba o normal uso dos recursos na rotina diária. É uma plataforma flexível e versátil que pode utilizar diferentes ferramentas pré-existentes de análise de dados sem re-programações ou adaptações, sendo independente quanto à ferramenta requerida para a tarefa de KDD. É ainda uma plataforma fiável já que incorpora características de segurança e controle das operações e com esquemas de tolerância a falhas. Fácil de utilizar dispondo de uma linguagem de especificação de tarefas de KDD simultaneamente poderosa e de fácil utilização por não especialistas.

Como diferencial deste trabalho podemos destacar:

- (a) proposta de um novo algoritmo de execução paralelo de ILP;
- (b) interligação do ambiente com um sistema de ILP permite a utilização de *Relational Data Mining* (RDM) para aplicações KDD em grandes bases de dados do mundo real superando as questões da exigência de elevado poder computacional;
- (c) o uso de recursos computacionais distribuídos e ociosos como uma alternativa para aplicações que necessitam elevado poder computacional;
- (d) o desenvolvimento de uma linguagem de descrição do processo de KDD pode viabilizar a disseminação desta tecnologia entre investigadores não habituados com o assunto, mas interessados em explorar conhecimentos “desconhecidos” em bases de dados;

- (e) interligação do HARVARD com ambientes *Grid Computing*, que permite estender ainda mais as funcionalidades do ambiente básico, e assim prover análises de dados ainda mais complexas, compartilhando de recursos computacionais externos à organização.
- (f) utilização de macro tarefas que facilitam o uso do sistema em tarefas de KDD. O uso de macro tarefas torna o sistema extensível pois basta desenvolver os *scripts* de implementação da macro tarefa e atualizar o pré-processador.

6.2 Trabalho futuro

Um projeto da dimensão do HARVARD pode sempre ser aperfeiçoado e aumentado a fim de despertar maior interesse e uso prático nas organizações interessadas em se beneficiar desta plataforma.

Melhoria de interface e interação com o utilizador É necessário que algumas melhorias sejam implementadas no HARVARD a fim de se tornar um ambiente ainda mais facilmente manipulável por utilizadores em geral. Afinal, certos utilizadores estão interessados apenas em analisar dados sem preocupação com configurações e requisitos técnicos extras para determinados ambientes computacionais. Para isso, é desejável desenvolver um ambiente com uma interface gráfica de modo a facilitar ainda mais a interação com o utilizador.

Para além disso, a criação de uma interface gráfica baseada em objetos padronizados a serem organizados na forma de um *workflow* que sistematicamente podem originar os ficheiros de tarefas KDD utilizados como entrada junto ao HARVARD. Isso pode contribuir para que o utilizador se preocupe ainda menos com a sintaxe e comandos do HARVARD.

Bibliotecas e repositórios de algoritmos Data Mining Outra possibilidade relevante é criar uma biblioteca de técnicas e tarefas KDD num repositório interno à

organização. Os algoritmos deverão estar, preferencialmente, codificadas em linguagem Java para que o sistema HARVARD fique ainda mais robusto. Esta biblioteca poderia estar adaptada ao pleno funcionamento em conjunto com um ambiente *Grid* da organização, ou mesmo, quando criada ser facilmente transferida para processamento em ambientes *Grid Computing* externos.

Melhoria nos aspectos de segurança dos dados A concepção original do HARVARD não prevê privacidade quanto ao tratamento dos dados. Isso significa que quando os dados são transferidos para uma estação cliente para a execução de uma determinada tarefa, o acesso aos dados residentes na estação poderia ser facultado ao próprio utilizador daquela estação. Embora o HARVARD seja concebido para ser executado num ambiente local, portanto interno a uma determinada organização e num momento de eventual ociosidade, é presumível que os dados em análise sejam protegidos de qualquer acesso. A sugestão então, é dotar o HARVARD de um módulo ou funcionalidade que permita manipular os dados na estação cliente, sem permitir o acesso aos mesmos dados, seja por aplicação residente na estação cliente ou algum utilizador estranho a operação do HARVARD. Estas preocupações de segurança são importantes para aplicação em bancos, por exemplo.

Integração com ambientes KDD, como Weka ou Rapidminer Os ambientes Weka e Rapidminer não operam de forma distribuída como o HARVARD. Assim, um trabalho interessante é prover uma interface entre os ambientes WEKA e/ou Rapidminer de modo que as tarefas especificadas nesses ambientes possam ser processadas no HARVARD de modo a estender as capacidades daquelas ferramentas e dar maior versatilidade ao HARVARD.

Linguagem de *workflow* Embora a linguagem de especificação de execução de tarefas seja poderosa para a realização de muitas tarefas de KDD, a sua extensão pode ser conveniente para a adaptação de algoritmos ao ambiente HARVARD. Por

exemplo, a definição de ciclos na linguagem pode permitir decompor os passos de um algoritmo num conjunto de tarefas que seria executado num ciclo. Muitos algoritmos tem um ciclo central em que se executam um conjunto de operações até que seja satisfeita uma condição. Se essas operações forem pesadas computacionalmente é vantajoso realizá-la com tarefas HARVARD em paralelo (se for possível) e o ciclo ser definido no workflow do processo KDD.

Bibliografia

- [Alp09] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2 edition, December 2009.
- [And04] D.P. Anderson. Boinc: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10, 8 Nov. 2004.
- [AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.
- [BB99] Mark Baker and Rajkumar Buyya. Cluster computing: The commodity supercomputer. *Softw., Pract. Exper.*, 29(6):551–576, 1999.
- [BBH99] Mark Baker, Rajkumar Buyya, and Daniel C. Hyde. Cluster computing: A high-performance contender. *IEEE Computer*, 32(7):79–80, 1999.
- [BBL02] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *Softw., Pract. Exper.*, 32(15):1437–1466, 2002.

-
- [BDH03] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, March 2003.
- [BP01] J. Batheja and M. Parashar. Adaptive cluster computing using javaspace. In *Cluster Computing, 2001. Proceedings. 2001 IEEE International Conference on*, pages 323–330, 8–11 October 2001.
- [BPK00] Helmut Berrer, Iain Paterson, and Jorg Keller. Evaluation of machine-learning algorithm ranking advisors, 2000.
- [Bra06] Tim Bray. Extensible markup language (xml) 1.0 (fourth edition). Technical report, W3C, 2006.
- [BS00] P. Brazdil and C. Soares. Ranking classification algorithms based on relevant performance information. In J. Keller and C. Giraud-Carrier, editors, *Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, 2000.
- [Cam00] R. Camacho. *Inducing Models of Human Control Skills using Machine Learning Algorithms*. PhD thesis, Department of Electrical Engineering and Computation, Universidade do Porto, 2000.
- [Cam04] Rui Camacho. Indlog –induction in logic. In José Alferes e João Leite, editor, *JELIA 2004 - 9th European Conference on Logics in Artificial Intelligence*, volume LNAI 3229, pages 718–721. Springer-Verlag, 2004. 27–30 September, Lisboa, Portugal.
- [CCK⁺00] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. Crisp-dm 1.0 step-by-step data mining guide. Technical report, The CRISP-DM consortium, August 2000.

- [CDG⁺06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.
- [CPC⁺11] Rui Camacho, Max Pereira, Vítor Santos Costa, Nuno A. Fonseca, Carlos Adriano Gonçalves, Carlos J. V. Simões, and Rui M. M. Brito. A relational learning approach to structure-activity relationships in drug design toxicity studies. *J. Integrative Bioinformatics*, 8(3), 2011.
- [CSF⁺08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.
- [CtCR08] Rui Camacho, Nuno A. Fonseca, Vitor Santos Costa, and R. Rocha. ILP :- Just Trie It. In H. Blockeel et al., editor, *Proceedings of the 2007 International Conference on Inductive Logic Programming (ILP 2007)*, number 4894 in Lecture Notes in Artificial Intelligence, pages 78–87. Springer-Verlag, 2008.
- [DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, jan 2008.
- [DKFY10] Frédéric Desprez, Ottmar Krämer-Fuhrmann, and Ramin Yahyapour. Cloud computing - introduction to the special theme. *ERCIM News*, 2010(83):12–13, 2010.
- [FK97] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*,

-
- 11(2)(2):115–128, 1997. Provides an overview of the Globus project and toolkit.
- [FK99] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 11, pages 259–278. MORGAN-KAUFMANN, 1999. The whole contents of the book is also useful for a complete understanding of Globus.
- [FMM99] Koichi Furukawa, Donald Michie, and Stephen Muggleton, editors. *Machine Intelligence 15, Intelligent Agents [St. Catherine’s College, Oxford, July 1995]*. Oxford University Press, 1999.
- [Fon06] Nuno A. Fonseca. *Parallelism in Inductive Logic Programming Systems*. PhD thesis, University of Porto, 2006.
- [Fos05] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, number 3779 in LNCS, pages 2–13. Springer-Verlag, 2005.
- [FPSSU96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.
- [FSC06] Nuno A. Fonseca, Fernando Silva, and Rui Camacho. April - An Inductive Logic Programming System. In *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA06)*, volume 4160 of *LNAI*, pages 481–484, Liverpool, September 2006. Springer-Verlag.
- [FSSC09] Nuno A. Fonseca, Ashwin Srinivasan, Fernando M. A. Silva, and Rui Camacho. Parallel ilp for distributed-memory architectures. *Machine Learning*, 74(3):257–279, 2009.

- [FSSCC05] Nuno A. Fonseca, Fernando Silva, Vítor Santos Costa, and Rui Camacho. Strategies to paralilize ILP systems. In *15th International Conference on Inductive Logic Programming (ILP 2005), Bonn, August 2005*, number 3625 in Lecture Notes in AI, pages 136–153, Berlin, 2005. Springer-Verlag.
- [FTL⁺01] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: a computation management agent for multi-institutional grids. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 55–63, 7-9 Aug. 2001.
- [FU96] Usama Fayyad and Ramasamy Uthurusamy. Data mining and knowledge discovery in databases. *Commun. ACM*, 39(11):24–26, 1996.
- [Han01] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.
- [Han05] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
- [Hei01] Paulo Heitlinger. *O guia pratico da XML*. Centro Atlantico Ltda, Lisboa, 10 2001.
- [IG96] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.
- [KC00] H. Kargupta and P. Chan, editors. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press, 2000.
- [KCJ98] D. Kilpatrick and M. Cameron-Jones. Numeric prediction using instance-based learning with encoding length selection. In *Progress in Connectionist-Based Information Systems*, 1998.

-
- [KFR08] Richard Kirkby, Eibe Frank, and Peter Reutemann. Weka explorer user guide for version 3-5-8. In *Technical Report of University of Waikato*. 2008.
- [KMLS92] R. D. King, S. Muggleton, R. A. Lewis, and M. J. Sternberg. Drug design by machine learning: the use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences*, 89(23):11322–11326, dec 1992.
- [KWA⁺01] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. Seti@home-massively distributed computing for seti. *Computing in Science & Engineering [see also IEEE Computational Science and Engineering]*, 3(1):78–83, Jan.-Feb. 2001.
- [LLM88] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor-a hunter of idle workstations. In *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111, 13-17 June 1988.
- [Mil08] Michael Miller. *Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online*. Que Publishing Company, 1 edition, 2008.
- [Mug95] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
- [MWK⁺06] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. Yale: rapid prototyping for complex data mining tasks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940, New York, NY, USA, 2006. ACM Press.

- [Sar01] Luis F. G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. In *CCGRID*, pages 337–346, 2001.
- [SBS99] Thomas Sterling, Donald J. Becker, and Daniel F. Savarese. *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters (Scientific and Engineering Computation)*. The MIT Press, May 1999.
- [SKMS97] Ashwin Srinivasan, Ross D. King, Stephen Muggleton, and Michael J. E. Sternberg. Carcinogenesis predictions using ilp. In *Proceedings of the 7th International Workshop on Inductive Logic Programming, ILP '97*, pages 273–287, London, UK, UK, 1997. Springer-Verlag.
- [SMKS94] Ashwin Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.
- [SR08] David Scuse and Peter Reutemann. Weka experimenter tutorial for version 3-5-8. In *Technical Report of University of Waikato*. 2008.
- [Sri03] Ashwin Srinivasan. Aleph manual, 2003.
- [TKH⁺05] Margaret Taft, Ramkumar Krishnan, Mark Hornick, Denis Muhkin, George Tang, Shiby, Thomas, and Peter Stengard. *Oracle Data Mining Concepts, 10g*. Oracle, release 2 (10.2) edition, June 2005.
- [T.M97] T.M.Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

-
- [TWML01] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – a distributed job scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.
- [vLFGL01] Gregor von Laszewski, Ian T. Foster, Jarek Gawor, and Peter Lane. A java commodity grid kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):645–662, 2001.
- [vLH05] Gregor von Laszewski and Mihael Hategan. Workflow concepts of the java cog kit. *J. Grid Comput.*, 3(3-4):239–258, 2005.
- [WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd edition edition, 2005.
- [Whi09] Tom White. *Hadoop: The Definitive Guide*. O’Reilly Media, Inc., 1st edition, 2009.
- [Wil07] Graham Williams. *Open Source Desktop Survival Guide*. Togaware, 03 2007. ISBN: 0-9757109-2-3.

Apêndice A

Lista de Traduções

- *AND-parallelism*: Paralelismo-E
- *Cloud Computing*: Computação em Nuvem
- *Cross Validation*: Validação Cruzada
- *Data Mining*: Extração de Conhecimento
- *Data set*: Conjunto de Dados
- *Grid Computing*: Computação em Grelha
- *Inductive Logic*: Lógica Indutiva
- *Inductive Logic Programming*: Indução de Programas em Lógica
- *Knowledge Discovery in Data Bases*: Extração de Conhecimento
- *Logic Programming*: Programação em Lógica
- *Machine Learning*: Aprendizagem Computacional

Apêndice B

Resultados do ILP

modeh(active(+drug))	
modeb(struc4(+drug,#component,#component)) modeb(struc3(+drug,#component,#component))	modeb(struc4(+drug,-component,#component)) modeb(subst(+component,#component,#component)) modeb(subst(+component,#component,-component)) modeb(subst(+component,-component,#component)) modeb(subst(+component,-component,-component)) modeb(branch(+component,#branchprop)) modeb(branch(+component,-branchprop)) modeb(gt(+branchprop,+branchprop)) modeb(gt(+branchprop,-branchprop)) modeb(sigma(+component,#sigmaprop)) modeb(sigma(+component,-sigmaprop)) modeb(gt(+sigmaprop,+sigmaprop)) modeb(gt(+sigmaprop,-sigmaprop)) modeb(polarisable(+component,#polariprop)) modeb(polarisable(+component,-polariprop)) modeb(gt(+polariprop,+polariprop)) modeb(pi_acceptor(+component,#piaccprop)) modeb(pi_acceptor(+component,-piaccprop)) modeb(gt(+piaccprop,+piaccprop)) modeb(pi_doner(+component,#pidonerprop)) modeb(pi_doner(+component,-pidonerprop)) modeb(gt(+pidonerprop,+pidonerprop)) modeb(h_acceptor(+component,#haccprop)) modeb(h_acceptor(+component,-haccprop)) modeb(gt(+haccprop,+haccprop)) modeb(h_doner(+component,#hdonerprop)) modeb(h_doner(+component,-hdonerprop)) modeb(gt(+hdonerprop,+hdonerprop)) modeb(flex(+component,#flexprop)) modeb(flex(+component,-flexprop)) modeb(gt(+flexprop,+flexprop)) modeb(gt(+flexprop,-flexprop)) modeb(size(+component,#sizeprop)) modeb(size(+component,-sizeprop)) modeb(gt(+sizeprop,+sizeprop)) modeb(gt(+sizeprop,-sizeprop)) modeb(gt(+polariprop,+polariprop)) modeb(polarisable(+component,#polariprop)) modeb(polarisable(+component,-polariprop)) modeb(polar(+component,#polarprop)) modeb(polar(+component,-polarprop)) modeb(gt(+polarprop,+polarprop)) modeb(gt(+polarprop,-polarprop)) modeb(struc4(+drug,#component,-component)) modeb(struc4(+drug,-component,-component)) modeb(struc3(+drug,#component,-component)) modeb(struc3(+drug,-component,-component))

Tabela B.1: Ilhas identificadas pelo Algoritmo 1 no conjunto de dados *Triazines*. O *recall number* (número máximo de soluções alternativas para predicados não determinísticos) não está mostrado nas declarações de modo pois só os tipos são relevantes. modeh() é a declaração de modo da cabeça da cláusula.

modeh(active(+drug))	
<pre> modeb(pharmacophore_fingerprint(+drug,#pf,#lp,#frq)) modeb(chemical_fingerprint(+drug,#chemical_fingerprint)) modeb(tetrahedral_stereoisomer_count(+drug, #tetrahedral_stereoisomer_count)) modeb(tautomer_count(+drug,#tautomer_count)) modeb(wiener_polarity(+drug,#wiener_polarity)) modeb(quaternary_carbon(+drug,#quaternary_carbon)) modeb(tertiary_carbon(+drug,#tertiary_carbon)) modeb(secondary_carbon(+drug,#secondary_carbon)) modeb(primary_carbon(+drug,#primary_carbon)) modeb(resonant_count(+drug,#resonant_count)) modeb(rotatable_bondcount(+drug,#rotatable_bondcount)) modeb(ringbondcount(+drug,#ringbondcount)) modeb(hetero_ringcount(+drug,#hetero_ringcount)) modeb(fused_ringcount(+drug,#fused_ringcount)) modeb(fusedaromatic_ringcount(+drug, #fusedaromatic_ringcount)) modeb(fusedaliphatic_ringcount(+drug, #fusedaliphatic_ringcount)) modeb(chiralcenter_count(+drug,#chiralcenter_count)) modeb(asymmetric_atomcount(+drug, #asymmetric_atomcount)) modeb(heteroaromatic_ringcount(+drug, #heteroaromatic_ringcount)) modeb(ringcount(+drug,#ringcount)) modeb(hetero_ringcount(+drug,#hetero_ringcount)) modeb(heteroaromatic_ringcount(+drug, #heteroaromatic_ringcount)) modeb(fused_ringcount(+drug,#fused_ringcount)) modeb(fusedaromatic_ringcount(+drug, #fusedaromatic_ringcount)) modeb(fusedaliphatic_ringcount(+drug, #fusedaliphatic_ringcount)) modeb(atLeastOneOfFuncGroups(+drug,#funcgroup)) </pre>	<pre> modeb(pharmacophore_fingerprint(+drug,-pf,-lp,-frq)) modeb(50,gtPharmacophoreArg3(+frq,#frq)) modeb(50,ltPharmacophoreArg3(+frq,#frq)) modeb(gtPharmacophoreArg2(+lp,#lp)) modeb(ltPharmacophoreArg2(+lp,#lp)) modeb(50,pharmacophore_group(+pf,#pharma_group)) modeb(pharmacophore_fingerprint(+drug,#pf,-lp,-frq)) modeb(pharmacophore_fingerprint(+drug,#pf,#lp,-frq)) modeb(pharmacophore_fingerprint(+drug,#pf,-lp,#frq)) </pre>
<pre> modeb(chemical_fingerprint(+drug,-chemical_fingerprint)) modeb(80,fp_group(+chemical_fingerprint, #chemical_fpgrupp)) modeb(80,fp_group(+chemical_fingerprint, #chemical_fpgrupp)) </pre>	<pre> modeb(tetrahedral_stereoisomer_count(+drug, -tetrahedral_stereoisomer_count)) modeb(inRangeStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(gteqStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(lteqStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(inRangeStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(gteqStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(lteqStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) </pre>
<pre> modeb(tautomer_count(+drug,-tautomer_count)) modeb(inRangeTaut(+tautomer_count,#tautomer_count, #tautomer_count)) modeb(gteqTaut(+tautomer_count,#tautomer_count)) modeb(lteqTaut(+tautomer_count,#tautomer_count)) modeb(inRangeTaut(+tautomer_count,#tautomer_count, #tautomer_count)) modeb(gteqTaut(+tautomer_count,#tautomer_count)) modeb(lteqTaut(+tautomer_count,#tautomer_count)) </pre>	<pre> modeb(wiener_polarity(+drug,-wiener_polarity)) modeb(inRangeWien(+wiener_polarity,#wiener_polarity, #wiener_polarity)) modeb(gteqWien(+wiener_polarity,#wiener_polarity)) modeb(lteqWien(+wiener_polarity,#wiener_polarity)) modeb(inRangeWien(+wiener_polarity,#wiener_polarity, #wiener_polarity)) modeb(gteqWien(+wiener_polarity,#wiener_polarity)) modeb(lteqWien(+wiener_polarity,#wiener_polarity)) </pre>
<pre> modeb(dreiding_energy(+drug,-dreiding_energy)) modeb(inRangeDreid(+dreiding_energy,#dreiding_energy, #dreiding_energy)) modeb(gteqDreid(+dreiding_energy,#dreiding_energy)) modeb(lteqDreid(+dreiding_energy,#dreiding_energy)) modeb(inRangeDreid(+dreiding_energy,#dreiding_energy, #dreiding_energy)) modeb(gteqDreid(+dreiding_energy,#dreiding_energy)) modeb(lteqDreid(+dreiding_energy,#dreiding_energy)) </pre>	<pre> modeb(bcut(+drug,-bcut)) modeb(inRangeBcut(+bcut,#bcut,#bcut)) modeb(gteqBcut(+bcut,#bcut)) modeb(lteqBcut(+bcut,#bcut)) modeb(inRangeBcut(+bcut,#bcut,#bcut)) modeb(gteqBcut(+bcut,#bcut)) modeb(lteqBcut(+bcut,#bcut)) </pre>
<pre> modeb(quaternary_carbon(+drug,-quaternary_carbon)) modeb(inRangeQuatC(+quaternary_carbon, #quaternary_carbon,#quaternary_carbon)) modeb(gteqQuatC(+quaternary_carbon, #quaternary_carbon)) modeb(lteqQuatC(+quaternary_carbon, #quaternary_carbon)) modeb(inRangeQuatC(+quaternary_carbon, #quaternary_carbon,#quaternary_carbon)) modeb(gteqQuatC(+quaternary_carbon, #quaternary_carbon)) modeb(lteqQuatC(+quaternary_carbon, #quaternary_carbon)) </pre>	<pre> modeb(tertiary_carbon(+drug,-tertiary_carbon)) modeb(inRangeTercC(+tertiary_carbon, #tertiary_carbon,#tertiary_carbon)) modeb(gteqTercC(+tertiary_carbon,#tertiary_carbon)) modeb(lteqTercC(+tertiary_carbon,#tertiary_carbon)) modeb(inRangeTercC(+tertiary_carbon, #tertiary_carbon,#tertiary_carbon)) modeb(gteqTercC(+tertiary_carbon,#tertiary_carbon)) modeb(lteqTercC(+tertiary_carbon,#tertiary_carbon)) </pre>

Tabela B.2: Parte 1 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados CPDBAS. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

<pre> modeb(secondary_carbon(+drug,-secondary_carbon)) modeb(inRangeSecC(+secondary_carbon, #secondary_carbon,#secondary_carbon)) modeb(gteqSecC(+secondary_carbon,#secondary_carbon)) modeb(lteqSecC(+secondary_carbon,#secondary_carbon)) modeb(inRangeSecC(+secondary_carbon, #secondary_carbon,#secondary_carbon)) modeb(gteqSecC(+secondary_carbon,#secondary_carbon)) modeb(lteqSecC(+secondary_carbon,#secondary_carbon)) </pre>	<pre> modeb(primary_carbon(+drug,-primary_carbon)) modeb(inRangePrimC(+primary_carbon,#primary_carbon, #primary_carbon)) modeb(gteqPrimC(+primary_carbon,#primary_carbon)) modeb(lteqPrimC(+primary_carbon,#primary_carbon)) modeb(inRangePrimC(+primary_carbon,#primary_carbon, #primary_carbon)) modeb(gteqPrimC(+primary_carbon,#primary_carbon)) modeb(lteqPrimC(+primary_carbon,#primary_carbon)) </pre>
<pre> modeb(resonant_count(+drug,-resonant_count)) modeb(inRangeRescount(+resonant_count, #resonant_count,#resonant_count)) modeb(gteqRescount(+resonant_count,#resonant_count)) modeb(lteqRescount(+resonant_count,#resonant_count)) modeb(inRangeRescount(+resonant_count, #resonant_count,#resonant_count)) modeb(gteqRescount(+resonant_count,#resonant_count)) modeb(lteqRescount(+resonant_count,#resonant_count)) </pre>	<pre> modeb(rotatable_bondcount(+drug,-rotatable_bondcount)) modeb(inRangeRotBond(+rotatable_bondcount, #rotatable_bondcount,#rotatable_bondcount)) modeb(gteqRotBond(+rotatable_bondcount, #rotatable_bondcount)) modeb(lteqRotBond(+rotatable_bondcount, #rotatable_bondcount)) modeb(inRangeRotBond(+rotatable_bondcount, #rotatable_bondcount,#rotatable_bondcount)) modeb(gteqRotBond(+rotatable_bondcount, #rotatable_bondcount)) modeb(lteqRotBond(+rotatable_bondcount, #rotatable_bondcount)) </pre>
<pre> modeb(ringbondcount(+drug,-ringbondcount)) modeb(inRangeRingbc(+ringbondcount,#ringbondcount, #ringbondcount)) modeb(gteqRingbc(+ringbondcount,#ringbondcount)) modeb(lteqRingbc(+ringbondcount,#ringbondcount)) modeb(inRangeRingbc(+ringbondcount,#ringbondcount, #ringbondcount)) modeb(gteqRingbc(+ringbondcount,#ringbondcount)) modeb(lteqRingbc(+ringbondcount,#ringbondcount)) </pre>	<pre> modeb(hetero_ringcount(+drug,-hetero_ringcount)) modeb(inRangeHetRing(+hetero_ringcount, #hetero_ringcount,#hetero_ringcount)) modeb(gteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(lteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(inRangeHetRing(+hetero_ringcount, #hetero_ringcount,#hetero_ringcount)) modeb(gteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(lteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(inRangeHetRing(+hetero_ringcount, #hetero_ringcount,#hetero_ringcount)) modeb(gteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(lteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(hetero_ringcount(+drug,-hetero_ringcount)) </pre>
<pre> modeb(fused_ringcount(+drug,-fused_ringcount)) modeb(inRangeFusedRing(+fused_ringcount, #fused_ringcount,#fused_ringcount)) modeb(gteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(lteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(inRangeFusedRing(+fused_ringcount, #fused_ringcount,#fused_ringcount)) modeb(gteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(lteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(inRangeFusedRing(+fused_ringcount, #fused_ringcount,#fused_ringcount)) modeb(gteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(lteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(fused_ringcount(+drug,-fused_ringcount)) </pre>	<pre> modeb(fusedaromatic_ringcount(+drug,- fusedaromatic_ringcount)) modeb(inRangeFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount,#fusedaromatic_ringcount)) modeb(gteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(lteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(inRangeFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount,#fusedaromatic_ringcount)) modeb(gteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(lteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(inRangeFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount,#fusedaromatic_ringcount)) modeb(gteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(lteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(fusedaromatic_ringcount(+drug,- fusedaromatic_ringcount)) </pre>
<pre> modeb(fusedaliphatic_ringcount(+drug,- fusedaliphatic_ringcount)) modeb(inRangeFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount,#fusedaliphatic_ringcount)) modeb(gteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(lteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(inRangeFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount,#fusedaliphatic_ringcount)) modeb(gteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(lteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(inRangeFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount,#fusedaliphatic_ringcount)) modeb(gteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(lteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(fusedaliphatic_ringcount(+drug,- fusedaliphatic_ringcount)) </pre>	<pre> modeb(chiralcenter_count(+drug,-chiralcenter_count)) modeb(inRangeChiral(+chiralcenter_count, #chiralcenter_count,#chiralcenter_count)) modeb(gteqChiral(+chiralcenter_count,#chiralcenter_count)) modeb(lteqChiral(+chiralcenter_count,#chiralcenter_count)) modeb(inRangeChiral(+chiralcenter_count, #chiralcenter_count,#chiralcenter_count)) modeb(gteqChiral(+chiralcenter_count,#chiralcenter_count)) modeb(lteqChiral(+chiralcenter_count,#chiralcenter_count)) </pre>

Tabela B.3: Parte 2 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados *CPDBAS*. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

modeb(heavy(+drug,-heavy)) modeb(inRangeHeavy(+heavy,#heavy,#heavy)) modeb(gteqHeavy(+heavy,#heavy)) modeb(lteqHeavy(+heavy,#heavy)) modeb(inRangeHeavy(+heavy,#heavy,#heavy)) modeb(gteqHeavy(+heavy,#heavy)) modeb(lteqHeavy(+heavy,#heavy))	modeb(mass(+drug,-mass)) modeb(inRangeMass(+mass,#mass,#mass)) modeb(gteqMass(+mass,#mass)) modeb(lteqMass(+mass,#mass)) modeb(inRangeMass(+mass,#mass,#mass)) modeb(gteqMass(+mass,#mass)) modeb(lteqMass(+mass,#mass))
modeb(logd(+drug,-logd)) modeb(inRangeLogD(+logd,#logd,#logd)) modeb(gteqLogD(+logd,#logd)) modeb(lteqLogD(+logd,#logd)) modeb(inRangeLogD(+logd,#logd,#logd)) modeb(gteqLogD(+logd,#logd)) modeb(lteqLogD(+logd,#logd))	modeb(refractivity(+drug,-refractivity)) modeb(inRangeRefract(+refractivity,#refractivity, #refractivity)) modeb(gteqRefract(+refractivity,#refractivity)) modeb(lteqRefract(+refractivity,#refractivity)) modeb(inRangeRefract(+refractivity,#refractivity, #refractivity)) modeb(gteqRefract(+refractivity,#refractivity)) modeb(lteqRefract(+refractivity,#refractivity))
modeb(msa(+drug,-msa)) modeb(inRangeMsa(+msa,#msa,#msa)) modeb(gteqMsa(+msa,#msa)) modeb(lteqMsa(+msa,#msa)) modeb(inRangeMsa(+msa,#msa,#msa)) modeb(gteqMsa(+msa,#msa)) modeb(lteqMsa(+msa,#msa))	modeb(pi_energy(+drug,-pi_energy)) modeb(inRangePIen(+pi_energy,#pi_energy,#pi_energy)) modeb(gteqPIen(+pi_energy,#pi_energy)) modeb(lteqPIen(+pi_energy,#pi_energy)) modeb(inRangePIen(+pi_energy,#pi_energy,#pi_energy)) modeb(gteqPIen(+pi_energy,#pi_energy)) modeb(lteqPIen(+pi_energy,#pi_energy))
modeb(mol_polarizability(+drug,-mol_polarizability)) modeb(inRangePolariz(+mol_polarizability,#mol_polarizability, #mol_polarizability,#mol_polarizability)) modeb(gteqPolariz(+mol_polarizability,#mol_polarizability)) modeb(lteqPolariz(+mol_polarizability,#mol_polarizability)) modeb(inRangePolariz(+mol_polarizability,#mol_polarizability, #mol_polarizability,#mol_polarizability)) modeb(gteqPolariz(+mol_polarizability,#mol_polarizability)) modeb(lteqPolariz(+mol_polarizability,#mol_polarizability))	modeb(asymmetric_atomcount(+drug,-asymmetric_atomcount)) modeb(inRangeAsatcount(+asymmetric_atomcount, #asymmetric_atomcount,#asymmetric_atomcount)) modeb(gteqAsatcount(+asymmetric_atomcount, #asymmetric_atomcount)) modeb(lteqAsatcount(+asymmetric_atomcount, #asymmetric_atomcount)) modeb(inRangeAsatcount(+asymmetric_atomcount, #asymmetric_atomcount,#asymmetric_atomcount)) modeb(gteqAsatcount(+asymmetric_atomcount, #asymmetric_atomcount)) modeb(lteqAsatcount(+asymmetric_atomcount, #asymmetric_atomcount))
modeb(heteroaromatic_ringcount(+drug,-heteroaromatic_ringcount)) modeb(inRangeHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount,#heteroaromatic_ringcount)) modeb(gteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(lteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(inRangeHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount,#heteroaromatic_ringcount)) modeb(gteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(lteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(inRangeHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount,#heteroaromatic_ringcount)) modeb(gteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(lteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(heteroaromatic_ringcount(+drug,-heteroaromatic_ringcount))	modeb(ringcount(+drug,-ringcount)) modeb(inRangeRcount(+ringcount,#ringcount,#ringcount)) modeb(gteqRcount(+ringcount,#ringcount)) modeb(lteqRcount(+ringcount,#ringcount)) modeb(inRangeRcount(+ringcount,#ringcount,#ringcount)) modeb(gteqRcount(+ringcount,#ringcount)) modeb(lteqRcount(+ringcount,#ringcount))

Tabela B.4: Parte 3 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados CPDBAS. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

modeb(tpsa(+drug,-tpsa)) modeb(inRangeTpsa(+tpsa,#tpsa,#tpsa)) modeb(gteqTpsa(+tpsa,#tpsa)) modeb(lteqTpsa(+tpsa,#tpsa)) modeb(inRangeTpsa(+tpsa,#tpsa,#tpsa)) modeb(gteqTpsa(+tpsa,#tpsa)) modeb(lteqTpsa(+tpsa,#tpsa))	modeb(structure_molecularweight(+drug,-molecularweight)) modeb(inRangeStrMolWeight(+molecularweight,#molecularweight,#molecularweight)) modeb(gteqStrMolWeight(+molecularweight,#molecularweight)) modeb(lteqStrMolWeight(+molecularweight,#molecularweight))
modeb(account(+drug,-account)) modeb(inRangeAccount(+account,#account,#account)) modeb(gteqAccount(+account,#account)) modeb(lteqAccount(+account,#account)) modeb(inRangeAccount(+account,#account,#account)) modeb(gteqAccount(+account,#account)) modeb(lteqAccount(+account,#account))	modeb(doncount(+drug,-doncount)) modeb(inRangeDcount(+doncount,#doncount,#doncount)) modeb(gteqDcount(+doncount,#doncount)) modeb(lteqDcount(+doncount,#doncount)) modeb(inRangeDcount(+doncount,#doncount,#doncount)) modeb(gteqDcount(+doncount,#doncount)) modeb(lteqDcount(+doncount,#doncount))
modeb(vdwsa(+drug,-vdwsa)) modeb(inRangeVdwsa(+vdwsa,#vdwsa,#vdwsa)) modeb(gteqVdwsa(+vdwsa,#vdwsa)) modeb(lteqVdwsa(+vdwsa,#vdwsa)) modeb(inRangeVdwsa(+vdwsa,#vdwsa,#vdwsa)) modeb(gteqVdwsa(+vdwsa,#vdwsa)) modeb(lteqVdwsa(+vdwsa,#vdwsa))	modeb(logp(+drug,-logp)) modeb(inRangeLogP(+logp,#logp,#logp)) modeb(gteqLogP(+logp,#logp)) modeb(lteqLogP(+logp,#logp)) modeb(inRangeLogP(+logp,#logp,#logp)) modeb(gteqLogP(+logp,#logp)) modeb(lteqLogP(+logp,#logp))
modeb(phenyl(+drug,-bindAtom,-atomList)) modeb(ketone(+drug,-bindAtom,-atomList)) modeb(hydroxyl(+drug,-bindAtom,-atomList)) modeb(cyano(+drug,-bindAtom,-atomList)) modeb(carboxylic_acid(+drug,-bindAtom,-atomList)) modeb(aldehyde(+drug,-bindAtom,-atomList)) modeb(amide(+drug,-bindAtom,-atomList)) modeb(nh2(+drug,-bindAtom,-atomList)) modeb(methoxy(+drug,-bindAtom,-atomList)) modeb(tButyl(+drug,-bindAtom,-atomList)) modeb(methyl(+drug,-bindAtom,-atomList))	

Tabela B.5: Parte 4 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados *CPDBAS*. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

modeh(active(+drug))	
<pre> modeb(pharmacophore_fingerprint(+drug,#pf,#lp,#frq)) modeb(chemical_fingerprint(+drug,#chemical_fingerprint)) modeb(tetrahedral_stereoisomer_count(+drug, #tetrahedral_stereoisomer_count)) modeb(tautomer_count(+drug,#tautomer_count)) modeb(wiener_polarity(+drug,#wiener_polarity)) modeb(quaternary_carbon(+drug,#quaternary_carbon)) modeb(tertiary_carbon(+drug,#tertiary_carbon)) modeb(secondary_carbon(+drug,#secondary_carbon)) modeb(primary_carbon(+drug,#primary_carbon)) modeb(resonant_count(+drug,#resonant_count)) modeb(rotatable_bondcount(+drug,#rotatable_bondcount)) modeb(ringbondcount(+drug,#ringbondcount)) modeb(hetero_ringcount(+drug,#hetero_ringcount)) modeb(fused_ringcount(+drug,#fused_ringcount)) modeb(fusedaromatic_ringcount(+drug, #fusedaromatic_ringcount)) modeb(fusedaliphatic_ringcount(+drug, #fusedaliphatic_ringcount)) modeb(chiralcenter_count(+drug,#chiralcenter_count)) modeb(asymmetric_atomcount(+drug, #asymmetric_atomcount)) modeb(heteroaromatic_ringcount(+drug, #heteroaromatic_ringcount)) modeb(ringcount(+drug,#ringcount)) modeb(hetero_ringcount(+drug,#hetero_ringcount)) modeb(heteroaromatic_ringcount(+drug, #heteroaromatic_ringcount)) modeb(fused_ringcount(+drug,#fused_ringcount)) modeb(fusedaromatic_ringcount(+drug, #fusedaromatic_ringcount)) modeb(fusedaliphatic_ringcount(+drug, #fusedaliphatic_ringcount)) modeb(atLeastOneOfFuncGroups(+drug,#funcgroup)) </pre>	<pre> modeb(pharmacophore_fingerprint(+drug,-pf,-lp,-frq)) modeb(50,gtPharmacophoreArg3(+frq,#frq)) modeb(50,ltPharmacophoreArg3(+frq,#frq)) modeb(gtPharmacophoreArg2(+lp,#lp)) modeb(ltPharmacophoreArg2(+lp,#lp)) modeb(50,pharmacophore_group(+pf,#pharma_group)) modeb(pharmacophore_fingerprint(+drug,#pf,-lp,-frq)) modeb(pharmacophore_fingerprint(+drug,#pf,#lp,-frq)) modeb(pharmacophore_fingerprint(+drug,#pf,-lp,#frq)) </pre>
<pre> modeb(chemical_fingerprint(+drug,-chemical_fingerprint)) modeb(80,fp_group(+chemical_fingerprint, #chemical_fpgroup)) modeb(80,fp_group(+chemical_fingerprint, #chemical_fpgroup)) </pre>	<pre> modeb(tetrahedral_stereoisomer_count(+drug, -tetrahedral_stereoisomer_count)) modeb(inRangeStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(gteqStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(lteqStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(inRangeStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(gteqStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) modeb(lteqStero(+tetrahedral_stereoisomer_count, #tetrahedral_stereoisomer_count)) </pre>
<pre> modeb(tautomer_count(+drug,-tautomer_count)) modeb(inRangeTaut(+tautomer_count,#tautomer_count, #tautomer_count)) modeb(gteqTaut(+tautomer_count,#tautomer_count)) modeb(lteqTaut(+tautomer_count,#tautomer_count)) modeb(inRangeTaut(+tautomer_count,#tautomer_count, #tautomer_count)) modeb(gteqTaut(+tautomer_count,#tautomer_count)) modeb(lteqTaut(+tautomer_count,#tautomer_count)) </pre>	<pre> modeb(wiener_polarity(+drug,-wiener_polarity)) modeb(inRangeWien(+wiener_polarity,#wiener_polarity, #wiener_polarity)) modeb(gteqWien(+wiener_polarity,#wiener_polarity)) modeb(lteqWien(+wiener_polarity,#wiener_polarity)) modeb(inRangeWien(+wiener_polarity,#wiener_polarity, #wiener_polarity)) modeb(gteqWien(+wiener_polarity,#wiener_polarity)) modeb(lteqWien(+wiener_polarity,#wiener_polarity)) </pre>
<pre> modeb(dreiding_energy(+drug,-dreiding_energy)) modeb(inRangeDreid(+dreiding_energy,#dreiding_energy, #dreiding_energy)) modeb(gteqDreid(+dreiding_energy,#dreiding_energy)) modeb(lteqDreid(+dreiding_energy,#dreiding_energy)) modeb(inRangeDreid(+dreiding_energy,#dreiding_energy, #dreiding_energy)) modeb(gteqDreid(+dreiding_energy,#dreiding_energy)) modeb(lteqDreid(+dreiding_energy,#dreiding_energy)) </pre>	<pre> modeb(bcut(+drug,-bcut)) modeb(inRangeBcut(+bcut,#bcut,#bcut)) modeb(gteqBcut(+bcut,#bcut)) modeb(lteqBcut(+bcut,#bcut)) modeb(inRangeBcut(+bcut,#bcut,#bcut)) modeb(gteqBcut(+bcut,#bcut)) modeb(lteqBcut(+bcut,#bcut)) </pre>
<pre> modeb(quaternary_carbon(+drug,-quaternary_carbon)) modeb(inRangeQuatC(+quaternary_carbon, #quaternary_carbon,#quaternary_carbon)) modeb(gteqQuatC(+quaternary_carbon, #quaternary_carbon)) modeb(lteqQuatC(+quaternary_carbon, #quaternary_carbon)) modeb(inRangeQuatC(+quaternary_carbon, #quaternary_carbon,#quaternary_carbon)) modeb(gteqQuatC(+quaternary_carbon, #quaternary_carbon)) modeb(lteqQuatC(+quaternary_carbon, #quaternary_carbon)) </pre>	<pre> modeb(tertiary_carbon(+drug,-tertiary_carbon)) modeb(inRangeTercC(+tertiary_carbon, #tertiary_carbon,#tertiary_carbon)) modeb(gteqTercC(+tertiary_carbon,#tertiary_carbon)) modeb(lteqTercC(+tertiary_carbon,#tertiary_carbon)) modeb(inRangeTercC(+tertiary_carbon, #tertiary_carbon,#tertiary_carbon)) modeb(gteqTercC(+tertiary_carbon,#tertiary_carbon)) modeb(lteqTercC(+tertiary_carbon,#tertiary_carbon)) </pre>

Tabela B.6: Parte 1 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados *DBPCAN*. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

<pre> modeb(secondary_carbon(+drug,-secondary_carbon)) modeb(inRangeSecC(+secondary_carbon, #secondary_carbon,#secondary_carbon)) modeb(gteqSecC(+secondary_carbon,#secondary_carbon)) modeb(lteqSecC(+secondary_carbon,#secondary_carbon)) modeb(inRangeSecC(+secondary_carbon, #secondary_carbon,#secondary_carbon)) modeb(gteqSecC(+secondary_carbon,#secondary_carbon)) modeb(lteqSecC(+secondary_carbon,#secondary_carbon)) </pre>	<pre> modeb(primary_carbon(+drug,-primary_carbon)) modeb(inRangePrimC(+primary_carbon,#primary_carbon, #primary_carbon)) modeb(gteqPrimC(+primary_carbon,#primary_carbon)) modeb(lteqPrimC(+primary_carbon,#primary_carbon)) modeb(inRangePrimC(+primary_carbon,#primary_carbon, #primary_carbon)) modeb(gteqPrimC(+primary_carbon,#primary_carbon)) modeb(lteqPrimC(+primary_carbon,#primary_carbon)) </pre>
<pre> modeb(resonant_count(+drug,-resonant_count)) modeb(inRangeRescount(+resonant_count, #resonant_count,#resonant_count)) modeb(gteqRescount(+resonant_count,#resonant_count)) modeb(lteqRescount(+resonant_count,#resonant_count)) modeb(inRangeRescount(+resonant_count, #resonant_count,#resonant_count)) modeb(gteqRescount(+resonant_count,#resonant_count)) modeb(lteqRescount(+resonant_count,#resonant_count)) </pre>	<pre> modeb(rotatable_bondcount(+drug,-rotatable_bondcount)) modeb(inRangeRotBond(+rotatable_bondcount, #rotatable_bondcount,#rotatable_bondcount)) modeb(gteqRotBond(+rotatable_bondcount, #rotatable_bondcount)) modeb(lteqRotBond(+rotatable_bondcount, #rotatable_bondcount)) modeb(inRangeRotBond(+rotatable_bondcount, #rotatable_bondcount,#rotatable_bondcount)) modeb(gteqRotBond(+rotatable_bondcount, #rotatable_bondcount)) modeb(lteqRotBond(+rotatable_bondcount, #rotatable_bondcount)) </pre>
<pre> modeb(ringbondcount(+drug,-ringbondcount)) modeb(inRangeRingbc(+ringbondcount,#ringbondcount, #ringbondcount)) modeb(gteqRingbc(+ringbondcount,#ringbondcount)) modeb(lteqRingbc(+ringbondcount,#ringbondcount)) modeb(inRangeRingbc(+ringbondcount,#ringbondcount, #ringbondcount)) modeb(gteqRingbc(+ringbondcount,#ringbondcount)) modeb(lteqRingbc(+ringbondcount,#ringbondcount)) </pre>	<pre> modeb(hetero_ringcount(+drug,-hetero_ringcount)) modeb(inRangeHetRing(+hetero_ringcount, #hetero_ringcount,#hetero_ringcount)) modeb(gteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(lteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(inRangeHetRing(+hetero_ringcount, #hetero_ringcount,#hetero_ringcount)) modeb(gteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(lteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(inRangeHetRing(+hetero_ringcount, #hetero_ringcount,#hetero_ringcount)) modeb(gteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(lteqHetRing(+hetero_ringcount,#hetero_ringcount)) modeb(hetero_ringcount(+drug,-hetero_ringcount)) </pre>
<pre> modeb(fused_ringcount(+drug,-fused_ringcount)) modeb(inRangeFusedRing(+fused_ringcount, #fused_ringcount,#fused_ringcount)) modeb(gteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(lteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(inRangeFusedRing(+fused_ringcount, #fused_ringcount,#fused_ringcount)) modeb(gteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(lteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(inRangeFusedRing(+fused_ringcount, #fused_ringcount,#fused_ringcount)) modeb(gteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(lteqFusedRing(+fused_ringcount,#fused_ringcount)) modeb(fused_ringcount(+drug,-fused_ringcount)) </pre>	<pre> modeb(fusedaromatic_ringcount(+drug,- fusedaromatic_ringcount)) modeb(inRangeFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount,#fusedaromatic_ringcount)) modeb(gteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(lteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(inRangeFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount,#fusedaromatic_ringcount)) modeb(gteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(lteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(inRangeFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount,#fusedaromatic_ringcount)) modeb(gteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(lteqFusedAro(+fusedaromatic_ringcount, #fusedaromatic_ringcount)) modeb(fusedaromatic_ringcount(+drug,- fusedaromatic_ringcount)) </pre>
<pre> modeb(fusedaliphatic_ringcount(+drug,- fusedaliphatic_ringcount)) modeb(inRangeFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount,#fusedaliphatic_ringcount)) modeb(gteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(lteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(inRangeFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount,#fusedaliphatic_ringcount)) modeb(gteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(lteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(inRangeFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount,#fusedaliphatic_ringcount)) modeb(gteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(lteqFusedAli(+fusedaliphatic_ringcount, #fusedaliphatic_ringcount)) modeb(fusedaliphatic_ringcount(+drug,- fusedaliphatic_ringcount)) </pre>	<pre> modeb(chiralcenter_count(+drug,-chiralcenter_count)) modeb(inRangeChiral(+chiralcenter_count, #chiralcenter_count,#chiralcenter_count)) modeb(gteqChiral(+chiralcenter_count,#chiralcenter_count)) modeb(lteqChiral(+chiralcenter_count,#chiralcenter_count)) modeb(inRangeChiral(+chiralcenter_count, #chiralcenter_count,#chiralcenter_count)) modeb(gteqChiral(+chiralcenter_count,#chiralcenter_count)) modeb(lteqChiral(+chiralcenter_count,#chiralcenter_count)) </pre>

Tabela B.7: Parte 2 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados *DBPCAN*. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

modeb(heavy(+drug,-heavy)) modeb(inRangeHeavy(+heavy,#heavy,#heavy)) modeb(gteqHeavy(+heavy,#heavy)) modeb(lteqHeavy(+heavy,#heavy)) modeb(inRangeHeavy(+heavy,#heavy,#heavy)) modeb(gteqHeavy(+heavy,#heavy)) modeb(lteqHeavy(+heavy,#heavy))	modeb(mass(+drug,-mass)) modeb(inRangeMass(+mass,#mass,#mass)) modeb(gteqMass(+mass,#mass)) modeb(lteqMass(+mass,#mass)) modeb(inRangeMass(+mass,#mass,#mass)) modeb(gteqMass(+mass,#mass)) modeb(lteqMass(+mass,#mass))
modeb(logd(+drug,-logd)) modeb(inRangeLogD(+logd,#logd,#logd)) modeb(gteqLogD(+logd,#logd)) modeb(lteqLogD(+logd,#logd)) modeb(inRangeLogD(+logd,#logd,#logd)) modeb(gteqLogD(+logd,#logd)) modeb(lteqLogD(+logd,#logd))	modeb(refractivity(+drug,-refractivity)) modeb(inRangeRefract(+refractivity,#refractivity, #refractivity)) modeb(gteqRefract(+refractivity,#refractivity)) modeb(lteqRefract(+refractivity,#refractivity)) modeb(inRangeRefract(+refractivity,#refractivity, #refractivity)) modeb(gteqRefract(+refractivity,#refractivity)) modeb(lteqRefract(+refractivity,#refractivity))
modeb(msa(+drug,-msa)) modeb(inRangeMsa(+msa,#msa,#msa)) modeb(gteqMsa(+msa,#msa)) modeb(lteqMsa(+msa,#msa)) modeb(inRangeMsa(+msa,#msa,#msa)) modeb(gteqMsa(+msa,#msa)) modeb(lteqMsa(+msa,#msa))	modeb(pi_energy(+drug,-pi_energy)) modeb(inRangePIen(+pi_energy,#pi_energy,#pi_energy)) modeb(gteqPIen(+pi_energy,#pi_energy)) modeb(lteqPIen(+pi_energy,#pi_energy)) modeb(inRangePIen(+pi_energy,#pi_energy,#pi_energy)) modeb(gteqPIen(+pi_energy,#pi_energy)) modeb(lteqPIen(+pi_energy,#pi_energy))
modeb(mol_polarizability(+drug,-mol_polarizability)) modeb(inRangePolariz(+mol_polarizability,#mol_polarizability, #mol_polarizability,#mol_polarizability)) modeb(gteqPolariz(+mol_polarizability,#mol_polarizability)) modeb(lteqPolariz(+mol_polarizability,#mol_polarizability)) modeb(inRangePolariz(+mol_polarizability,#mol_polarizability, #mol_polarizability,#mol_polarizability)) modeb(gteqPolariz(+mol_polarizability,#mol_polarizability)) modeb(lteqPolariz(+mol_polarizability,#mol_polarizability))	modeb(asymmetric_atomcount(+drug,-asymmetric_atomcount)) modeb(inRangeAsatcount(+asymmetric_atomcount, #asymmetric_atomcount,#asymmetric_atomcount)) modeb(gteqAsatcount(+asymmetric_atomcount, #asymmetric_atomcount)) modeb(lteqAsatcount(+asymmetric_atomcount, #asymmetric_atomcount)) modeb(inRangeAsatcount(+asymmetric_atomcount, #asymmetric_atomcount,#asymmetric_atomcount)) modeb(gteqAsatcount(+asymmetric_atomcount, #asymmetric_atomcount)) modeb(lteqAsatcount(+asymmetric_atomcount, #asymmetric_atomcount))
modeb(heteroaromatic_ringcount(+drug,-heteroaromatic_ringcount)) modeb(inRangeHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount,#heteroaromatic_ringcount)) modeb(gteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(lteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(inRangeHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount,#heteroaromatic_ringcount)) modeb(gteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(lteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(inRangeHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount,#heteroaromatic_ringcount)) modeb(gteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(lteqHARcount(+heteroaromatic_ringcount, #heteroaromatic_ringcount)) modeb(heteroaromatic_ringcount(+drug,-heteroaromatic_ringcount))	modeb(ringcount(+drug,-ringcount)) modeb(inRangeRcount(+ringcount,#ringcount,#ringcount)) modeb(gteqRcount(+ringcount,#ringcount)) modeb(lteqRcount(+ringcount,#ringcount)) modeb(inRangeRcount(+ringcount,#ringcount,#ringcount)) modeb(gteqRcount(+ringcount,#ringcount)) modeb(lteqRcount(+ringcount,#ringcount))

Tabela B.8: Parte 3 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados *DBPCAN*. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

modeb(tpsa(+drug,-tpsa)) modeb(inRangeTpsa(+tpsa,#tpsa,#tpsa)) modeb(gteqTpsa(+tpsa,#tpsa)) modeb(lteqTpsa(+tpsa,#tpsa)) modeb(inRangeTpsa(+tpsa,#tpsa,#tpsa)) modeb(gteqTpsa(+tpsa,#tpsa)) modeb(lteqTpsa(+tpsa,#tpsa))	modeb(structure_molecularweight(+drug,-molecularweight)) modeb(inRangeStrMolWeight(+molecularweight,#molecularweight,#molecularweight)) modeb(gteqStrMolWeight(+molecularweight,#molecularweight)) modeb(lteqStrMolWeight(+molecularweight,#molecularweight))
modeb(account(+drug,-account)) modeb(inRangeAccount(+account,#account,#account)) modeb(gteqAccount(+account,#account)) modeb(lteqAccount(+account,#account)) modeb(inRangeAccount(+account,#account,#account)) modeb(gteqAccount(+account,#account)) modeb(lteqAccount(+account,#account))	modeb(doncount(+drug,-doncount)) modeb(inRangeDcount(+doncount,#doncount,#doncount)) modeb(gteqDcount(+doncount,#doncount)) modeb(lteqDcount(+doncount,#doncount)) modeb(inRangeDcount(+doncount,#doncount,#doncount)) modeb(gteqDcount(+doncount,#doncount)) modeb(lteqDcount(+doncount,#doncount))
modeb(vdwsa(+drug,-vdwsa)) modeb(inRangeVdwsa(+vdwsa,#vdwsa,#vdwsa)) modeb(gteqVdwsa(+vdwsa,#vdwsa)) modeb(lteqVdwsa(+vdwsa,#vdwsa)) modeb(inRangeVdwsa(+vdwsa,#vdwsa,#vdwsa)) modeb(gteqVdwsa(+vdwsa,#vdwsa)) modeb(lteqVdwsa(+vdwsa,#vdwsa))	modeb(logp(+drug,-logp)) modeb(inRangeLogP(+logp,#logp,#logp)) modeb(gteqLogP(+logp,#logp)) modeb(lteqLogP(+logp,#logp)) modeb(inRangeLogP(+logp,#logp,#logp)) modeb(gteqLogP(+logp,#logp)) modeb(lteqLogP(+logp,#logp))
modeb(phenyl(+drug,-bindAtom,-atomList)) modeb(ketone(+drug,-bindAtom,-atomList)) modeb(hydroxyl(+drug,-bindAtom,-atomList)) modeb(cyano(+drug,-bindAtom,-atomList)) modeb(carboxylic_acid(+drug,-bindAtom,-atomList)) modeb(aldehyde(+drug,-bindAtom,-atomList)) modeb(acetyl(+drug,-bindAtom,-atomList)) modeb(methyl(+drug,-bindAtom,-atomList))	

Tabela B.9: Parte 4 - Ilhas identificadas pelo Algoritmo 1 no conjunto de dados *DBPCAN*. O *recall number* não está mostrado nas declarações de modo pois só os tipos são relevantes.

Apêndice C

Exemplos e Scripts

Listing C.1: *Script* principal para geração do código e dados na implementação de *feature subset selection* usando dados em formato arff e algoritmos do Weka.

```
#!/bin/csh

4  if ($#argv != 4) then
    echo "syntax: mkFeatureSelection classifier trainset tuneset depth"
    exit
  endif

9  ### System Settings
  set OUTPUT="output"
  set GETRESULTS_FILE="getResults"
  set JOBSDIR="jobsdir"
  set COMBINATIONS="attCombinations"
14 set ATT_FILTER = "weka.filters.unsupervised.attribute.Remove"

### Global variables
set CLASSIFIER = $1
echo " [Weka classifier is '$CLASSIFIER']"
19 set TRAINING_SET=$2
echo " [Training set file is '$TRAINING_SET']"
set TUNING_SET=$3
echo " [Tuning set file is '$TUNING_SET']"
set MAXDEPTH=$4 # depth limit for a breadth-first search on attributes
24 echo " [Depth limit is '$MAXDEPTH']"

# Check the information provided
# make sure $JOBSDIR directory exists
mkdir -p $JOBSDIR
29 \rm -r -f $JOBSDIR/* >& /dev/null

#####
#      G L O B A L      V A R s
```

```

34 #####
   set MEMOPTION = " -Xmx8192M "

#####
   # Preparing the cycle
39 #####

   # bf-selection $nAttributes $nAttributes $Attfile
   # prepare feature selection list of attributes to remove
   # initializations
44 @ nAttributes = `grep -i "@attribute" $TRAINING_SET | wc -l` - 1

#####
   # Feature Selection Cycle
49 #####

   pushd $JOBSDIR >& /dev/null

   cp -f ../$TRAINING_SET ../$TUNING_SET .
54   cp -r ../weka.jar .

   ../gencombinations.yap $nAttributes $MAXDEPTH $COMBINATIONS

   echo " [Feature selection step]"
59

   # no feature selection depth = 0
   echo "#!/bin/csh" > run
   echo "java $MEMOPTION -cp weka.jar $CLASSIFIER -t $TRAINING_SET -T $TUNING_SET > $OUTPUT" >>
       run
   chmod +x run
64   tar czvf feature0.tgz run weka.jar $TRAINING_SET $TUNING_SET >& /dev/null

   set depth=1
   set counter=1

69   # Feature subsetSelection MAIN cycle
   while ($depth <= $MAXDEPTH)

       echo " [Processing level $depth]"

74       set attSelectionList=`head - $depth $COMBINATIONS | tail -1`

       foreach attcombination ($attSelectionList)

           echo -n "."

79           java $MEMOPTION -cp weka.jar $ATT_FILTER -i $TRAINING_SET -o featureTrain${counter}.arff -R
               $attcombination
           java $MEMOPTION -cp weka.jar $ATT_FILTER -i $TUNING_SET -o featureTune${counter}.arff -R
               $attcombination

           echo "#!/bin/csh" > run
84           echo "java $MEMOPTION -cp weka.jar $CLASSIFIER -t featureTrain${counter}.arff -T
               featureTune${counter}.arff > $OUTPUT" >\
> run
               chmod +x run

```

```

tar czvf feature${counter}.tgz run weka.jar featureTrain${counter}.arff featureTune${
counter}.arff >& /dev/null
\rm -f featureTrain${counter}.arff featureTune${counter}.arff
89 @ counter++
end # of foreach of feature selectin level combination
echo ""
@ depth++
end # end of Feature Selection cycle
94

#####
#                               Final Results
#####

99 @ MAX= $counter - 1
#####
# Generate (by instantiation) the script that collects the results of
# each parameter combination and returns, fro each measure, the best
104 # value and the corresponding parameter combination
#####

cat ../getResults.model | sed {s/MAXVALUECONST/$MAX/} > $GETRESULTS_FILE
chmod +x $GETRESULTS_FILE
109 popd >& /dev/null
exit

```

Listing C.2: *Script* auxiliar para geração do código do nó que combina o resultado dos nós que correram em paralelo e dá o resultado final.

```
#!/bin/csh

4  ### Global variables

    set FILESTEM="output"
    set RESULTS="finalResult"
    set MAX=MAXVALUECONST

9

#####
#                               Final Results
#####

14  echo ""
    echo "                [Collecting Final Results]"
    echo ""

    set bestAcc=0
19  set bestCombination=""

    set i=0
    while ($i <= $MAX)

24      set acc='grep "Correctly Classified Instances" output$i | tail -1|awk '{printf("%.2f", $5)}'
        if ( 'echo $acc "-" $bestAcc " > 0" | bc' == 1) then
            set bestAcc='echo $acc'
            set bestCombination='echo $i'
        echo "Best combination updates at file "$i
29      endif

        @ i++
    end

34

    echo "Best accuracy          = $bestAcc" > $RESULTS
    echo "Best combination      = $bestCombination" >> $RESULTS

    exit
```

Listing C.3: *Script* principal para geração do código e dados na implementação de *feature subset selection* usando ILP.

```

2  #!/bin/csh

    if ($#argv != 2) then
        echo "syntax: mkFeatureSelection dataset depth"
    7  exit
    endif

    ### System Settings

12  set OUTPUT="output"
    set GETRESULTS_FILE="computeResult"
    set JOBSDIR="jobsdir/"
    set COMBINATIONS="detCombinations"
    set TMPDETERMS="____determs"
17  set GREPFILE="_exclude.grep"

    # Check the information provided

    set DSDIR="original/"
22
    if (! -f $DSDIR$1".b" ) then
        echo "file $DSDIR$1.b does not exist"
        exit
    else
27        set BCKFILE='echo $DSDIR$1".b"'
    endif

    if (! -f $DSDIR$1".f") then
        echo "file $DSDIR$1.f does not exist"
32        exit
    endif

    if (! -f $DSDIR$1".n") then
        echo "file $DSDIR$1.n does not exist"
37        exit
    endif

    set DATASET=$1
42  echo "    [Data set is '$DATASET']"
    set MAXDEPTH=$2          # depth limit for a breadth-first search on attributes
    echo "    [Depth limit is '$MAXDEPTH']"

    # make sure $JOBSDIR directory exists
47  mkdir -p $JOBSDIR
    \rm -r -f $JOBSDIR/* >& /dev/null

    #####
    # Preparing the cycle
52  #####

    # bf-selection
    # prepare feature selection list of attributes

```

```

57 set nDeterminations='grep -i '^:-' $BCKFILE|grep determination |wc -l'
   @ DepthLimit = $nDeterminations - 1

   if ($MAXDEPTH > $DepthLimit) then
       set MAXDEPTH=$MaxDepth
62   echo "Depth limit is $MAXDEPTH"
   endif

   echo "Found $nDeterminations determinations"

67 # associate each determination with a number
   grep -i '^:-' $BCKFILE|grep determination | awk '{print NR"| "$0}' > $JOBSDIR$TMPDETERMS

   # generate combinations
       ./gencombinations.yap $nDeterminations $MAXDEPTH $JOBSDIR"/"$COMBINATIONS
72
   #####
   # Feature Selection Cycle
   #####

77   pushd $JOBSDIR >& /dev/null

       ln -s ../$DSDIR${DATASET}* .
       cp ../aleph.pl .
       echo " [Feature selection step]"

82
       # All features data set
       cp -f $DATASET".b" param0.b
       cp -f $DATASET".f" param0.f
       cp -f $DATASET".uf" param0.uf
87       cp -f $DATASET".n" param0.n
       cp -f $DATASET".un" param0.un

       cat ../runILP|sed -e {s/DataSet/param0/} -e {s/RecFile/\`param0\.trace\`/} -e {s/POSTEST/\`
           param0\.uf\`/} -e {s/NEGSTEST/\`pa\
ram0\.un\`/} > run
92   chmod +x run
       tar czvf param0.tgz param0.? param0.u[nf] aleph.pl run >& /dev/null
       \rm -f param0.? param0.u[nf]

       set i=1
97   foreach comb ('cat $COMBINATIONS')
       echo -n "."
       set splitted='echo $comb|sed {s/\./ /}'
       echo -n " " > $GREPFILE
       foreach number ($splitted)
102         grep "^$number|" $TMPDETERMS |awk -F'|' '{print $2}'| sed -e {s/-//} -e {s/\ //g}>>
           $GREPFILE
       end

       grep -v -f $GREPFILE $DATASET".b" > param$i".b"
       cp -f $DATASET".f" param$i".f"
107       cp -f $DATASET".uf" param$i".uf"
       cp -f $DATASET".n" param$i".n"
       cp -f $DATASET".un" param$i".un"

       cat ../runILP|sed -e {s/DataSet/param${i}/} -e {s/RecFile/\`param${i}\.trace\`/} -e {s/

```

```

112 POSTEST/\ 'param${i}\.uf\'/} -e {s/\
NEGSTEST/\ 'param${i}\.un\'/} > run
    chmod +x run
    tar czvf param$i".tgz" param$i"."? param$i".u[nf] aleph.pl run >& /dev/null

    \rm -f param$i"."? param$i".u"?

117 @ i++
    end

    echo ""

122 #####
#                               Final Results
#####

127 @ MAX= $i - 1

#####
# Generate (by instantiation) the script that collects the results of
# each parameter combination and returns, for each measure, the best
132 # value and the corresponding parameter combination
#####

cat ../computeResult.model | sed {s/MAXVALUECONST/$MAX/} > $GETRESULTS_FILE
chmod +x $GETRESULTS_FILE

137 popd >& /dev/null
exit

```

Listing C.4: *Script* auxiliar para geração do código do nó que combina o resultado dos nós que correram em paralelo e dá o resultado final.

```

1  #!/bin/csh

### Global variables

6  set RESULTS="finalResult"
   set MAX=MAXVALUECONST
   set FILESTEM="param"

11

   set i=0
   while ($i <= $MAX)
       if (! -e $FILESTEM${i}.trace) then
           echo "[ERROR: can not find the settings file '$FILESTEM${i}.trace']"
16       exit
       endif
       @ i++
   end

21

#####
#                               Final Results
#####

26  echo ""
   echo "                               [Collecting Final Results]"
   echo ""

   set bestAcc=0
31  set bestFile=0
   set bestI=-1

   set i=0
   while ($i <= $MAX)
36
       set acc='grep "Accuracy = " $FILESTEM${i}.trace | tail -1 | awk '{ printf("%.2f", $3) }{'
       if ( 'echo $acc "-" $bestAcc " > 0" | bc' == 1) then
           set bestAcc='echo $acc '
           set bestFile='echo $FILESTEM${i}.trace '
41           set bestI=$i
       endif

       @ i++
   end

46

   echo "Best accuracy = $bestAcc " > $RESULTS
   echo "Best file      = $bestFile" >> $RESULTS
51   set differences='diff ../Task0/param0.b ../Task${bestI}/param${bestI}.b | grep
       determination | awk '{ print $3 }{'
   echo "Remove the following determinations to get the best result:" >> $RESULTS
   if ($#differences == 0) then
       echo "none ... All determinations should be used" >> $RESULTS

```

```
else
56   foreach determ ('echo $differences ')
       echo " " "$determ >> $RESULTS
       end
endif
61
exit
```

Listing C.5: *Script* principal para geração do código e dados na implementação de sintonização de parâmetros usando algoritmos do Weka.

```

3  #!/bin/csh

    if ($#argv != 4) then
        echo "syntax:      mkParameterTuning settings trainSet tuningSet classpos"
8      exit
    endif

    ### System Settings

13   set OUTPUT="output"
    set GETRESULTS_FILE="getResults"
    set JOBSDIR="jobsdir"

    ### Global variables

18   set SETTINGS_FILE=$1
    echo "    [Settings file is '$SETTINGS_FILE']"

    set TRAINING_SET=$2
23   echo "    [Training set file is '$TRAINING_SET']"

    set TUNING_SET=$3
    echo "    [Tuning set file is '$TUNING_SET']"

28   set CLASS_POS=$4
    echo "    [Class position is '$CLASS_POS']"

    # Check the information provided

33   if (! -e $SETTINGS_FILE) then
        echo "[ERROR: can not find the settings file '$SETTINGS_FILE']"
        exit
    endif

38   # make sure $JOBSDIR directory exists
    mkdir -p $JOBSDIR
    \rm -r -f $JOBSDIR/*
    #####
    #          G L O B A L          V A R s
43   #####

    set MEMOPTION = " -Xmx8192M "

    #####

48   #                      MAIN cycle
    #####

        pushd $JOBSDIR >& /dev/null

53   cp -r ../$TRAINING_SET ../$TUNING_SET ../$SETTINGS_FILE ../weka.jar .

        set Classifier = 'head -1 $SETTINGS_FILE'

```

```

echo " [Using ML algorithm: "$Classifier"]"

58 set paramCombinations='./mkParametersCombinations $SETTINGS_FILE'
set i=1
foreach parametersSet ($paramCombinations)

    set parset='echo $parametersSet|sed -e {s/\\,/\\ /g} -e {s/\\'//g} -e {s/\\%/\\ /g}'

63 echo " [generating combination "$parset"]"

    echo '#!/bin/csh' > run
    echo "" >> run
68 echo "java $MEMOOPTION -cp weka.jar $Classifier $parset -c $CLASS_POS -t $TRAINING_SET -T
    $TUNING_SET -i > $OUTPUT$i" >> r\
un
    chmod +x run
    tar czvf parm${i}.tgz run $TRAINING_SET $TUNING_SET weka.jar >& /dev/null
    \rm -f run >& /dev/null
73 @ i++
end # end of Parameter tuning cycle

@ MAX= $i - 1
#####
78 # Generate (by instantiation) the script that collects the results of
# each parameter combination and returns, for each measure, the best
# value and the corresponding parameter combination
#####

83 cat ../getResults.model | sed {s/MAXVALUECONST/$MAX/} > $GETRESULTS_FILE
chmod +x $GETRESULTS_FILE

\rm -f $TRAINING_SET $TUNING_SET $SETTINGS_FILE
popd >& /dev/null
88 exit

```

Listing C.6: *Script* auxiliar para geração do código do nó que combina o resultado dos nós que correram em paralelo e dá o resultado final.

```

1  #!/bin/csh

   ### Global variables

6  set FILESTEM="output"
   set RESULTS="finalResult"
   set MAX=MAXVALUECONST

11 set i=1
   while ($i <= $MAX)
       if (! -e $FILESTEM$i) then
           echo "[ERROR: can not find the settings file '$FILESTEM$i']"
           exit
16       endif
       @ i++
   end

21 #####
   #                               Final Results
   #####
   echo ""
   echo "                               [Collecting Final Results]"
26 echo ""

   set bestAcc=0
   set bestTP=0
   set bestFP=100
31 set bestPrec=0
   set bestRec=0
   set bestFmes=0
   set bestRoc=0

36 set i=1
   while ($i <= $MAX)

       set acc='grep "Correctly Classified Instances" output$i | tail -1|awk '{printf("%.2f", $5)}'
       if ( 'echo $acc "-" $bestAcc " > 0" | bc' == 1) then
41         set bestAcc='echo $acc'
         set bestAccSettings='grep Options output$i|awk '{for(i=2;i <= NF;i++) printf $i" ";
           printf("\n");}'
         endif

       set line='grep 'Weighted Avg' output$i |tail -1'

46       set tp='echo $line|awk '{print $3}''
       if ( 'echo $tp "-" $bestTP " > 0" | bc' == 1) then
           set bestTP='echo $tp|awk '{printf("%.2f", $1)}'
           set bestTPSettings='grep Options output$i|awk '{for(i=2;i <= NF;i++) printf $i" "; printf
51         ("\\n");}'
       endif
   end

```

```

56   set fp='echo $line|awk '{print $4}''
    if ( 'echo $bestFP "-" $fp " > 0" | bc' == 1) then
        set bestFP='echo $fp|awk '{printf("%.2f", $1)}''
        set bestFPSettings='grep Options output$i|awk '{for(i=2;i <= NF;i++) printf $i " "; printf
            ("\n");}''
    endif

61   set prec='echo $line|awk '{print $5}''
    if ( 'echo $prec "-" $bestPrec " > 0" | bc' == 1) then
        set bestPrec='echo $prec|awk '{printf("%.2f", $1)}''
        set bestPrecSettings='grep Options output$i|awk '{for(i=2;i <= NF;i++) printf $i " ";
            printf("\n");}''
    endif

66   set rec='echo $line|awk '{print $6}''
    if ( 'echo $rec "-" $bestRec " > 0" | bc' == 1) then
        set bestRec='echo $rec|awk '{printf("%.2f", $1)}''
        set bestRecSettings='grep Options output$i|awk '{for(i=2;i <= NF;i++) printf $i " ";
            printf("\n");}''
    endif

71   set fmes='echo $line|awk '{print $7}''
    if ( 'echo $fmes "-" $bestFmes " > 0" | bc' == 1) then
        set bestFmes='echo $fmes|awk '{printf("%.2f", $1)}''
        set bestFmesSettings='grep Options output$i|awk '{for(i=2;i <= NF;i++) printf $i " ";
            printf("\n");}''
    endif

76   set roc='echo $line|awk '{print $8}''
    if ( 'echo $roc "-" $bestRoc " > 0" | bc' == 1) then
        set bestRoc='echo $roc|awk '{printf("%.2f", $1)}''
        set bestRocSettings='grep Options output$i|awk '{for(i=2;i <= NF;i++) printf $i " ";
            printf("\n");}''
    endif

    @ i++
end

86

echo "Best accuracy          = $bestAcc [$bestAccSettings]" > $RESULTS
echo "Best true positives   = $bestTP [$bestTPsettings]" >> $RESULTS
echo "Best false positives  = $bestFP [$bestFPsettings]" >> $RESULTS
91 echo "Best precision      = $bestPrec [$bestPrecSettings]" >> $RESULTS
echo "Best recall          = $bestRec [$bestRecSettings]" >> $RESULTS
echo "Best F measure       = $bestFmes [$bestFmesSettings]" >> $RESULTS
echo "Best ROC              = $bestRoc [$bestRocSettings]" >> $RESULTS

```

Listing C.7: *Script* principal para geração do código e dados na implementação de validação cruzada usando algoritmos do Weka.

```
#!/bin/csh

5  if ($#argv != 4) then
    echo "syntax: mkCVweka dataset classifier nfolds classPosition"
    exit
  endif

10  set DATASET=$1
    echo "    [Data set is $DATASET]"

    set CLASSIFIER=$2
    echo "    [Using classiifer $CLASSIFIER]"

15  set CV_FOLDS=$3
    echo "    [Using ${CV_FOLDS}-fold cross validation]"

    set CLASS_POS=$4          #class position
20  echo "    [Class position is $CLASS_POS]"

    set MEMOPTION = " -Xmx8192M "

    if (! -e $DATASET) then
25    echo "[ERROR: can not find file $DATASET]"
    exit
  endif

#####
30  #      G L O B A L      V A R s
#####

# Weka filter to build CV blocks

35  set FILTER="weka.filters.unsupervised.instance.RemoveFolds"
    set SEED=1

# Directories and [temporary] files
set RESULTS_FILE="cvResultsFile"
40  set WORK_DIR="work/"
    set TMP="tmpResults"

# make sure work directory exists
mkdir -p $WORK_DIR >& /dev/null
45  # general clean up
    \rm -r -f $WORK_DIR* >& /dev/null

#####
50  # Preparing the cycle
#####

echo "[Generating CV folds ...]"

55
```

```
pushd $WORK_DIR >& /dev/null

    cp -f ../weka.jar ../$DATASET .
set i=1
60 while ($i <= $CV_FOLDS)
    echo "[Generating fold $i/$CV_FOLDS]"

    # get the training set
    java -cp weka.jar $FILTER -V -N $CV_FOLDS -F $i -S $SEED -i $DATASET -o cvTrain${i}.arff
65
    # get the test set
    java -cp weka.jar $FILTER -N $CV_FOLDS -F $i -S $SEED -i $DATASET -o cvTest${i}.arff

    # generate the script to run this pair of cv blocks
70 echo '#!/bin/csh' > run
    echo "java $MEMOOPTION -cp weka.jar $CLASSIFIER -c $CLASS_POS -t cvTrain${i}.arff -T cvTest${i}.
        arff -i > cv${i}.result" >> ru\
n
    chmod +x run
    tar czvf fold${i}.tgz run cvTest${i}.arff cvTrain${i}.arff weka.jar >& /dev/null
75 @ i++
end

echo "[done]"
80
    cat ../cvResults.model | sed {s/CVFOLDS/$CV_FOLDS/} > cvResults
    chmod +x cvResults

85 popd >& /dev/null
exit
```

Listing C.8: *Script* auxiliar para geração do código do nó que combina o resultado dos nós que correram em paralelo e dá o resultado final.

```

#!/bin/csh
4
### Global variables

set RESULTS="finalResult"
set MAX=CVFOLDS
9

set i=1
while ($i <= $MAX)
    if (! -e cv${i}.result) then
        echo "[ERROR: can not find the results file '$FILESTEM$i']"
14        exit
    endif
    @ i++
end

19

#####
#                               Final Results
#####

24 echo ""
echo "                               [Collecting Final Results]"
echo ""

#!/bin/csh
29

set TMP="___tmp___"
set MAX=CVFOLDS
set OUTFILE="cvFinalResult"

34

echo -n "(0" > $TMP
set i=1
while ($i <= $MAX)
    grep "Root mean squared error" cv${i}.result | tail -1 | awk '{ printf "+$5}' >> $TMP
39    @ i++
end
echo " )/"$MAX >> $TMP

set average='cat $TMP|bc -l '
44 echo -n "sqrt((0" > $TMP
set i=1
while ($i <= $MAX)
    set rmserr='grep "Root mean squared error" cv${i}.result | tail -1 | awk '{ print $5}''
    echo -n "+("$average-"$rmserr") * (" $average-"$rmserr")" >> $TMP
49    @ i++
end
set num='echo $MAX'
@ num--
echo " )/"$num)" >> $TMP
54

set stdev='cat $TMP|bc -l '

```

59

```
echo $average " " $stdev |awk '{ printf("RMSerr = %.3f(%.3f)\n", $1, $2);}' > $OUTFILE  
  
\rm -f $TMP  
  
exit
```

Listing C.9: *Script* principal para geração do código e dados na implementação de validação cruzada usando sistema de ILP Aleph.

```
#!/bin/csh
3
if ($#argv < 2) then
    echo "syntax: mkCV filestem numberOfFolds"
    exit
endif
8
set DSDIR="original/"

if (! -f $DSDIR$1".b" ) then
    echo "file $DSDIR$1.b does not exist"
13    exit
endif

if (! -f $DSDIR$1".f" ) then
    echo "file $DSDIR$1.f does not exist"
18    exit
endif

if (! -f $DSDIR$1".n" ) then
    echo "file $DSDIR$1.n does not exist"
23    exit
endif

# check if shuffle exists otherwise compile it
if (! -f shuffle) then
    if (! -f shuffle.c) then
        echo "file shuffle.c does not exist"
        exit
    endif
    gcc shuffle.c -lm -o shuffle >& /dev/null
33 endif

set CVDIR="cvdir"
set TMPDIR="tmp"
set FILESTEM=$1
38 set TMP="_lixo_sed_"
set EXTRAFILES='ls -lh $DSDIR | sed 's/*/g' | awk '{if(FNR > 1) print $NF}' | grep -v -f filepat '

mkdir -p $CVDIR
43 rm -r $CVDIR/* >& /dev/null
mkdir -p $TMPDIR
rm -r $TMPDIR/* >& /dev/null

pushd $TMPDIR >& /dev/null
48 ln -s ../shuffle .
set DSDIR=".." $DSDIR

#####
# Split the positives
53 #####

set TRAIN_SUFFIX=".f"
```

```

set TEST_SUFFIX=".uf"
set file=$DSDIR$1$TRAIN_SUFFIX
58 set nfolds=$2
set nexamples='wc -l $file|awk '{print $1}''

# check if number of examples is a multiple of the number of blocks
if ('expr $nexamples % $nfolds' == 0 ) then
63     set blocksize='expr $nexamples / $nfolds '
else
    set blocksize='expr $nexamples / $nfolds + 1 '
endif

68     echo "[Blocksize for positives is $blocksize]"
./shuffle $DSDIR$file | split -d -l $blocksize
#echo "./shuffle $file | split -d -l $blocksize"

set j="1"
73 while ($j <= $nfolds)
    set i="0"
    echo -n " " > "../$CVDIR"/cv"$j$TRAIN_SUFFIX"
    while ($i < $nfolds)
        if ($i == 'expr $j - 1') then # test block
78         if ($i < 10) then
            cp -f "x0"$i "../$CVDIR"/cv"$j$TEST_SUFFIX"
        else
            cp -f "x"$i "../$CVDIR"/cv"$j$TEST_SUFFIX"
        endif
83     else # training block
        if ($i < 10) then
            cat "x0"$i >> "../$CVDIR"/cv"$j$TRAIN_SUFFIX"
        else
            cat "x"$i >> "../$CVDIR"/cv"$j$TRAIN_SUFFIX"
88         endif
        endif
        @ i++
    end
    @ j++
93 end

rm -f x0*

#####
98 # Split the negatives
#####

set TRAIN_SUFFIX=".n"
set TEST_SUFFIX=".un"
103 set file=$DSDIR$1$TRAIN_SUFFIX

set nexamples='wc -l $DSDIR$file|awk '{print $1}''

# check if number of examples is a multiple of the number of blocks
108 if ('expr $nexamples % $nfolds' == 0 ) then
    set blocksize='expr $nexamples / $nfolds '
else
    set blocksize='expr $nexamples / $nfolds + 1 '
endif
113

```

```

        echo "[Blocksize for negatives is $blocksize]"
./shuffle $DSDIR$file | split -d -l $blocksize

set j="1"
118 while ($j <= $nfolds)
    set i="0"
    echo -n " " > "../$CVDIR"/cv"$j$TRAIN_SUFFIX"
    while ($i < $nfolds)
        if ($i == 'expr $j - 1') then # test block
123         if ($i < 10) then
            cp -f "x0"$i "../$CVDIR"/cv"$j$TEST_SUFFIX"
        else
            cp -f "x"$i "../$CVDIR"/cv"$j$TEST_SUFFIX"
        endif
128     else # training block
        if ($i < 10) then
            cat "x0"$i >> "../$CVDIR"/cv"$j$TRAIN_SUFFIX"
        else
            cat "x"$i >> "../$CVDIR"/cv"$j$TRAIN_SUFFIX"
133        endif
    endif
    @ i++
    end
    @ j++
138 end

rm -f x0*
popd >& /dev/null

143 pushd $CVDIR >& /dev/null
    \cp -f ../aleph.pl . >& /dev/null

#####
# copy the background files
148 #####

set SUFFIX=".b"
set file=$DSDIR$1".b"

153 set j="1"
while ($j <= $nfolds)
    \cp -f $DSDIR$file "cv"$j$SUFFIX
    @ j++
end

158 foreach extraFile ($EXTRAFILES)
    \cp -f $DSDIR$extraFile .
end

163 # foreach cv fold make a self contained tar file

set j="1"
while ($j <= $nfolds)
    cat ../runILP | sed -e {s/DataSet/cv$j/g} -e {s/RecFile/'cv$j\trace'/g} -e {s/POSTEST/'
        cv$j\uf'/g} -e {s/NEGSTEST/'\
168 cv$j\un'/g} > runILP
    chmod a+x runILP
    tar czvf cv$j.tgz cv$j.* $EXTRAFILES runILP aleph.pl

```

```
173 \rm -f cv$j.? cv$j.u?  
    @ j++  
end  
    # copy the script to collect the final result  
    cat ../cvResult | sed {s/CVFOLDS/$nfolds/} > cvResult  
    chmod +x cvResult  
178 popd >& /dev/null  
  
exit
```

Listing C.10: *Script* auxiliar para geração do código do nó que combina o resultado dos nós que correram em paralelo e dá o resultado final.

```
#!/bin/csh
4  set TMP="____tmp____"
   set MAX=CVFOLDS
   set OUTFILE="cvFinalResult"

9

   echo -n "(0" > $TMP
   set i=1
   while ($i <= $MAX)
       grep Accuracy cv${i}.trace | tail -1| awk '{printf "+$3}' >> $TMP
14   @ i++
   end
   echo ")/"$MAX >> $TMP

   set average='cat $TMP|bc -l'

19

   echo -n "sqrt((0" > $TMP
   set i=1
   while ($i <= $MAX)
       set acc='grep Accuracy cv${i}.trace | tail -1| awk '{print $3}'
24   echo -n "+("$average-"$acc") * (" $average-"$acc")" >> $TMP
       @ i++
   end
   set num='echo $MAX'
   @ num--
29   echo ")/"$num)" >> $TMP

   set stdev='cat $TMP|bc -l'

   echo $average " " $stdev |awk '{printf("accuracy = %.3f(%.3f)\n", $1, $2);}' > $OUTFILE
34 \rm -f $TMP
```


Índice

- Árvores de Decisão, 39
- árvores de decisão, 36
- Accuracy*, 22
- Alan Turing, 37
- análise de tendências, 19
- Aprendizagem Computacional, 15, 51, 52
- Apriori, 24
- Arecibo, 63
- associação, 18
- attribute-value language - AVL, 39
- AVL, 39
- background knowledge, 40
- Bases de Dados, 1
- BDR, 49
- Beowulf, 64
- Berkeley Open Infrastructure for Networ-
king Computing, 65
- Bigtable, 66
- cláusulas Horn, 40
- Classificação, 24
- classificação, 19
- Cloud Computing, 69
- cluster, 64
- Cluster Computing*, 55
- clustering, 19
- Commodity Grid (CoG) kit*, 92
- Computação na Nuvem, 69
- Condor, 64
- Condor*, 96
- Condor-API-Java, 92, 96
- Condor-G, 65
- Confiança, 22
- CRAN - Comprehensive R Archive Network,
51
- CRISP-DM, 32, 54
- Cross Industry Standard Process for Data
Mining, 31
- DaimlerChrysler, 32
- Data Cleaning, 29
- Data Integration, 29
- Data Mining, 15, 25
- Data Mining*, 54
- Data Mining Advisor, 23
- Data Mining Distribuído, 26, 27
- Data Reduction, 29

- Data Transformation, 29
- Data Warehouse, 29
- Data Warehouse (DW), 28
- DBMS, 29
- detecção de desvios, 19
- Distributed and Parallel Knowledge Discovery, 26
- distributed.net, 63
- Error rate, 22
- Estatística, 14
- Extração de Conhecimento (EC), 10
- home, 65
- Gestor de Tarefas (GT), 77
- Global Grid Forum (GGF), 64
- Globus Metacomputing Toolkit (GMT), 64
- Globus Toolkit, 65
- Globus Toolkit 4.0 (GT4)*, 92
- GNU - General Licence Public, 31
- GNU General Public License (GPL), 64
- Google, 10
- Grid Computing, 59
- Hadoop, 66
- HARVARD, 73, 78
- HARVARD-g, 94, 96
- Horn, 40
- HPC (High Performance Computing), 59
- ILP, 42
- Integração de dados, 29
- Java, 4, 74, 81
- Java*, 95, 96
- JDBC, 92
- jGlobus*, 94, 95
- k-means, 36
- Knowledge Discovery in Database (KDD), 10
- Knowledge Discovery in Database, 12
- Knowledge Discovery in Databases, 2
- lan, 62
- Limpeza dos Dados, 28
- Linux, 4, 74
- MAHOUT, 67
- man, 62
- MapReduce, 67, 68
- market basket, 24, 36
- Medidas de Interesse, 21
- meta-computação, 58
- meta-computador, 59
- metacomputing, 58
- MetaL, 23
- Multi-Relational Data Mining, 5
- Naive Bayes, 36
- NCR, 32

OLAP (On Line Analytical Processing), 2

Open Grid Services Architecture (OGSA), 64

outliers, 19

p2p, 68

peer-to-peer, 69

Projeto R, 50

PROLOG, 44

Prolog, 43

R, 50

R, 53

Rapidminer, 53

RATTLE, 51

RBF (Radial Basis Functions), 36

Reconhecimento de Padrões, 14

Redes Neurais, 39

Redução de dados, 29

Regras de Associação, 23

regressão linear, 19

Relational Data Mining, 128

Relational Data Mining (RDM), 46

S-plus, 50

SAS, 54

SAS Enterprise Miner, 54

segmentação, 19

home, 63, 65

SETI - Search for Extraterrestrial Intelligence, 63

SGBD, 34

Sistema de Gestão de Bases de Dados, 34

SPSS, 32, 53

SPSS Clementine, 54

Statistical Package for the Social Sciences, 53

sumarização, 18

super-computadores, 58

Suporte, 22

Tarefas de Data Mining, 18

tarefas de Data Mining, 17

Transformação de dados, 29

Waikato Environment for Knowledge Analysis, 51

wan, 62

Web, 17, 75

Web Mining, 17

WEKA, 51, 95

Windows, 4, 74

workflow, 76

x.509, 96

XML, 53, 78, 81

Yahoo, 10

YALE, 52

Yale, 95

Yet Another Learning Environment, 52