Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Projeto e Implementação do Módulo de *Clustering* Hierárquico do DISCOVER

Jean Metz
Maria Carolina Monard

Nº 278

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Agosto de 2006.

Projeto e Implementação do Módulo de *clustering* Hierárquico do DISCOVER *

Jean Metz
Maria Carolina Monard

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística
Laboratório de Inteligência Computacional
Caixa Postal 668, 13560-970 - São Carlos, SP, Brasil
e-mail: {metzz, mcmonard}@icmc.usp.br,

Resumo:

Diversas ferramentas têm sido propostas para a execução das tarefas envolvidas no processo de extração de conhecimento e aplicações de Mineração de Dados. Usualmente, essas ferramentas não atendem à todas as etapas do processo, mas sim a alguma tarefa específica, por exemplo o agrupamento de dados utilizado no aprendizado não-supervisionado. Neste trabalho é apresentado o projeto de um módulo de *clustering* hierárquico, integrado ao ambiente computacional DISCOVER, que dispõe de diversas ferramentas que podem ser utilizadas nas etapas do processo de Mineração de Dados e Textos. Esse módulo atua na etapa de aquisição de conhecimento utilizando algoritmos de aprendizado de máquina e facilita a execução do *clustering* em um ambiente único e padronizado, com suporte às técnicas de pré-processamento de dados e pós-processamento de conhecimento além das facilidades de agregação de novos algoritmos de *clustering* ao módulo proposto.

Palavras-Chave: Aprendizado de Máquina, Extração de Padrões, Algoritmos de Clustering

---

Agosto 2006

*Trabalho desenvolvido com o apoio do CNPq.

Este documento foi preparado com o formatador de textos LaTeX. O sistema de citações de referências bibliográficas utiliza o padrão *Apalike* do sistema bibTeX.

Sumário

Sumário i

Lista de Figuras iii

Lista de Tabelas v

1 Introdução 1

2 Definições e Notação 3

3 *Clustering* Hierárquico 4

4 Uma Visão Geral da Biblioteca Cluster 3.0 6
4.1 Descrição das Medidas de Similaridade 7
4.2 Algoritmos de *Clustering* Implementados 11
4.2.1 Algoritmos de Particionamento 11
4.2.2 Algoritmos Hierárquicos 14
4.3 Arquivos Utilizados 21
4.3.1 Arquivos de Entrada 21
4.3.2 Arquivos de Saída 22
4.4 Interface com Linguagens de *Script* 23

5 Projeto e Implementação do Módulo de *Clustering* Hierárquico do DISCOVER 24
5.1 A Biblioteca de Classes DOL 25
5.2 O Gerenciador de Experimentos SNIFFER 26
5.3 DSX - DISCOVER STANDARD SINTAX 27
5.4 Descrição do Módulo de *Clustering* Hierárquico 29
5.4.1 O Módulo CLUSTER.PM 31
5.4.2 O Ambiente de Gerenciamento de Experimentos e Análise do *Clustering* 31

6 Considerações finais 38

Referências Bibliográficas 41

Lista de Figuras

1 Dendograma 5
2 Distância de Manhattan. 8
3 Modelo de misturas finitas Gaussianas. 13
4 *Single Link* 15
5 *Complete Link* 16
6 Matriz de distâncias. 18
7 *Average Link* 19
8 Módulo CLUSTER.PM 31
9 Ambiente de execução de experimentos SHELL*Cluster*. 32
10 Módulo de *clustering* hierárquico do DISCOVER. 38

Lista de Tabelas

1 Tabela atributo-valor. 4
2 Matriz de distância. 4
3 Notação utilizada. 4
4 Relação entre atributos para o cálculo do coeficiente de Kendall's τ. 11
5 Exemplo 1: dados para o *clustering* com a biblioteca Cluster 3.0. 22
6 Exemplo 2: dados para o *clustering* com a biblioteca Cluster 3.0. 22
7 Arquivo `toy.cdt`. 23
8 Arquivo `toy.gtr`. 24
9 Conjunto de dados `toy`. 28
10 Arquivo `toy.names`. 28
11 Arquivo `toy.dat`. 30
12 Arquivo `toy.mask`. 30
13 Identificadores das medidas de similaridade. 33
14 Identificadores dos algoritmos de *clustering* hierárquico. 33
15 Valores padrão para os parâmetros de execução do *clustering*. . . 33
16 Arquivo `.ds`. 36
17 Arquivo `.p`. 36
18 Arquivo `.job`. 37

1 Introdução

O *clustering* é uma técnica de aprendizado de máquina não-supervisionado, usualmente implementado em ferramentas para análise e exploração de dados utilizadas em aplicações de Mineração de Dados e Textos, nas quais o objetivo é a extração de padrões e/ou informações úteis que possam estar embutidas nos dados. O conhecimento obtido por meio da análise dessas informações pode ser de grande valia para tomada de decisão em qualquer aplicação.

São muitas as ferramentas computacionais desenvolvidas para a tarefa de extração de padrões e descoberta de conhecimento. O Mineset[1] (Rathjens, 2000), por exemplo, é uma ferramenta comercial freqüentemente utilizada em aplicações de Mineração de Dados (MD). Essa ferramenta, desenvolvida pela *Silicon Gaphics*, é composta por um conjunto de recursos utilizados na realização de tarefas de MD e análise multi-dimensional dos dados, tais como alguns dos principais algoritmos de Aprendizado de Máquina (AM). Conta também com poderosas ferramentas de visualização, as quais permitem, além da visualização dos dados, a visualização de estatísticas geradas sobre esses dados e dos resultados obtidos a partir das ferramentas de MD. Entretanto, O Mineset é uma ferramenta comercial de alto custo, o que inviabiliza a sua aquisição para uso em universidades e/ou muitos centros de pesquisas. Além disso, faz uso de algoritmos proprietários o que dificulta o acesso às heurísticas, estruturas e representações internas desses algoritmos, e, conseqüentemente, não contribuem para o desenvolvimento científico e para a comparação com outros algoritmos.

Algumas alternativas para superar as limitações de custo podem ser encontradas nas ferramentas de domínio público, tais como a biblioteca $\mathcal{MLC}^{++}$ — *Machine Learning in* $\mathcal{C}^{++}$ — (Kohavi et al., 1996), o WEKA — *Waikato Environment for Knowledge Analysis* — (Witten and Frank, 2000), YALE — *Yet Another Learning Environment*[2] e *Orange*[3] (Demsar and Zupan, 2004). Entretanto, essas ferramentas apresentam algumas características indesejáveis ou não adequadas para o desenvolvimento de muitos trabalhos. A biblioteca $\mathcal{MLC}^{++}$ trata os algoritmos de aprendizado como objetos fechados, não fornecendo uma visão única dos modelos que podem ser extraídos. Além disso, o processo de incorporação de novas funcionalidades à biblioteca não é trivial, pois os módulos adicionados devem ser adaptados aos padrões adotados pela biblioteca e todos os módulos, inclusive os que já estavam configurados antes da adição do novo módulo, devem ser recompilados. A falta de atualizações também é um problema com a biblioteca $\mathcal{MLC}^{++}$, que está aos cuidados da *SGI — Silicon Graphics* desde 1995, e sua última versão data de 1997.

O WEKA, por outro lado, é uma sistema muito bom, mas reimplementa os algoritmos de aprendizado utilizando a linguagem de programação JAVA[4], o que padroniza as interfaces dos algoritmos e produz código uniforme, facilitando a inclusão de novas funcionalidades. Entretanto, a recodificação dos algoritmos está sempre sujeita à tomadas de decisão que podem acarretar a obtenção de resultados diferentes dos encontrados com o código original proposto pelos autores do algoritmo.

[1] http://www.sgi.com
[2] http://www.yale.cs.uni-dortmund.de
[3] http://www.ailab.si/orange
[4] http://java.sun.com/reference/index.html

O ambiente YALE também é implementado em JAVA, mas ao contrário do WEKA, utiliza alguns algoritmos em suas implementações originais, o que possibilita a reprodução dos mesmos resultados apresentados pelos idealizadores desses algoritmos se aplicados sobre os mesmos conjuntos de dados.

O Orange é uma ferramenta para MD baseada em componentes escritos em linguagem C++. Esses componentes podem ser acessados a partir de *scripts* Phyton ou por meio de uma interface de objetos chamados de *Orange Widgets*. Assim como o YALE, essa ferramenta disponibiliza a implementação original de alguns algoritmos amplamente utilizados na literatura, mas também possui implementações próprias de diversos indutores e outros algoritmos para análise de dados e extração de conhecimento.

Além desses ambientes de *software*, que integram uma grande quantidade de ferramentas para as mais diversas técnicas utilizadas em aplicações de AM, há também um pacote de *software*, Cluto (Karypis, 2003), implementado exclusivamente para a execução do *clustering* hierárquico e particional. Outros algoritmos podem ser encontrados também em ambientes estatísticos, como o R (Team, 2005) (`http://www.r-project.org/`), que dispõe de diversos pacotes com implementações de algoritmos de *clustering*.

Apesar de toda essa gama de ferramentas, poucas suportam todas as etapas envolvidas na Mineração de Dados. Considerando essa característica, e aliado ao fato de que muitas pesquisas em AM desenvolvidas por diferentes pesquisadores no LABIC — Laboratório de Inteligência Computacional do ICMC-USP, têm em geral alguns componentes em comum, existindo em algumas situações duplicação de esforços no desenvolvimento de *softwares*, surgiu a idéia do projeto DISCOVER, proposto inicialmente em (Baranauskas and Batista, 2000), o qual visa fornecer um ambiente integrado para apoiar todas as etapas do processo de descoberta de conhecimento, oferecendo funcionalidades voltadas para Aprendizado de Máquina, Mineração de Dados e Mineração de Textos. Motivados pela necessidade de ferramentas de AM não-supervisionado que pudessem ser executadas dentro de um ambiente único com diversas facilidades e ferramentas para pré e pós-processamento de dados, decidiu-se desenvolver um módulo de *clustering* hierárquico e integrá-lo ao ambiente DISCOVER.

Nesse contexto, o objetivo deste trabalho consiste em descrever o projeto e implementação desse módulo de *clustering* hierárquico. No desenvolvimento desse módulo foi utilizada uma biblioteca escrita na linguagem ANSI C, com implementações de diversas rotinas para execução do *clustering*. Além disso, a biblioteca de classes do DISCOVER foi utilizada para fazer a integração do módulo proposto com os demais módulos disponíveis no nesse ambiente. Esse fato também facilitou a execução de experimentos de *clustering*, no sentido de que outras técnicas podem ser aplicadas aos conjuntos de dados nas etapas de pré e pós-processamento em um ambiente único e integrado. Para que isso fosse possível, foi necessária a elaboração de uma sintaxe padrão para a descrição dos clusters de modo que as saídas de todos os algoritmos fossem representadas de maneira coerente e padronizada. O fator determinante nesse ponto foi, justamente, encontrar essa sintaxe padrão, uma vez que cada algoritmo de *clustering* possui o seu padrão de representação particular. Em um trabalho desenvolvido anteriormente, (Metz and Monard, 2006), foi realizado um levantamento bibliográfico dos algoritmos de *clustering* hierárquico mais freqüentemente utilizados, assim como as representações e estruturas de da-

dos internas. Com base nesse estudo, foram decididas as características do módulo de *clustering* hierárquico do DISCOVER, cuja implementação é descrita neste trabalho.

Este trabalho está organizado da seguinte maneira: Na Seção 2, são apresentadas algumas definições e notações matemáticas utilizadas para a descrição dos conceitos envolvidos nas tarefas de aprendizado. Na Seção 3, são apresentadas brevemente as etapas para a realização do *clustering*, mais especificamente o *clustering* hierárquico. Uma descrição mais detalhada de todo o processo envolvido na realização do *clustering*, abordagens e algoritmos pode ser encontrada em (Metz and Monard, 2006). Em seguida, na Seção 4 é descrita a biblioteca utilizada para a implementação do módulo de *clustering* proposto. Finalmente na Seção 5 são apresentados em detalhes o projeto e a implementação desse módulo de *clustering* hierárquico. Na Seção 6 são apresentadas as considerações finais deste trabalho.

2 Definições e Notação

Em tarefas de aprendizado de máquina, comumente o conjunto de dados utilizado para análise é representado por meio de uma estrutura de matriz, freqüentemente chamada de tabela atributo-valor. Essa tabela é utilizada em tarefas de aprendizado supervisionado e também em tarefas de aprendizado não-supervisionado.

A entrada para um algoritmo de aprendizado supervisionado consiste usualmente de um conjunto E de N exemplos (ou casos) de treinamento $E = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ rotulados com os valores y de uma função f desconhecida $y = f(\mathbf{x})$, onde os valores $\mathbf{x}_i$ são vetores da forma $(x_{i1}, x_{i2}, \dots, x_{iM})$ cujos componentes são valores discretos ou contínuos relacionados ao conjunto de atributos $X = \{X_1, X_2, \dots, X_M\}$. Ou seja, x_{il} denota o valor do atributo X_l do exemplo i. Dado esse conjunto de exemplos de treinamento, o algoritmo induz uma hipótese h que deve aproximar a verdadeira função f, tal que dado um novo exemplo $\mathbf{x}$, $h(\mathbf{x})$ prediz o valor y correspondente. No caso dos valores y pertencerem a um conjunto discreto de N_{C_l} classes, *i.e.*, $y \in \left\{C_1, \dots, C_{N_{C_l}}\right\}$, a tarefa de aprendizado é chamada de classificação. Por outro lado, se $y \in \mathbb{R}$, essa tarefa é denominada de regressão.

O formato atributo-valor, comumente usado para representar exemplos, é apresentado na Tabela 1. Como mencionado, em aprendizado supervisionado cada exemplo é associado a uma classe (rótulo), que pode ser discreta (classificação) ou contínua (regressão). No caso de aprendizado não-supervisionado, não há informação sobre a classe associada a cada exemplo. Nesse contexto, cada exemplo é então representado por um vetor no espaço M-dimensional.

Em tarefas de aprendizado não-supervisionado, especificamente o *clustering*, pode-se também utilizar como entrada para os algoritmos uma matriz de distâncias — Tabela 2 — com N linhas e N colunas ao invés da tabela atributo-valor. Nesse caso, as células da matriz utilizada no processo de *clustering* contêm o valor de distância entre cada par de exemplos, de acordo com alguma medida ou métrica de distância. Uma vez que as medidas de distâncias utilizadas são simétricas, a matriz construída a partir desses valores também é simétrica. Além disso, a diagonal principal contém todos os elementos com valor zero, pois a distância entre um elemento (exemplo) e ele mesmo

	X_1	X_2	$\ldots$	X_M	Classe (Y)
E_1	x_{11}	x_{12}	$\vdots$	x_{1M}	y_1
E_2	x_{21}	x_{22}	$\vdots$	x_{2M}	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
E_N	x_{N1}	x_{N2}	$\vdots$	x_{NM}	y_N

Tabela 1: Representação do conjunto de dados por meio da tabela atributo-valor.

é zero.

Exemplos	E_1	E_2	E_3	$\ldots$	E_N
E_1	0	$dist(E_1, E_2)$	$dist(E_1, E_3)$	$\ldots$	$dist(E_1, E_N)$
E_2	$dist(E_2, E_1)$	0	$dist(E_2, E_3)$	$\ldots$	$dist(E_2, E_N)$
E_3	$dist(E_3, E_1)$	$dist(E_3, E_2)$	0	$\ldots$	$dist(E_3, E_N)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	0	$\vdots$
E_N	$dist(E_N, E_1)$	$dist(E_N, E_2)$	$dist(E_N, E_3)$	$\ldots$	0

Tabela 2: Matriz de distância utilizada pelos algoritmos de *clustering*.

Um resumo da notação utilizada na tabela atributo-valor e na matriz de distâncias, assim como a notação utilizada nas equações descritas neste trabalho, é apresentado na Tabela 3.

N:	número de exemplos do conjunto de dados
M:	número de atributos dos exemplos
N_{C_l}:	número de classes do conjunto de dados
$E_i,\ i = 1, 2, ...N$:	exemplo i do conjunto de dados E
$X_l,\ l = 1, 2, ...M$:	atributo l do conjunto de atributos X
x_{il}:	valor do atributo l do exemplo i
w_{E_i}:	peso do exemplo E_i
w_{X_l}:	peso do atributo X_l
$sim(E_i, E_j)$:	similaridade entre os exemplos E_i e E_j
$dist(E_i, E_j)$:	distância entre os exemplos E_i e E_j
$cor(E_i, E_j)$:	correlação entre os exemplos E_i e E_j
$\bar{x_i}$:	média dos valores dos atributos do exemplo E_i, *i.e.*, $\bar{x_i} = \frac{\sum_{l=1}^{M} x_{il}}{M}$
σ_i:	desvio padrão dos valores dos atributos do exemplo E_i
nc_τ:	número de pares concordantes para o cálculo do τ
nd_τ:	número de pares discordantes para o cálculo do τ
k:	número de clusters
$P(C_i)$:	probabilidade do cluster i

Tabela 3: Notação utilizada.

3 *Clustering* Hierárquico

Assim como nas outras abordagens de *clustering*, o *clustering* hierárquico constrói os agrupamentos de modo que exemplos presentes no mesmo cluster possuam alta similaridade e exemplos pertencentes à clusters diferentes possuam baixa similaridade. Entretanto, uma distinção entre essa abordagem e as demais é que o resultado obtido não é constituído apenas de uma

partição do conjunto de dados inicial, mas sim de uma hierarquia que descreve um particionamento diferente em cada nível analisado. Essa hierarquia de agrupamento é freqüentemente representada por uma estrutura de árvore conhecida como Dendograma, ilustrada na Figura 1.

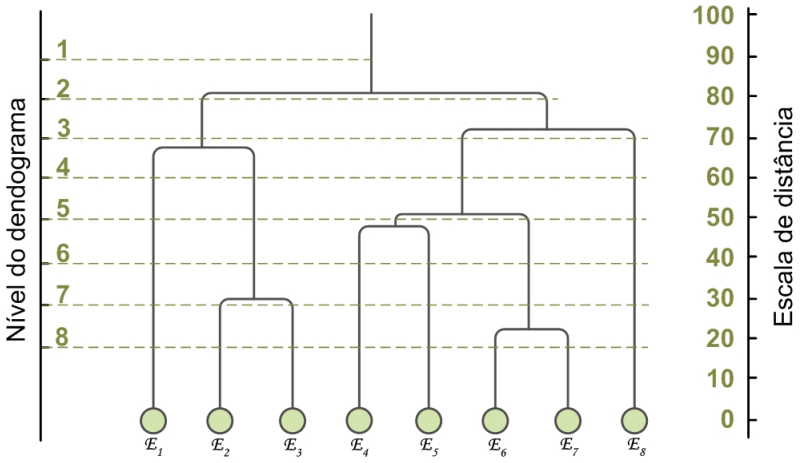

Figura 1: *Clustering* hierárquico representado por meio de um Dendograma (Metz and Monard, 2006).

Duas estratégias podem ser utilizadas com o *clustering* hierárquico:

1. Aglomerativa; e
2. Divisiva.

A primeira segue a direção *botton-up* e a segunda a direção inversa, *top-down*.

Qualquer que seja a abordagem utilizada, algumas etapas devem ser executadas para a correta aplicação do processo de *clustering*, como: pré-processamento, seleção da medida para mensurar o grau de similaridade entre exemplos/-clusters, execução do algoritmo de *clustering*, avaliação dos resultados e, finalmente, a interpretação dos clusters encontrados. Essas etapas são brevemente descritas a seguir.

Pré-processamento: todas as tarefas de transformação dos dados devem ser realizadas nessa etapa, como a normalização dos atributos, a redução de dimensionalidade por meio da seleção dos atributos mais relevantes, a criação de novos atributos ou qualquer alteração que possa contribuir para a geração de um modelo melhor (Liu and Motoda, 1998).

Seleção da medida para o cálculo da similaridade: nessa etapa devem ser determinadas as medidas de similaridade utilizadas pelo algoritmo de *clustering*, as quais devem ser selecionadas considerando as características do conjunto de dados, entre elas o tipo e a escala dos atributos. Uma revisão de vários tipos de medidas de similaridade, como métricas de distâncias e medidas de correlação, pode ser encontrada em (Jain and Dubes, 1988) e (Everitt, 1993).

Execução do algoritmo de agrupamento: nesse estágio o conjunto de exemplos deve ser submetido a algum algoritmo para a construção dos clusters.

Avaliação dos resultados: essa etapa é realizada para quantificar a qualidade dos resultados obtidos, o que é usualmente feito por meio de índices estatísticos, que avaliam a adequabilidade dos clusters ao conjunto de dados. Uma descrição dos índices mais freqüentemente utilizados pode ser encontrada em (Halkidi et al., 2002a) e (Halkidi et al., 2002b).

Interpretação: é nessa etapa que os clusters identificados são examinados para atribuição de algum significado conceitual que esteja relacionado ao domínio da aplicação. Nesse contexto, o especialista tem papel fundamental, pois avalia os clusters para investigar se existe algum significado prático, e/ou diferenças entre clusters distintos, considerando os padrões representados em cada cluster.

Na próxima seção são apresentadas algumas características da biblioteca Cluster 3.0, utilizada para implementação do módulo de *clustering* hierárquico do DISCOVER.

4 Uma Visão Geral da Biblioteca Cluster 3.0

A biblioteca Cluster 3.0, desenvolvida no Laboratório de Análise de informações de DNA, no Centro de Genoma Humano do Instituto de Ciências Médicas da Universidade de Tokyo (Hoon et al., 2005), é uma coleção de rotinas que implementam os algoritmos de *clustering* freqüentemente referenciados na literatura, como os hierárquicos *Centroid Link*, *Single Link*, *Complete Link* e *Average Link* e os particionais *k-means*, *k-medians* e *k-medoids*. Além disso, a biblioteca contém rotinas para a realização do *clustering* por meio de Redes Auto-Organizáveis (*Self-Organizing Maps — SOM*) e para identificação de características relevantes por meio da Análise de Componentes Principais (*Principal Component Analysis — PCA*).

Como dito anteriormente, os clusters são formados a partir da relação de similaridade (ou distância) entre os exemplos contidos no conjunto de dados. É necessário, então, entender o que é exatamente a similaridade ou como ela pode ser obtida a partir dos exemplos disponíveis para análise. Na biblioteca Cluster 3.0, foram implementadas seis medidas de correlação e duas métricas de distância entre pontos no espaço Euclidiano. É importante observar que as medidas de distância fornecem o complemento da similaridade, portanto, quanto maior a distância entre dois exemplos, menor é sua similaridade e quanto menor a distância, maior a similaridade.

A biblioteca dispõem ainda de estratégias para atribuição de pesos aos atributos e/ou exemplos, assim como rotinas para tratamento de valores faltantes ou desconhecidos. Desse modo, se o peso do atributo X_i é w_i, sendo w_i diferente de um, significa que ele terá maior ou menor influência no *clustering*, dependendo se o valor de w_i é maior ou menor que um, respectivamente. O mesmo ocorre com os exemplos que possuem peso diferente de um.

Em diversos experimentos de *clustering*, freqüentemente alguns dos valores dos atributos que descrevem os exemplos não são informados, fazendo com que haja necessidade de estratégias para o tratamento desses valores desconhecidos durante o cálculo da similaridade. Nesses casos, pode-se utilizar uma máscara que indica quais valores não são conhecidos. Assim, se o valor para um atributo não foi informado, ele não é considerado no cálculo da similaridade.

A seguir são descritos alguns detalhes da implementação da biblioteca Cluster 3.0, entre eles as medidas de distância disponíveis, os algoritmos de *clustering* e os arquivos utilizados pelos procedimentos implementados nessa biblioteca, a qual foi utilizada para implementar o módulo de *clustering* hierárquico do DISCOVER, descrito neste trabalho.

4.1 Descrição das Medidas de Similaridade

Diversos índices de proximidade foram propostos para o cálculo da similaridade entre exemplos, os quais devem satisfazer as seguintes condições:

i. $dist(E_i, E_j) \geq 0, \forall(i, j)$ (positividade)

ii. $dist(E_i, E_j) = 0$ se e somente se $E_i = E_j$ (identidade)

iii. $dist(E_i, E_j) = dist(E_j, E_i)$ (simetria)

Para que uma medida de distância seja considerada uma métrica ela deve satisfazer, além das três propriedades anteriores, a propriedade da desigualdade triangular:

iv. $dist(E_i, E_j) \leq dist(E_i, E_q) + dist(E_q, E_j), \forall(i, j, q)$

A similaridade pode ser calculada a partir de medidas de distância ou medidas de correlação. Para medidas de distância, a similaridade entre dois exemplos é definida pela Equação 1:

$$sim(E_i, E_j) = 1 - dist(E_i, E_j) \tag{1}$$

onde $dist(E_i, E_j)$ é a distância entre os exemplos E_i e E_j calculada a partir de uma métrica de distância pré-definida.

As métricas de distância implementadas na biblioteca Cluster 3.0 são:

***Euclidiana*:** essa métrica de distância, definida pela Equação 2, é provavelmente a mais utilizada pelos algoritmos de *clustering*. Ela expressa a distância geométrica Euclidiana entre os exemplos em um espaço multidimensional.

$$dist(E_i, E_j) = \sqrt{\sum_{l=1}^{M} (x_{il} - x_{jl})^2} \tag{2}$$

***Manhattan/city-block*:** essa métrica, também conhecida como distância-L_1, pode ser definida como a distância entre dois pontos no espaço Euclidiano, em um sistema de coordenadas cartesianas fixo, como sendo a soma dos comprimentos das projeções dos segmentos de reta entre os pontos dos eixos das coordenadas, *i.e.*, é a soma das diferenças absolutas dos valores dos atributos. Por exemplo, no espaço $\mathbb{R}^2$ a distância de Manhattan entre o ponto p_1 com coordenadas (x_1, y_1) e o ponto p_2 com coordenadas (x_2, y_2) é $|x_1 - x_2| + |y_1 - y_2|$. De maneira mais formal, essa métrica é definida pela Equação 3.

$$dist(E_i, E_j) = \sum_{l=1}^{M} |x_{il} - x_{jl}| \tag{3}$$

A distância de Manhattan é assim denominada porque em várias cidades é praticamente impossível estabelecer uma rota entre dois pontos através de uma reta, devido ao fato das cidades serem subdivididas em quadras com prédios, conforme ilustrado na Figura 2.

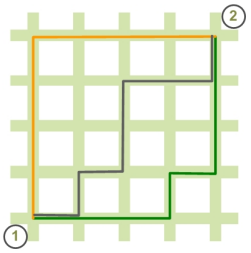

Figura 2: Distância de Manhattan: as três rotas apresentam o mesmo comprimento.

A similaridade baseada em medidas de correlação, diferentemente das métricas de distância, são semi-métrica pois não satisfazem a propriedade de desigualdade triangular. As medidas de correlação implementadas na biblioteca Cluster 3.0 são apresentadas a seguir.

Correlação de Pearson: esse coeficiente, também chamado de *coeficiente de correlação produto-momento*, ou simplesmente *r de Pearson*, mede o grau da correlação e a direção dessa correlação entre duas variáveis. Esse coeficiente, assume apenas valores entre -1 e 1. Quando $r = 1$, significa que há uma correlação perfeita entre as duas variáveis analisadas. Se $r = -1$, significa que há uma correlação inversa, *i.e.*, se uma das variáveis aumenta a outra diminui. Finalmente, se $r = 0$, significa que não há dependência linear entre as variáveis. Esse coeficiente é calculado por meio da Equação 4.

$$r(E_i, E_j) = \frac{1}{M} \sum_{l=1}^{M} (\frac{x_{il} - \bar{x}_i}{\sigma_i})(\frac{x_{jl} - \bar{x}_j}{\sigma_j}) \tag{4}$$

onde $\bar{x}_i$ e $\bar{x}_j$ representam as médias dos valores dos atributos dos exemplos E_i e E_j, respectivamente e, σ_i e σ_j seus desvios padrão.

A similaridade entre exemplos, calculada por meio de índices de correlação, tem o mesmo valor resultante do cálculo da correlação, *i.e.*, $sim(E_i, E_j) = cor(E_i, E_j)$. Por outro lado, a distância entre os exemplos E_i e E_j calculada por meio de um coeficiente de correlação, é dada pelo seu complemento: $dist(E_i, E_j) = 1 - cor(E_i, E_j)$. Quando o índice de correlação utilizado é o de Pearson, a similaridade entre dois exemplos é $sim(E_i, E_j) = r(E_i, E_j)$ e distância $dist(E_i, E_j) = 1 - r(E_i, E_j)$.

É importante observar que o coeficiente de Pearson centraliza automaticamente os dados subtraindo sua média, e os normaliza dividindo por seus respectivos desvios padrão. Em algumas situações essa normalização pode ser útil, por outro lado, acarreta na perda de informação.

Correlação de Pearson absoluto: o valor absoluto do coeficiente de correlação de Pearson resulta em um valor entre zero e um, que indica se há ou não correlação entre os dados, mas não informa se a correlação é positiva ou negativa. A similaridade entre dois exemplos é obtida a partir da Equação 5:

$$sim(E_i, E_j) = |r(E_i, E_j)| \tag{5}$$

onde $r(E_i, E_j)$ é o coeficiente de correlação de Pearson definido pela Equação 4.

Correlação de Pearson não-centralizado: em alguns casos é preferível utilizar o coeficiente de correlação não-centralizado, que é definido pela Equação 6. Esse coeficiente é equivalente ao cosseno do ângulo entre dois vetores.

$$r_U(E_i, E_j) = \frac{1}{M} \sum_{l=1}^{M} (\frac{x_{il}}{\sigma_i^{(0)}})(\frac{x_{jl}}{\sigma_j^{(0)}}) \tag{6}$$

onde

$$\sigma_i^{(0)} = \sqrt{\frac{1}{M} \sum_{l=1}^{M} x_{il}^2} \tag{7}$$

e

$$\sigma_j^{(0)} = \sqrt{\frac{1}{M} \sum_{l=1}^{M} x_{jl}^2} \tag{8}$$

A similaridade é calculada a partir da Equação 9:

$$sim(E_i, E_j) = r_U(E_i, E_j) \tag{9}$$

Correlação de Pearson não-centralizado e absoluto: de maneira análoga à correlação absoluta de Pearson, pode-se também utilizar a correlação absoluta não-centralizada. Nesse caso, a similaridade é calculada por meio da Equação 10:

$$sim(E_i, E_j) = |r_U(E_i, E_j)| \tag{10}$$

Geometricamente, o valor absoluto do coeficiente de correlação não-centralizado é igual ao cosseno do menor ângulo entre os dois exemplos analisados.

O coeficiente de correlação de Spearman e o coeficiente de Kendall's τ, descritos a seguir, são duas medidas de similaridade não paramétrica baseadas no coeficiente de correlação de Pearson, porém mais robustas em relação à presença de *outliers*.

Correlação de Spearman: o processo para o cálculo desse coeficiente de correlação inicia por meio da substituição de cada valor dos atributos dos exemplos por seus índices relativos à sua magnitude, *i.e.*, se o valor do atributo x_{i1} é menor que o valor do atributo x_{i2}, então seus índices são, respectivamente, 1 e 2. Em seguida, a correlação de Pearson é calculada sobre os vetores a partir desses índices ao invés de utilizar os valores dos atributos originais. A notação utilizada para esse coeficiente é $r_S(E_i, E_j)$.

Como ilustração, considere os exemplos:

$$E_1 = \{2.3, 6.7, 4.5, 20.8\} \text{ e } E_2 = \{2.1, 5.9, 4.4, 4.2\}.$$

O primeiro passo para o cálculo é substituir seus valores pelos seus respectivos índices:

$$E_1 = \{1, 3, 2, 4\} \text{ e } E_2 = \{1, 4, 3, 2\}$$

A partir desses índices é, então, calculado o coeficiente de correlação de Pearson, por meio da Equação 4. Para isso, os valores de média e desvio padrão dos os exemplos envolvidos devem ser calculados:

$$\overline{x}_1 = 2.5,\ \sigma_1 = 1.29$$
$$\text{e}$$
$$\overline{x}_2 = 2.5,\ \sigma_2 = 1.29$$

$$r_S(E_1, E_2) = \frac{1}{4}\sum_{l=1}^{4}(\frac{x_{1l} - 2.5}{1.29})(\frac{x_{2l} - 2.5}{1.29}) = \frac{1}{4} \times 1.2 = 0.3 \tag{11}$$

Desse modo, a correlação de Spearman para os exemplos E_1 e E_2 é: $r_S(E_1, E_2) = 0.3$.

A similaridade é obtida por meio da Equação 12:

$$sim(E_i, E_j) = r_S(E_i, E_j) \tag{12}$$

onde $r_S(E_i, E_j)$ é o coeficiente de correlação de Spearman.

Kendall's τ: esse coeficiente é muito semelhante ao coeficiente de correlação de Spearman, mas ao invés de utilizar os índices dos atributos em relação a todos os atributos, apenas os índices relativos entre pares de exemplos são utilizados para o cálculo da correlação. Para isso, considere todos os pares de atributos dos exemplos envolvidos no cálculo. Dois pares de atributos são concordantes se

$$x_{il} < x_{i(l+1)} \text{ e } x_{jl} < x_{j(l+1)}$$
$$\text{ou}$$
$$x_{il} > x_{i(l+1)} \text{ e } x_{jl} > x_{j(l+1)}$$

e são discordantes se

$$x_{il} < x_{i(l+1)} \text{ e } x_{jl} > x_{j(l+1)}$$

ou

$$x_{il} > x_{i(l+1)} \text{ e } x_{jl} < x_{j(l+1)}$$

Utilizando os dois exemplos anteriores, *i.e.*, $E_1 = \{2.3, 6.7, 4.5, 20.8\}$ e $E_2 = \{2.1, 5.9, 4.4, 4.2\}$, essas relações encontram-se representadas na Tabela 4.

-	(2.3, 2.1)	(6.7, 5.9)	(4.5, 4.4)	(20.8, 4.2)
(2.3, 2.1)	-	<<	<<	<<
(6.7, 5.9)	>>	-	>>	<>
(4.5, 4.4)	>>	<<	-	<>
(20.8, 4.2)	>>	><	><	-

Tabela 4: Relação entre atributos para o cálculo do coeficiente de Kendall's τ.

A partir dessa tabela, observa-se que há oito pares concordantes (» e «) e quatro pares discordantes (<> e ><). Denominando nc_τ o número de pares concordantes e nd_τ o número de pares discordantes, tem-se $nc_\tau = 8$ e $nd_\tau = 4$. O coeficiente de Kendall's τ é definido pela Equação 13:

$$\tau(E_i, E_j) = \frac{nc_\tau - nd_\tau}{M(M-1)/2} \tag{13}$$

Assim, para os exemplos E_1 e E_2 considerados tem-se: $\tau(E_1, E_2) = \frac{8-4}{4\times\frac{3}{2}} = 0.666$.

A similaridade entre dois exemplos é definida pela Equação 14:

$$sim(E_i, E_j) = \tau(E_i, E_j) \tag{14}$$

Além das medidas aqui descritas, outras medidas de similaridade têm sido propostas. Uma revisão geral de vários tipos de medidas de similaridade pode ser encontrada em (Jain and Dubes, 1988) e (Everitt, 1993).

4.2 Algoritmos de *Clustering* Implementados

Como mencionado anteriormente existem diversas abordagens de *clustering*, entre as quais estão as abordagens particional ou de otimização e a hierárquica. Os algoritmos implementados na biblioteca Cluster 3.0, descritos a seguir, seguem essas duas abordagens.

4.2.1 Algoritmos de Particionamento

Os algoritmos de *clustering* particionais dividem os exemplos em k clusters de modo que a soma das distâncias entre os elementos de cada cluster e seus respectivos centróides seja mínima. Inicialmente, os algoritmos que utilizam essa abordagem selecionam aleatoriamente k exemplos e os utilizam como os centróides representantes dos clusters. Em seguida, o algoritmo atribui os exemplos restantes aos clusters cujos centróides estão mais próximos. Após, são executados iterativamente os seguintes três passos:

1. Atualizar os centróides;

2. Para cada exemplo, determinar o centróide mais próximo;

3. Redistribuir os exemplos nos clusters cujos centróides estejam mais próximos.

Esse processo iterativo pára quando não houver mais mudança nos clusters, *i.e.*, quando não for mais necessária a redistribuição dos exemplos para outros clusters.

Na biblioteca Cluster 3.0 há três métodos de *clustering* particionais implementados:

1. *k-means*;

2. *k-medians*; e

3. *k-medoids*.

Esses algoritmos diferem no modo como o centro do cluster é determinado. No *k-means*, por exemplo, o centro é definido como a média sobre todos os exemplos que pertencem ao cluster. No *k-medians*, ao invés de usar a média, a mediana é calculada para determinar o centróide. No *k-medoids*, por outro lado, o centro do cluster é definido como sendo o exemplo que apresenta a menor soma de distância para os outros exemplos do cluster.

Para esses algoritmos particionais, é necessário que sejam determinadas as k sementes iniciais, a partir das quais são gerados os k clusters finais. Usualmente, essas sementes são sorteadas entre os exemplos presentes no conjunto de dados. Na implementação desse procedimento nos algoritmos da biblioteca Cluster 3.0, foi utilizada uma outra biblioteca, chamada ranlib, que dispõe de rotinas para o cálculo de números aleatórios, a qual pode ser obtida através do endereço eletrônico `http://www.netlib.org/random/index.html`.

O processo iterativo de atualização e otimização dos clusters executado nos algoritmos particionais implementados na biblioteca Cluster 3.0, fazem uso do algoritmo *Expectation-Maximization* — (EM) (Dempster et al., 1977) — para maximizar a estimativa de parâmetros de modelos probabilísticos, nesse caso o modelo de misturas finitas. Para isso, esse algoritmo alterna entre duas etapas.

1. *Expectation* (etapa E): calcula os valores dos parâmetros esperados para as variáveis latentes; e

2. *Maximization* (etapa M): calcula a estimativa de probabilidade máxima desses parâmetros a partir de um conjunto de exemplos e atualiza as variáveis latentes com os valores obtidos na etapa E.

Uma mistura finita é um conjunto de k distribuições de probabilidades em que cada distribuição representa um cluster (Witten and Frank, 2000). A situação mais simples de um problema de misturas finitas acorre quando existe apenas um atributo numérico com distribuição normal e com diferentes médias e desvios padrão para cada cluster, conforme ilustrado na Figura 3. Nessa figura, são apresentados dois clusters, C_1 e C_2, que descrevem uma distribuição normal com médias e desvio padrão μ_{C_1}, σ_{C_1} e μ_{C_2}, σ_{C_2} respectivamente, assim como as probabilidades $P(C_1)$ para o cluster C_1 e $P(C_2)$ para

o cluster C_2. Desse modo, a cada exemplo do conjunto de dados é atribuída a probabilidade de possuir um certo conjunto de valores para seus atributos dado que esse exemplo é membro de um dos clusters. Nesse contexto, a tarefa do algoritmo *EM* é determinar o valor de cinco parâmetros de probabilidade para cada cluster, especificamente:

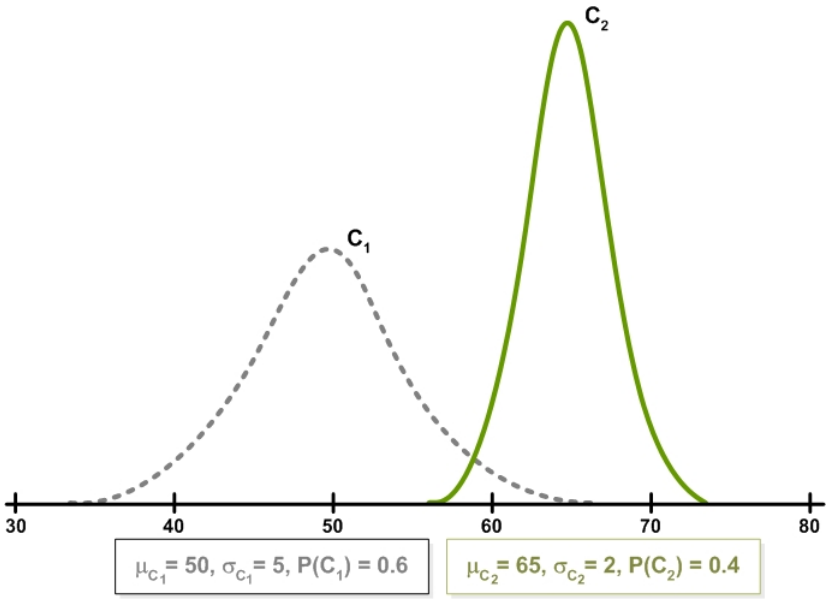

Figura 3: Modelo de misturas finitas Gaussianas (Witten and Frank, 2000).

1. a média e o desvio padrão do cluster 1 (μ_{C_1}, σ_{C_1});
2. a média e o desvio padrão do cluster 2 (μ_{C_2}, σ_{C_2}); e
3. a probabilidade do cluster 1, $P(C_1)$ (a probabilidade do cluster 2 é $1 - P(C_1)$).

A partir desses parâmetros, é possível calcular a probabilidade de cada exemplo ser atribuído a uma das distribuições. Dado um exemplo E_i, a probabilidade de ele pertencer ao cluster C_i é calculada por meio da Equação 15.

$$P(C_i|E_i) = \frac{P(E_i|C_i) \times P(C_i)}{P(E_i)} = \frac{f(E_i; \mu_{C_i}, \sigma_{C_i})}{P(E_i)} \tag{15}$$

onde $f(E_i; \mu_{C_i}, \sigma_{C_i})$ é a função de distribuição normal para o cluster C_i. Essa função é definida pela Equação 16.

$$f(E_i; \mu, \sigma) = \frac{1}{(\sqrt{2\pi}\sigma)} e^{-\frac{(E_i-\mu)^2}{2\sigma^2}} \tag{16}$$

O problema para a estimativa das probabilidades é que não se tem conhecimento, *a priori*, da distribuição dos dados e dos cinco parâmetros do modelo que se deve estimar. Então, a estratégia utilizada por esse algoritmo é similar a do algoritmo *k-means*, ou seja, o conjunto de parâmetros é inicialmente configurado com valores aleatórios que são recalculados enquanto a convergência desejada não é atingida. O procedimento geral pode ser descrito pelos seguintes três passos:

1. **Inicialização**: atribuir valores aleatórios aos cinco parâmetros;

2. **Etapa E**: Usar a função de densidade de probabilidade da distribuição normal para calcular a probabilidade do cluster para cada exemplo — Equação 16; e

3. **Etapa M**: usar os valores das probabilidades obtidos na etapa anterior para recalcular as estimativas dos parâmetros do modelo. Se o critério de parada não for satisfeito retornar à etapa E.

O algoritmo termina quando a função de otimização do *clustering* não apresentar mais mudanças significativas. No caso do algoritmo *k-means*, as iterações terminam quando não houver mais mudanças dos exemplos entre os clusters.

Como visto anteriormente, a identificação dos clusters é feita por meio de um processo iterativo de atribuição dos exemplos aos cluster e atualização dos centróides, e o processo é repetido até que não haja mais mudanças nos centróides. Esse processo é simples e muitas vezes eficiente, mas os clusters finais nem sempre representam uma solução global ótima, e sim uma solução ótima local. Além disso, esses algoritmos apresentam uma grande variância nos resultados, uma vez que a atribuição das sementes iniciais é feita de maneira aleatória, acarretando em soluções completamente diferentes a cada execução do algoritmo. Uma estratégia para aumentar as chances de encontrar uma solução ótima global, é executar várias vezes esse processo e atribuir o melhor resultado como a solução final.

4.2.2 Algoritmos Hierárquicos

Algoritmos de *clustering* hierárquico são essencialmente diferentes dos algoritmos particionais. Como visto anteriormente, os algoritmos hierárquicos descrevem os clusters por meio de uma estrutura de árvore conhecida como dendograma e, ao contrário dos particionais, o processo de *clustering* hierárquico é determinístico, pois não há variação nos resultados.

A biblioteca Cluster 3.0 implementa quatro métodos de *clustering* hierárquico:

1. *Single Link*;

2. *Complete Link*;

3. *Average Link*; e

4. *Centroid Link*.

Esses algoritmos utilizam a matriz de distâncias dos dados, a qual é atualizada a cada iteração do algoritmo, de modo que os valores de distância entre os novos clusters gerados e os outros clusters sejam corretos. Os algoritmos *Single*, *Complete* e *Average Link*, podem operar diretamente sobre a matriz de distâncias, não havendo a necessidade de acesso aos exemplos, *i.e.*, a tabela atributo-valor. Por outro lado, o método *Centroid Link* precisa acessar diretamente os exemplos para atualizar os centróides dos novos clusters, e então atualizar a matriz de distâcias. Essa matriz, em geral, é armazenada em uma

estrutura de N linhas e N colunas, onde N é o número de exemplos do conjunto de dados, como visto na Seção 2. Entretanto, ela pode ser armazenada em uma estrutura mais econômica, não utilizando as $N \times N$ posições de memória, mas apenas a porção triangular inferior. Nesse caso, a distância entre os elementos i e j pode ser encontrada na posição (i, j) se $j < i$, em (j, i) se $j > i$. Para $j = i$ a distância é zero. Essa estrutura de armazenamento é implementada nos algoritmos de *clustering* hierárquico da biblioteca Cluster 3.0, descritos brevemente a seguir.

***Single Link*:** nesse algoritmo, o conjunto de dados é inicialmente considerado um grafo esparso, no qual os exemplos representam os nós e as arestas formam um caminho entre os nós contidos num mesmo sub-grafo (cluster). Uma aresta conecta dois exemplos quando esses exemplos fazem parte do mesmo cluster com distância igual ao peso da aresta. A distância entre clusters, obtida por meio da técnica de vizinhos mais próximos (*Nearest Neighbor* — NN), é a menor distância entre os pares de exemplos dos clusters, sendo cada exemplo pertencente a um desses clusters. Assim, a junção ou agrupamento de dois clusters corresponde à adição de uma aresta entre o par de exemplos mais próximos, onde cada exemplo pertence a um cluster diferente, conforme ilustrado na Figura 4.

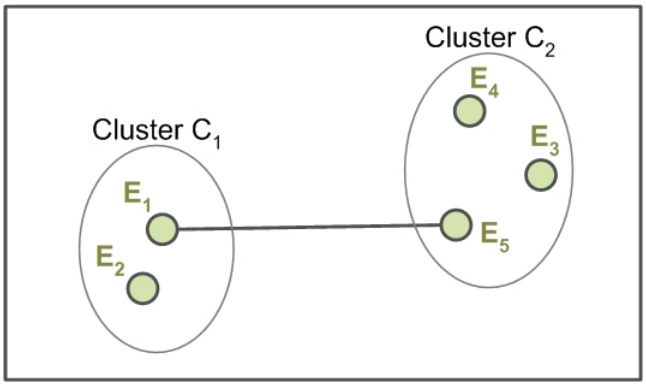

Figura 4: *Single Link*: menor distância entre dois clusters (Metz and Monard, 2006).

A implementação desse algoritmo na biblioteca Cluster 3.0 é baseada na proposta de Sibson (1973), que a torna mais eficiente em termos de tempo de execução e uso de recursos de memória, quando comparada com outras implementações desse mesmo algoritmo, pois essa implementação não faz alterações na matriz de distâncias mas utiliza um estratégia diferente que lhe garante maior eficiência.

***Complete Link*:** esse algoritmo é muito semelhante ao *Single Link*. Entretanto, a estratégia utilizada para calcular a distância entre dois clusters considera a maior distância entre um par de exemplos, sendo cada exemplo pertencente a um cluster distinto, conforme ilustrado na Figura 5.

O pseudo-código desse algoritmo é apresentado no Algoritmo 1.

Esse algoritmo recebe como entrada a matriz de distâncias (`distmatrix`) e o número de exemplos do conjunto de dados (`nElements`). Utiliza um

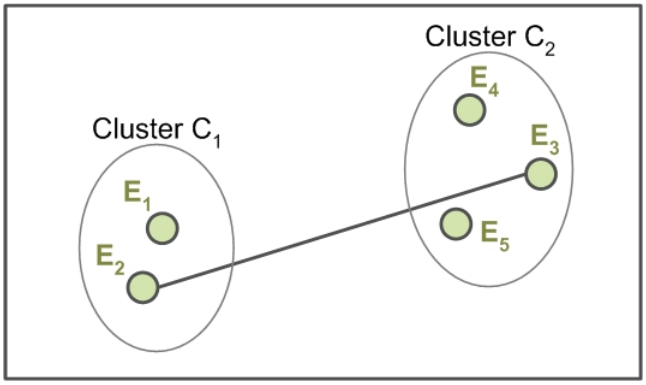

Figura 5: *Complete Link*: maior distância entre dois clusters (Metz and Monard, 2006).

Algorithm 1: Complete Link

Input: distmatrix, nElements
Output: result, linkdist

```
1  for j = 0; j < nElements; j++ do
2      clusterId[j] = j ;
3  end
4  nNodes = nElements ;
5  while nNodes > 1 do
6      linkdist[nElements - nNodes] = findClosest(distmatrix, iSaved,
       jSaved) ;
7      updateDistances(iSaved, jSaved) ;
8      extractLine(iSaved, distmatrix) ;
9      result[nElements - nNodes][0] = clusterId[iSaved] ;
10     result[nElements - nNodes][1] = clusterId[jSaved] ;
11     clusterId[jSaved] = nNodes - nElements - 1 ;
12     clusterId[iSaved] = clusterId[–nNodes] ;
13 end
```

vetor (`clusterId[]`) para armazenar a identificação dos clusters que inicialmente são unitários, cada um representado pelo exemplo que o constitui. Desse modo, o vetor `clusterId[]` é inicialmente configurado com o índice de cada exemplo nas suas respectivas posições — linhas 1 a 3 do Algoritmo 1. Em seguida, na linha 5, inicia-se o laço principal. As variáveis `iSaved` e `jSaved` são instanciadas para armazenar os índices dos clusters agrupados em cada iteração. Para isso, o procedimento `findClosest()` na linha 6 procura pelo par de clusters mais semelhantes e armazena seus identificadores em `iSaved` e `jSaved`. Além disso, esse procedimento retorna o valor da distância entre esses dois clusters e o atribui ao vetor `linkdist[]`. Esse último vetor contém os valores de distância resultantes de cada agrupamento de clusters.

Após a identificação do par de clusters a serem agrupados, a matriz de distâncias é atualizada na linha 7 por meio da chamada ao procedimento

`updateDistances()`, descrito em seguida. Nesse momento, o cluster representado por `jSaved` passa a ser o novo cluster, agrupando, além dos seus elementos, todos os exemplos que antes faziam parte do cluster `iSaved`. Em seguida, o procedimento `extractLine()` na linha 8 exclui a linha que representa o cluster `iSaved` da matriz de distâncias, pois esse cluster já não existe de maneira isolada, mas como parte do novo cluster representado por `jSaved`.

O Algoritmo 2 descreve o procedimento `updateDistances()`. Para entender melhor como funciona esse procedimento, suponha que `iSaved` represente o cluster cinco e `jSaved` o cluster três, para um conjunto de oito clusters. Nesse caso, os clusters cinco e três foram selecionados para o agrupamento e constituirão apenas um cluster representado por `jSaved`. Assim, a matriz de distâcias deve ser atualizada de modo que a distância do cluster `jSaved` a todos os outros clusters corresponda à distância do novo cluster aos demais.

Algorithm 2: Complete Link: updateDistances()

Input: distmatrix, iSaved, jSaved, nNodes
Output: distmatrix atualizada

```
1  i = iSaved ;
2  j = jSaved ;
3  sum = NiSaved + NjSaved ;
4  for k = 0; k < j; k++ do
5  |  distmatrix[j][k] = max(distmatrix[i][k], distmatrix[j][k]);
6  end
7  for k = j + 1; k < i; k++ do
8  |  distmatrix[k][j] = max(distmatrix[i][k], distmatrix[k][j]);
9  end
10 for k = i + 1; k < nNodes; k++ do
11 |  distmatrix[k][j] = max(distmatrix[k][i], distmatrix[k][j]);
12 end
13 for k = 0; k < i; k++ do distmatrix[i][k] = distmatrix[nNodes - 1][k]
14 for k = i + 1; k < nNodes - 1; k++ do distmatrix[k][i] = distmatrix[nNodes -
   1][k]
```

A Figura 6 representa a matriz de distâncias utilizada pelos algoritmos de *clustering* implementados na biblioteca Cluster 3.0. É importante lembrar que apenas a porção triangular inferior é instanciada, não sendo possível acessar os elementos da porção triangular superior da matriz. Além disso, os elementos da diagonal principal da matriz são todos zero. No procedimento de atualização da matriz, apenas os elementos da linha e coluna correspondentes ao cluster `jSaved` (cluster três), em cor cinza, devem ser alterados. Assim, o primeiro laço do procedimento (linhas 4

a 6 do Algoritmo 2) é responsável por atualizar os elementos na linha três da matriz de distâncias, representados pelo símbolo ♢ na Figura 6. Nos dois laços seguintes, linhas 7 a 9 e linhas 10 a 12, os valores dos elementos na coluna três da matriz, representados pelo símbolo ◯ e □ respectivamente, são alterados. Até esse ponto, a linha cinco da matriz, que representa o cluster `iSaved`, e a linha sete, que representa o último cluster nesse exemplo, não foram alteradas. As alterações nessas linhas da matriz são realizadas pelas instruções dos próximos dois laços.

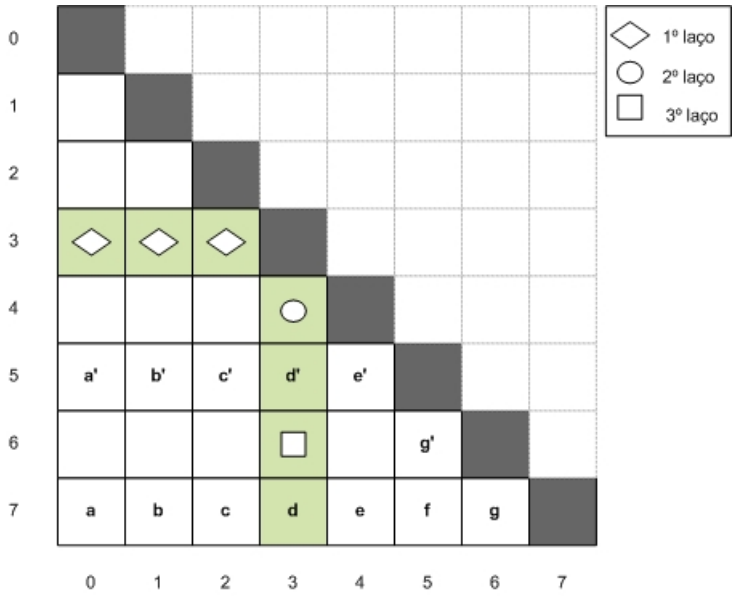

Figura 6: Matriz de distâncias.

O cluster `iSaved` não existirá isoladamente a partir desse agrupamento, conseqüentemente, será eliminado da matriz de distâncias. Para isso, as informações correspondentes ao cluster sete tomarão o lugar do cluster `iSaved`. O laço da linha 13 do Algoritmo 2 é responsável por executar essa alteração, atribuindo os valores da linha sete à linha cinco da matriz. Essa alteração é ilustrada na Figura 6 por meio da atribuição dos valores representados por **a**, **b**, **c**, **d** e **e** que estão na linha sete, no lugar dos valores **a'**, **b'**, **c'**, **d'** e **e'** na linha cinco. Entretanto, somente essa alteração não faz com que o cluster, antes representado na linha sete da matriz, tenha todos os valores de distância em relação aos outros clusters corretamente colocados na matriz. Para isso, é necessário que o valor de distância entre o cluster seis e sete seja substituído na linha seis, o que é feito pelo último laço no Algoritmo 2 (linha 14), atribuindo o valor representado pelo caractere **g** à posição $(6, 7)$ da matriz (linhas 6 e coluna 7), onde está o valor **g'**. Após esse laço, a última linha da matriz, que contém informação redundante, é desconsiderada completando o procedimento de atualização.

As estruturas que armazenam o resultado do *clustering* gerado a partir da execução do Algoritmo 1 são: `linkdist[]` e `result[]`. Esse último, contém os identificadores dos clusters envolvidos em cada evento de agrupamento e `linkdist[]` contém a distância que resultou desse agrupamento. Uma descrição mais detalhada dessa representação do *clustering*, conhecida como *Pointer Representation*, pode ser encontrada em (Metz and Monard, 2006).

***Average Link*:** os algoritmos *Complete Link* e *Single Link* (distância máxima e mínima respectivamente) representam dois extremos em termos de distância entre clusters. O algoritmo *Average Link*, por outro lado, faz uso de uma abordagem intermediária entre as duas anteriores, na qual a distância entre dois clusters é definida como a média das distâncias entre todos os pares de exemplos em cada cluster, ilustrado na Figura 7.

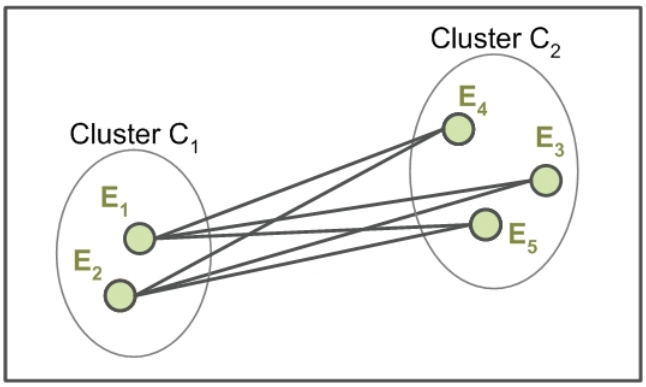

Figura 7: *Average Link*: média das distâncias entre os dois clusters.

Assim como no algoritmo *Complete Link* — Algoritmo 1 —, esse algoritmo recebe como entrada a matriz de distâncias e a quantidade de exemplos do conjunto de dados. Seu resultado também é armazenado nos vetores `linkdist[]` e `result[]` que formam a estrutura de dados *Pointer Representation*. Nesse algoritmo é necessário que o tamanho de cada cluster seja conhecido e atualizado a cada mudança nos agrupamentos. Para isso, um vetor é utilizado, no qual a posição i armazena a quantidade de exemplos contidos no cluster i.

A diferença entre os algoritmos *Average Link* e *Complete Link* está no modo como a matriz de distâncias é atualizada. Os elementos que devem ser alterados na matriz são exatamente os mesmos, entretanto, a maneira com que o novo valor das distâncias é calculado difere. Nesse caso, ao invés de utilizar o valor máximo da distâncias, o novo valor é obtido por meio da média entre as distâncias. Para isso, é necessário que o tamanho de cada cluster seja informado ao procedimento para o cálculo da média. Isso é feito por meio dos parâmetros `NiSaved` e `NjSaved` que represen-

tam, respectivamente, a quantidade de exemplos nos clusters `iSaved` e `jSaved`. O pseudo-código do procedimento `updateDistances()` utilizado no algoritmo *Average Link* é apresentado no Algoritmo 3.

Algorithm 3: Average Link: updateDistances()

Input: distmatrix, NiSaved, NjSaved, iSaved, jSaved, nNodes
Output: distmatrix atualizada

```
1  i = iSaved ;
2  j = jSaved ;
3  sum = NiSaved + NjSaved ;
4  for k = 0; k < j; k++ do
5  |  distmatrix[j][k] = distmatrix[i][k] × NiSaved + distmatrix[j][k] ×
   |  NjSaved ;
6  |  distmatrix[j][k] = distmatrix[j][k] / sum ;
7  end
8  for k = j + 1; k < i; k++ do
9  |  distmatrix[k][j] = distmatrix[i][k] × NiSaved + distmatrix[k][j] ×
   |  NjSaved ;
10 |  distmatrix[k][j] = distmatrix[k][j] / sum ;
11 end
12 for k = i + 1; k < nNodes; k++ do
13 |  distmatrix[k][j] = distmatrix[k][i] × NiSaved + distmatrix[k][j] ×
   |  NjSaved ;
14 |  distmatrix[k][j] = distmatrix[k][j] / sum ;
15 end
16 for k = 0; k < i; k++ do distmatrix[i][k] = distmatrix[nNodes - 1][k]
17 for k = i + 1; k < nNodes - 1; k++ do distmatrix[k][i] = distmatrix[nNodes -
   1][k]
```

***Centroid Link*:** os três algoritmos anteriores operam diretamente sobre a matriz de distâncias e não necessitam acesso aos valores dos atributos dos exemplos. Este algoritmo, em contrapartida, precisa acessar diretamente o valor de cada atributo doe exemplos. No método *Centroid Link*, os clusters são representados por um vetor (centróide) que armazena a média de cada atributo, o qual é utilizado como ponto de referência para o cálculo da distância entre os pares de clusters. Dessa maneira, a distância entre dois clusters é definida como sendo a distância entre seus centróides. Esse algoritmo opera de modo semelhante aos algoritmos anteriores. Também recebe como entrada a matriz de distâncias, o número de exemplos e armazena o resultado do *clustering* em uma estrutura de dados chamada *Pointer Representation*. Porém, o conjunto de exemplos também deve ser passado como parâmetro. A diferença entre algoritmo e os outros descritos anteriormente está relacionada ao cálculo de distâncias e a atualização dos centróides, cujo procedimento de atualização é apresentado no Algoritmo 4:

Algorithm 4: Centroid Link: updateCentroids()

Input: iSaved, jSaved, dismatrix, distId, nNodes, inode
Output: distmatrix atualizada

```
1  i = iSaved ;
2  j = jSaved ;
3  distId[i] = distId[nNodes - inode] ;
4  for k = 0; k < i; k++ do  distmatrix[i][k] = distmatrix[nNodes - inode][k]
5  for k = i + 1; i < nNodes - inode; k++ do
6      distmatrix[k][i] = distmatrix[nNodes - inode][k] ;
7  end
8  distId[j] = -inode - 1 ;
9  for k = 0 ; k < j; k++; do
10     distmatrix[j][k] = distance(data, j, k) ;
11 end
12 for k = j + 1; k < nNodes - inode; k++ do
13     distmatrix[k][j] = distance(data, j, k) ;
14 end
```

4.3 Arquivos Utilizados

O modo mais usual de armazenamento de dados para tarefas de aprendizado de máquina consiste em utilizar arquivos texto. Esses arquivos armazenam diversas informações sobre o conjunto de dados, tais como número de exemplos e atributos, rótulo para os exemplos e atributos, os exemplos propriamente ditos e mais alguma informação que seja relevante para a aplicação e/ou ferramenta utilizada. Nesta seção são apresentados brevemente os formatos de arquivos de entrada e saída utilizados pelo aplicativo de *clustering* disponível com a biblioteca Cluster 3.0, um utilitário com interface gráfica que faz uso das rotinas implementadas nessa biblioteca.

4.3.1 Arquivos de Entrada

Em geral, as ferramentas de *clustering* manipulam arquivos texto bastante simples, nos quais cada linha representa um exemplo. Os atributos desses exemplos são normalmente separados por vírgulas (“,”) ou “tabs” (caractere de tabulação não visível) e podem conter valores de diversos tipos e escalas. Além disso, algumas vezes o conjunto de dados pode estar incompleto, com informações faltantes ou mesmo desconhecidas. Como exemplo, considere o conjunto de dados descrito na Tabela 5.

Essa tabela representa a versão mais simples do arquivo de entrada reconhecido pelo aplicativo Cluster 3.0. A primeira célula da tabela (conteúdo **RAMDOM**), indica o nome da relação ou do conjunto de dados. No exemplo apresentado, o conjunto de dados foi gerado com números aleatórios. Se os dados dessa tabela representassem informações sobre esportes, o conteúdo dessa célula poderia ser **SPORTS** ou qualquer outro rótulo especificado pelo usuário. As outras células da primeira linha armazenam o rótulo dos atributos que descrevem os exemplos. Nesse caso, os atributos são X_1, X_2 e X_3. As células da primeira coluna, com exceção da primeira, contêm um identificador para cada exemplo, o qual pode ser qualquer conjunto de caracteres

RANDOM	X_1	X_2	X_3
E_1	1.000000	1.300000	2.400000
E_2	0.800000	2.100000	4.200000
E_3	1.100000	1.300000	0.800000
E_4	0.900000	0.800000	0.700000
E_5	1.200000	1.000000	1.100000
E_6	1.500000	2.000000	5.000000
E_7	0.700000	0.500000	2.400000
E_8	0.500000	0.900000	4.200000
E_9	1.100000	1.900000	3.300000
E_10	1.000000	0.700000	2.800000

Tabela 5: Exemplo 1: dados para o *clustering* com a biblioteca Cluster 3.0.

alfa-numérico. Nesse exemplo, os identificadores são E_i, onde $i = 1, 2, ..., 10$. As outras células, contêm os valores dos atributos. Além dessas informações, outras podem estar presentes no arquivo. Um exemplo completo de um arquivo no formato reconhecido pelo aplicativo Cluster 3.0 é apresentado na Tabela 6.

RANDOM	NAME	GWEIGHT	X_1	X_2	X_3
EWEIGHT			1	1	1
E_1	$exemplo_1$	1	1.000000	1.300000	2.400000
E_2	$exemplo_2$	1	0.800000	2.100000	4.200000
E_3	$exemplo_3$	1	1.100000	1.300000	0.800000
E_4	$exemplo_4$	1	0.900000	0.800000	0.700000
E_5	$exemplo_5$	1	1.200000	1.000000	1.100000
E_6	$exemplo_6$	1	1.500000	2.000000	5.000000
E_7	$exemplo_7$	1	0.700000	0.500000	2.400000
E_8	$exemplo_8$	1	0.500000	0.900000	4.200000
E_9	$exemplo_9$	1	1.100000	1.900000	3.300000
E_10	$exemplo_10$	1	1.000000	0.700000	2.800000

Tabela 6: Exemplo 2: dados para o *clustering* com a biblioteca Cluster 3.0.

A segunda linha, assim como as colunas dois e três são opcionais. A coluna com rótulo **NAME** permite armazenar uma descrição simples para cada exemplo, diferente do identificador na primeira coluna.

Por padrão, todos os exemplos/atributos são tratados de maneira idêntica. Entretanto, em algumas aplicações é interessante que um sub-conjunto de exemplos ou atributos tenham um peso maior que outros. Nesses casos, é possível especificar pesos por meio das células da linha **EWEIGHT** para os atributos; ou das células da coluna **GWEIGHT** para os exemplos. O valor padrão para os pesos é 1, *i.e.*, todos os exemplos/atributos possuem o mesmo peso.

4.3.2 Arquivos de Saída

O processo de *clustering* hierárquico executado por meio do aplicativo Cluster 3.0 resulta em até três arquivos com extensões `.cdt`, `.gtr` e `.atr`, descritos a seguir utilizando um conjunto de dados chamado `toy`.

- O arquivo `toy.cdt` (*clustered data table*), contém o conjunto de dados com suas linhas e colunas reordenadas de acordo com o resultado obtido pelo algoritmo e segue o formato de arquivo apresentado na Tabela 7.

Para a construção desse arquivo, uma nova coluna, que contém um identificador único para cada exemplo/atributo considerado no processo de *clustering*, é adicionada. Esse identificador é utilizado na criação da estrutura de árvore (dendograma) que é armazenado no arquivo `.gtr` e/ou `.atr`.

GID	RANDOM	NAME	GWEIGHT	X_1	X_2	X_3
EWEIGHT				1.000000	1.000000	1.000000
GENE3X	E_2	$exemplo_2$	1.000000	0.900000	0.800000	0.700000
GENE2X	E_4	$exemplo_4$	1.000000	1.100000	1.300000	0.800000
GENE4X	E_5	$exemplo_5$	1.000000	1.200000	1.000000	1.100000
GENE0X	E_1	$exemplo_1$	1.000000	1.000000	1.300000	2.400000
GENE6X	E_7	$exemplo_7$	1.000000	0.700000	0.500000	2.400000
GENE9X	E_10	$exemplo_10$	1.000000	1.000000	0.700000	2.800000
GENE5X	E_6	$exemplo_6$	1.000000	1.500000	2.000000	5.000000
GENE1X	E_3	$exemplo_3$	1.000000	0.800000	2.100000	4.200000
GENE8X	E_9	$exemplo_9$	1.000000	1.100000	1.900000	3.300000
GENE7X	E_8	$exemplo_8$	1.000000	0.500000	0.900000	4.200000

Tabela 7: Arquivo `toy.cdt`. Os exemplos são ordenados de acordo com o resultado do *clustering*.

- Os arquivos `.gtr` (*gene tree*) e `.atr` (*array tree*) são arquivos texto que armazenam o dendograma gerado a partir dos dados. O arquivo `.gtr` é criado somente quando o *clustering* é executado sobre os exemplos (linhas), enquanto que o arquivo `.atr` é criado somente quando o *clustering* considera os atributos (colunas) para o agrupamento.

  Esses arquivos são gerados da seguinte maneira: quando o processo de *clustering* começa, é atribuído um identificador único para cada exemplo, por exemplo GENE1X. Esses identificadores são adicionados ao arquivo `.cdt`, representados pela coluna *GID* (*gene id*) conforme ilustrado na Tabela 7. Ao passo que os nós são criados, eles recebem também um identificador único, começando a partir de NODE1X, NODE2X e assim sucessivamente. Cada evento de agrupamento é escrito no arquivo `.gtr` e/ou `.atr` com o identificador do novo nó, os identificadores dos clusters que resultaram no agrupamento e a distância que resultou na junção dos clusters envolvidos. A única diferença entre os arquivos `.gtr` e `.atr` são os rótulos utilizados nos identificadores. No caso do arquivo `.gtr`, os identificadores sempre seguem a forma GENE1X, enquanto que no arquivo `.atr` os identificadores são da forma ARRY1X. Um exemplo do arquivo `.gtr` é apresentado na Tabela 8. O dendograma escrito nos arquivos `.gtr`/`.atr` seguem a estrutura de dados *Pointer Representation*[5].

4.4 Interface com Linguagens de *Script*

A biblioteca Cluster 3.0 foi implementada na linguagem ANSI C. Entretanto, ela pode ser facilmente integrada com outros sistemas ou *scripts* implementados em outras linguagens de programação, tais como Perl (PERL, 1999), Python (Lutz, 2001) e TclTk (Welch et al., 2003). Juntamente com a implementação da biblioteca, encontram-se disponíveis extensões escritas em Perl e Python.

[5]A descrição desse formato pode ser encontrada em (Metz and Monard, 2006).

NODE1X	GENE4X	GENE2X	0.977173
NODE2X	GENE6X	GENE9X	0.965158
NODE3X	GENE3X	NODE1X	0.964557
NODE4X	NODE2X	GENE0X	0.924910
NODE5X	GENE8X	GENE1X	0.887064
NODE6X	GENE7X	NODE5X	0.777733
NODE7X	GENE5X	NODE6X	0.717661
NODE8X	NODE3X	NODE4X	0.627153
NODE9X	NODE7X	NODE8X	0.000000

Tabela 8: Arquivo `toy.gtr`.

O pacote `Algorithm::Cluster` (extensão Perl da biblioteca) dispões das seguintes funcionalidades:

- Algoritmos *k-means* e *k-medoids* para o *clustering* particional;
- Algoritmos *Single Link*, *Complete Link*, *Average Link* e *Centroid Link* para o *clustering* hierárquico;
- Redes auto-organizáveis, e;
- Métodos para o cálculo da média, mediana dos exemplos e da matriz de distâncias.

A extensão Python da biblioteca, pacote `Pycluster`, também apresenta essas funcionalidades. Outras extensões podem ser implementadas por meio da ferramenta SWIG — *Simplified Wrapper and Interface Generator*[6].

5 Projeto e Implementação do Módulo de *Clustering* Hierárquico do DISCOVER

Diversas ferramentas foram implementadas para a execução do *clustering*. Muitas dessas ferramentas são de domínio público, como *Cluto — Clustering Toolkit*[7] — e *Algorithm::Cluster*. No entanto, a integração dessas ferramentas de *clustering* com outras ferramentas de aprendizado é, em geral, um processo complexo. No DISCOVER (Baranauskas and Batista, 2000; Batista and Monard, 2003; Prati, 2003), por outro lado, estão disponíveis ferramentas utilizadas em todas as etapas do processo de MD, as quais são acessadas por meio do ambiente DLE - DISCOVER LEARNING ENVIRONMENT (Batista and Monard, 2003), composto pela biblioteca de classes do DISCOVER — DOL - DISCOVER OBJECT LIBRARY — e pelo ambiente de experimentos SNIFFER. Nesse contexto, o objetivo deste trabalho é apresentar o projeto e implementação de um módulo de *clustering* hierárquico que será integrado ao ambiente DISCOVER e fará uso das diversas facilidades já implementadas na biblioteca de classes desse ambiente.

Nesta Seção são apresentadas brevemente algumas características da biblioteca de classes do DISCOVER, do ambiente de gerenciamento de experimentos e avaliação experimental SNIFFER, assim como a descrição do projeto e

[6] http://www.swig.org/

[7] http://glaros.dtc.umn.edu/gkhome/views/cluto

implementação do módulo de *clustering* hierárquico proposto neste trabalho. Dentre as características apresentadas estão as bibliotecas utilizadas, a descrição dos arquivos de entrada e saída, como são tratados valores faltantes ou desconhecidos e o ambiente de experimentos e análise dos resultados do *clustering*.

5.1 A Biblioteca de Classes DOL

A biblioteca DOL - DISCOVER OBJECT LIBRARY (Batista and Monard, 2006) é uma biblioteca orientada a objetos baseada em padrões de projeto. As classes dessa biblioteca implementam as tarefas de manipulação e gerenciamento de dados mais comuns em pré-processamento, como gerenciamento de diferentes sintaxes de arquivos de dados e atributos, amostragens, métodos de *resampling*, estatísticas descritivas, normalizações de dados, entre outros. Essa biblioteca foi implementada com o objetivo de prover funcionalidades para as tarefas de pré-processamento de dados e dar suporte à criação de novos métodos de pré-processamento de dados. Para isso, a DOL foi desenvolvida em uma arquitetura modular na qual cada módulo é constituído de uma ou mais classes que realizam um conjunto bem definido de tarefas. Essa biblioteca possui um módulo central, chamado CORE, que é responsável por manter o conjunto de exemplos utilizado em um estrutura de dados específica do DISCOVER, DSX. Além disso, ela disponibiliza atualmente, mais de 60 métodos capazes de consultar e manipular essa estrutura. O módulo CORE é o único que, necessariamente, deve ser carregado por uma aplicação que utiliza a biblioteca, enquanto que os demais módulos são carregados apenas em casos que suas funcionalidades sejam necessárias.

O conjunto de rotinas da biblioteca DOL permite ao usuário/desenvolvedor uma forma simples de acesso e manipulação dos dados, os quais podem estar armazenados em arquivos texto ou em tabelas de bancos de dados relacionais. Entre as principais funcionalidades da biblioteca DOL pode-se citar: manipulação de atributos e exemplos, integração com diversos sistemas de aprendizados de máquina, integração com sistemas gerenciadores de bancos de dados, filtros de exemplos, estatísticas descritivas e correlações, métodos de *resampling* e conversores de sintaxes. Essas características são descritas brevemente a seguir.

Manipulação de atributos e exemplos: por meio dos métodos implementados na biblioteca DOL, é possível obter informações do conjunto de dados que está sendo pré-processado e efetuar diversas alterações nesses dados. Entre as principais tarefas de manipulação de atributos e dados pode-se citar: a alteração dos nomes de atributos, a conversão de tipos de atributos, a modificação de posição de atributos em um conjunto de dados, a alteração do atributo classe, a remoção temporária ou definitiva de atributos e/ou exemplos e a adição de exemplos ao conjunto de dados.

Integração com diversos sistemas de AM: a biblioteca DOL pode ser integrada com os principais sistemas de aprendizado acadêmicos e comerciais. Para isso, foram criados conversores de dados que transformam os dados da sintaxe DSX para as sintaxes de vários sistemas de aprendizado, entre eles: $\mathcal{C}4.5$ e $\mathcal{C}4.5rules$ (Quinlan, 1993), $ID3$ (Quinlan, 1986) e $\mathcal{CN}2$ (Clark and Niblett, 1989; Clark and Boswell, 1991).

Integração com sistemas de bancos de dados: como dito anteriormente, os dados podem estar armazenados em arquivos texto ou em tabelas de sistemas gerenciadores de bancos de dados. A biblioteca DOL dispõem de métodos que permitem a sua integração com os gerenciadores de bancos de dados mais freqüentemente utilizados, como MySQL, Oracle e Postgres.

Filtro de Exemplos e Atributos: a biblioteca DOL oferece um sistema de filtros que permite ocultar temporariamente parte dos exemplos ou dos atributos de um conjunto de dados. Os filtros podem ser criados a partir de expressões lógicas envolvendo atributos ou indicando explicitamente, a partir de uma lista, os exemplos que devem ser filtrados.. Essa funcionalidade é a base para a implementação dos métodos para seleção aleatória e estratificada de amostras.

Estatísticas Descritivas e Correlações: estatísticas descritivas podem ser utilizadas tanto pelo usuário quanto por métodos de pré-processamento de dados. Ainda, informações sobre correlações entre atributos, como correlação linear e covariância, são a base para a construção de diversos métodos de pré-processamento de dados.

Métodos de *Resampling*: são métodos estatísticos capazes de estimar apropriadamente a taxa de erro verdadeira, mesmo em um conjunto reduzido de exemplos. Atualmente, a biblioteca de classes DOL provê dois métodos de *resampling*: a validação cruzada e a validação cruzada extratificada.

Conversores: os diversos sistemas de aprendizado de máquina integrados com o DISCOVER fazem uso de diferentes sintaxes de representação dos dados, o que acarreta a necessidade de implementar conversores capazes de transformar os dados que estão na sintaxe padrão do DISCOVER para o formato dos diferentes sistemas utilizados e, desses formatos para a sintaxe DSX.

5.2 O Gerenciador de Experimentos SNIFFER

Freqüentemente, para avaliar e comparar algoritmos de aprendizado é necessário executar um experimento diversas vezes. Diante de tantas execuções, torna-se muito difícil gerenciar os experimentos manualmente, sendo necessário automatizar essa tarefa. Por esse motivo, o ambiente computacional SNIFFER (Batista and Monard, 2003) foi desenvolvido e integrado ao DISCOVER podendo acessar todos os algoritmos e ferramentas disponíveis nesse ambiente. Assim, é possível comparar o desempenho de novos métodos de pré-processamento de dados aplicados a diversos sistemas de aprendizado.

Uma outra característica importante do SNIFFER é automatizar, sempre que possível, a publicação dos resultados. Como freqüentemente os resultados dos experimentos envolvem uma grande quantidade de valores numéricos, é muito comum a introdução de erros durante o processo manual de confecção de tabelas e gráficos. Nesse sentido, o SNIFFER garante ao usuário que os dados publicados são fiéis aos resultados produzidos nos experimentos. Essa garantia é obtida a partir dos relatórios resumidos e detalhados dos resultados obtidos, além da tabulação desses resultados em um formato que pode

ser utilizado para gerar gráficos com a ferramenta Gnuplot[8] e tabelas com o processador de textos LaTeX [9]. As principais funcionalidades desse ambiente são o gerenciamento das sintaxes dos diferentes sistemas de aprendizado, aplicação de métodos de *resampling*, recuperação de taxas de erro, cálculos e comparação de medidas de desempenho.

O ambiente SNIFFER complementa a biblioteca DOL, pois permite que diferentes indutores proposicionais e métodos de pré-processamento de dados sejam avaliados e comparados experimentalmente de uma forma rápida e segura. Um fator importante que possibilitou essa integração das diferentes técnicas e ferramentas de aprendizado de máquina em um ambiente único, como é o caso do DISCOVER, foi a elaboração de uma sintaxe padrão para a representação dos dados, chamada de DSX - DISCOVER STANDARD SINTAX, descrita na Seção 5.3.

Outras características do ambiente computacional DLE - DISCOVER LEARNING ENVIRONMENT, tais como detalhes de arquitetura, projeto e implementação do ambiente, os quais estão fora do escopo deste trabalho, podem ser encontrados em (Batista and Monard, 2003).

5.3 DSX - DISCOVER STANDARD SINTAX

A sintaxe padrão do DISCOVER utiliza arquivos texto para declarar atributos e seus respectivos tipos, e os valores que esses atributos assumem em um conjunto de exemplos. Os nomes dos atributos são declarados em um arquivo de nomes, identificados pela extensão `.names`. Os valores que esses atributos assumem em cada exemplo do conjunto de dados são declarados em outro arquivo com a extensão `.data`. Os dois arquivos devem ter o mesmo nome, se diferenciando apenas pela extensão. Assim, um conjunto de exemplos somente está na sintaxe padrão DSX se esses dois arquivos estiverem presentes. A sintaxe dos arquivos de nomes (`.names`) e dados (`.data`) é uma extensão do formato utilizado pelo algoritmo $C4.5$. Essa extensão tem como principal objetivo adequar o formato dos arquivos com as necessidades do projeto DISCOVER.

A primeira declaração em um arquivo de nomes define qual deve ser o nome do atributo classe, se ele existir. Se o atributo classe não existir, então o nome `null` deve ser utilizado. Em outras palavras, `null` indica que o conjunto de exemplos correspondente é não-supervisionado.

No caso do atributo classe existir, ele pode ser qualquer atributo presente no conjunto de exemplos. Após a declaração do atributo classe, são declarados os demais atributos com seus identificadores e domínio associado. São válidos os identificadores que são combinações de números, letras e "_" (sublinhado), em qualquer seqüência. Para identificadores mais complexos que envolvem outros caracteres que não sejam os especificados anteriormente (como espaços, letras acentuadas, etc) é necessário colocar o identificador entre aspas. Desse modo, são identificadores válidos: `abc`, `1`, `1a`, `_1a`, "`_12a`", "`válido`". Para ilustração, considere o conjunto de dados artificial `toy` — Tabela 9 —, com três atributos `X_1`, `X_2` e `X_3` cujos valores foram gerados aleatoriamente. Nesse conjunto de dados não há um atributo classe. O arquivo `toy.names` é mostrado na Tabela 10.

[8] http://www.gnuplot.org
[9] http://www.tug.org/tetex/

1.000000,	1.300000,	2.400000
0.800000,	2.100000,	4.200000
1.100000,	1.300000,	0.800000
0.900000,	0.800000,	0.700000
1.200000,	1.000000,	1.100000
1.500000,	2.000000,	5.000000
0.700000,	0.500000,	2.400000
0.500000,	0.900000,	4.200000
1.100000,	1.900000,	3.300000
1.000000,	0.700000,	2.800000

Tabela 9: Conjunto de dados `toy`.

```
null.

X_1:    real.
X_2:    real.
X_3:    real.
```

Tabela 10: Arquivo `toy.names`.

O arquivo de dados correspondente ao arquivo de nomes contém N exemplos, sendo um exemplo por linha. Cada exemplo E_i consiste dos valores dos atributos desse exemplo, separados por vírgula. Os valores de cada atributo devem pertencer ao domínio correspondente, especificado no arquivo de nomes (`.names`). Desta maneira, o "separador de registros" é o caracter de nova linha (representado em muitas linguagens de programação por "`\n`") e o separador de campos é a vírgula. Cada valor presente em uma linha está associado a um atributo do arquivo de nomes. Sendo assim, a ordem em que os valores são colocados em uma determinada linha deve ser a mesma na qual os atributos são declarados no arquivo de nomes. Valores "desconhecidos" são representados por um ponto de interrogação (?) e valores *"não-se-aplica"* são representados com um ponto de exclamação (!).

Os tipos de dados definidos até o momento na sintaxe DSX são:

- *Nominal*;
- *Enumerated*;
- *Integer*;
- *Real*;
- *String*;
- *Date*; e
- *Time*.

Uma descrição mais detalhada desses tipos de dados pode ser encontrada em (Batista and Monard, 2003) ou através do endereço eletrônico `http://www.icmc.usp.br/~gbatista`.

É importante lembrar que com esse formato padrão definido para o DISCOVER, os conjuntos de dados podem ser utilizados tanto em tarefas de aprendizado supervisionado como também em tarefas de aprendizado não-supervisionado.

5.4 Descrição do Módulo de *Clustering* Hierárquico

Como mencionado, para o desenvolvimento do módulo de *clustering* hierárquico do DISCOVER foram utilizadas diversas facilidades disponíveis no ambiente computacional DLE - DISCOVER LEARNING ENVIRONMENT, cujo principal objetivo é prover um *framework* para facilitar a implementação e avaliação experimental de sistemas de aprendizado de máquina a serem integrados ao DISCOVER. Esse ambiente é composto por dois módulos distintos: DOL - DISCOVER OBJECT LIBRARY, uma biblioteca de classes que contém a implementação de métodos para o pré-processamento de dados; e o ambiente para gerenciamento de experimentos SNIFFER, utilizado para automatizar avaliações e comparações experimentais de algoritmos de aprendizado de máquina, ambos descritos nas seções anteriores. É importante observar que este módulo de *clustering* não faz uso do SNIFFER, pois implementa seu próprio ambiente para execução de experimentos, chamado SHELL*Cluster*. Essa decisão foi tomada com base na necessidade de maior interação com o usuário, uma vez que o SNIFFER não propicia tal característica.

A integração de um módulo de *clustering* em um ambiente tal como o DISCOVER, propicia o usuário/pesquisador maior flexibilidade para a execução de experimentos e para análise dos resultados. Com o auxílio das ferramentas de pré-processamento disponíveis no DISCOVER, o usuário pode organizar e gerenciar melhor seus experimentos, utilizando uma sintaxe padrão em todas as etapas do processo de MD. Além disso, pode-se utilizar as ferramentas implementadas nesse ambiente para avaliar os resultados e gerar regras que auxiliam na explicação do conhecimento extraído pelos algoritmos de *clustering*.

Os algoritmos de *clustering* utilizados neste módulo são os implementados na biblioteca Cluster 3.0 — Seção 4. Apesar de essa biblioteca dispor de um aplicativo executável para a realização do *clustering*, optamos por desenvolver outro aplicativo mais simples, o DH*Cluster*, que somente utiliza os procedimentos de *clustering* hierárquico da biblioteca Cluster 3.0, ignorando os outros procedimentos disponíveis. Além disso, nessa implementação optamos por simplificar o formato dos arquivos de entrada para a execução do *clustering*. Nesse sentido, a sintaxe utilizada pelo aplicativo disponível na biblioteca Cluster 3.0 difere da sintaxe adotada pelo DH*Cluster* e, também, da sintaxe DSX. Essas diferenças acarretaram a implementação de conversores que pudessem transformar os dados da sintaxe DSX para o formato reconhecido pelo programa DH*Cluster*.

Como dito anteriormente, a biblioteca Cluster 3.0 permite que sejam utilizados conjuntos de dados com valores faltantes, e, também, que sejam atribuídos pesos para os atributos e exemplos. O formato utilizado pelo aplicativo Cluster 3.0 possibilita que todas essas informações sejam armazenadas em um único arquivo juntamente com a declaração dos exemplos. No formato do aplicativo DH*Cluster*, essas informações devem ser armazenadas em arquivos separados, tornando mais simples a manipulação desses arquivos e a construção de *parsers* para os mesmos. Assim, dois arquivos são utilizados pelo DH*Cluster*: `.dat` e `.mask`. O arquivo `.dat` armazena os dados e as configurações de peso para os atributos e exemplos. Nesse arquivo, a primeira linha contém as pesos dos atributos separados por espaços em branco. Da segunda à ultima linha são armazenadas as informações de pesos dos exemplos (primeira coluna) e os valores dos atributos para cada exemplo (demais

colunas), também separados por espaços em branco. Na Tabela 11 é ilustrado um conjunto de dados no formato reconhecido pelo aplicativo DH*Cluster*. O valor padrão para os pesos é 1 (um), indicando que todos os exemplos/atributos são tratados igualmente. Porém, os pesos podem assumir qualquer valor de tipo real.

1	1	1	
1	1.000000	1.300000	2.400000
1	0.800000	2.100000	4.200000
1	1.100000	1.300000	0.800000
1	0.900000	0.800000	0.700000
1	1.200000	1.000000	1.100000
1	1.500000	2.000000	5.000000
1	0.700000	0.500000	2.400000
1	0.500000	0.900000	4.200000
1	1.100000	1.900000	3.300000
1	1.000000	0.700000	2.800000

Tabela 11: Arquivo `toy.dat`: entrada para o aplicativo DH*Cluster*.

No arquivo `.mask` correspondente — Tabela 12 —, cada linha representa um exemplo do conjunto de dados e as colunas, separadas por espaços, representam os atributos. É importante observar que nesse caso, os valores armazenados não correspondem aos valores dos atributos, mas indicam quais atributos devem ser considerados para o cálculo da distância. Assim, o valor 1 (um) indica que esse atributo está presente no respectivo exemplo, enquanto que o valor 0 (zero) indica que esse valor não foi informado ou é desconhecido e, portanto, não deve ser considerado no processo de *clustering*. Na sintaxe DSX existe, também, a possibilidade de utilizar o valor *"não-se-aplica"* para os atributos. Esse valor é convertido para 0 (zero) durante a criação do arquivo `.mask`.

1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1

Tabela 12: Arquivo `toy.mask`.

Com o objetivo de melhorar a integração do ambiente DLE - DISCOVER LEARNING ENVIRONMENT com os conversores implementados e o aplicativo DH*Cluster*, e, também, a interação do usuário como módulo de *clustering* do DISCOVER, implementamos, utilizando a linguagem de programação Perl (PERL, 1999), uma biblioteca (módulo Perl) e um *script* para execução dos experimentos e análise dos resultados do *clustering*. Esse módulo Perl integra a biblioteca DLE com o aplicativo DH*Cluster* e agrega outras funcionalidades que auxiliam na execução dos experimentos e avaliação dos resultados. Nas próximas Seções são apresentados alguns detalhes das funcionalidades do módulo CLUSTER.PM e do *script* SHELL*Cluster*.

5.4.1 O Módulo CLUSTER.PM

A necessidade de manipulação dos dados, como a conversão entre formatos, normalizações de dados, seleção de atributos e outras tarefas de pré-processamento para o *clustering*, motivaram o desenvolvimento de um módulo capaz de encapsular essas funcionalidades. Assim, diversos procedimentos para preparação de dados e análise do *clustering* podem ser integrados em um único ambiente acessível para que outras ferramentas sejam codificadas e façam uso das facilidades disponíveis nesse módulo —Figura 8

Figura 8: Módulo CLUSTER.PM

Com a implementação em linguagem Perl, a criação de *scripts* torna-se bastante simples, permitindo maior flexibilidade ao usuário/desenvolvedor para utilização do módulo de *clustering* hierárquico do DISCOVER. Além disso, possibilita que novos algoritmos sejam utilizados, desde que conversores para os respectivos formatos sejam implementados.

Nesse módulo Perl, implementamos procedimentos de conversão do formato DSX para o formato reconhecido pelo aplicativo DH*Cluster*, leitura e cálculo da matriz de distâncias dos dados, procedimento para o corte horizontal em diferentes níveis da hierarquia do dendograma, estatísticas descritivas sobre os clusters identificados e a criação de arquivos supervisionados, nos quais, o conjunto de dados é rotulado utilizando os clusters como atributo classe.

5.4.2 O Ambiente de Gerenciamento de Experimentos e Análise do *Clustering*

Para tornar mais interativo o processo de execução dos experimentos e análise do *clustering*, implementamos um ambiente de gerenciamento de experimentos e análise dos agrupamentos gerados pelos experimentos realizados com o módulo de *clustering* do ambiente DISCOVER. O SHELL*Cluster* pode ser visto como uma camada sobre o ambiente DLE e o módulo Perl CLUSTER.PM, conforme ilustrado na Figura 9.

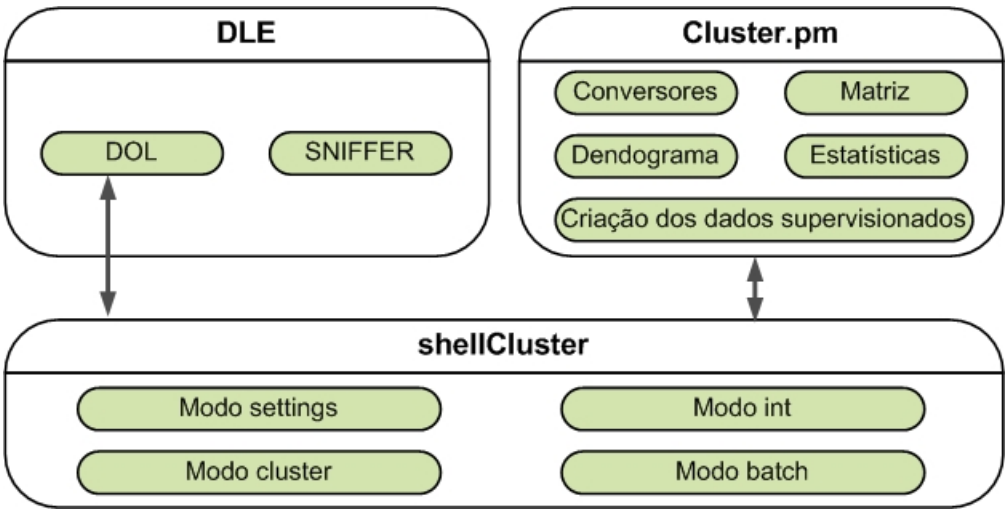

Figura 9: Ambiente de execução de experimentos SHELL*Cluster*.

Esse *script* possibilita a realização de experimentos de maneira interativa, por meio da alteração dos parâmetros de execução e análise, simplificando muito o processo de execução de experimentos, característica fundamental na área de aprendizado de máquina. Para isso, estão disponíveis quatro modos de operação: `settings`, `cluster`, `int` e `batch`, descritos a seguir:

`modo settings`: nesse modo de operação, são definidos os parâmetros utilizados para execução do *clustering* e também na análise dos resultados. Além disso, é nesse modo que o conjunto de dados é carregado para a memória. São sete os comandos disponíveis ao usuário:

1. `setup`;
2. `transpose`;
3. `rtranspose`;
4. `distmatrix`;
5. `rdistmatrix`;
6. `load`; e
7. `view`.

O comando `setup` é utilizado para definir os parâmetros de execução do algoritmo de *clustering*. A sintaxe desse comando é apresentada a seguir:

```
setup [-f FILE] <[-d OPTION] [-m OPTION] [-ip PATH] [-op PATH]>
```

O parâmetro `-f`, define o nome do conjunto de dados utilizado para o experimento. Esse conjunto de dados deve estar no formato especificado pela sintaxe DSX — Seção 5.3.

O parâmetro `-d`, define qual medida de similaridade será utilizada pelo aplicativo DH*Cluster*. Além das oito medidas implementadas na biblioteca Cluster 3.0 (Seção 4), é possível também utilizar a medida de distância euclidiana quadrada. Na Tabela 13 são apresentadas as nove opções de medidas de similaridade disponíveis, juntamente com os caracteres

Identificador	Medida
e	Distância Euclidiana
w	Distância Euclidiana Quadrada
b	Distância de Manhattam
c	Correlação de Pearson
a	Correlação de Pearson Absoluta
u	Correlação de Pearson Não-Centralizada
x	Correlação de Pearson Não-Centralizada Absoluta
s	Correlação de Spearman's
k	Kendall's tau

Tabela 13: Identificadores das medidas de similaridade.

utilizados para identificar cada uma delas. Se o valor especificado para o parâmetro `-d` é diferente dos apresentados nessa tabela, o valor padrão `s` — correlação de Spearman — é utilizado.

O parâmetro `-m` define qual algoritmo será utilizado para *clustering*. As opções são apresentadas na Tabela 14.

Identificador	Algoritmo
s	*Single Link*
m	*Complete Link*
a	*Average Link*
c	*Centroid Link*

Tabela 14: Identificadores dos algoritmos de *clustering* hierárquico.

Os parâmetros `-ip` e `-op` definem, respectivamente, o endereço do diretório onde estão os arquivos de entrada (`.names` e `.data`) e o endereço do diretório no qual devem ser colocados os arquivos de saída do *clustering*.

Como observado, são cinco os parâmetros que podem ser definidos pelo comando `setup`, dos quais apenas um é obrigatório e os demais são opcionais. Assim, os parâmetros entre "<" e ">" não precisam ser informados, pois os mesmos possuem valores padrão que são utilizados na execução dos experimentos quando nenhum outro valor é informado. O valor padrão para cada parâmetro é apresentado na Tabela 15.

Parâmetro	Valor padrão
`-ip` (diretório dos arquivos de entrada)	~/workspace/experiments/input/
`-op` (diretório dos arquivos de saída)	~/workspace/experiments/output/
`-d` (medida de distância)	s
`-m` (método de *clustering*)	a

Tabela 15: Valores padrão para os parâmetros de execução do *clustering*.

Além do comando `setup`, outros seis comandos podem ser utilizados no modo `settings`. O comando `transpose` indica que o *clustering* deve ser realizado sobre as colunas (atributos), ao invés das linhas (exemplos) do conjunto de dados. O comando `rtranspose`, por outro lado, faz exatamente o contrário, *i.e.*, as linhas são utilizadas e não mais as colunas. Por padrão, o *clustering* é executado considerando as linhas, e só realiza o *clustering* sobre as colunas se for definido pelo comando `transpose`.

De maneira análoga, são utilizados os dois comandos `distmatrix` e `rdistmatrix`. O primeiro indica que a matriz de distâncias já foi calcula e está disponível no arquivo `.matrix` no diretório especificado pelo parâmetro `-op`. Já o segundo comando `rdistmatrix`, indica que a matriz ainda não foi calculada. Por padrão, o SHELL*Cluster* considera que a matriz de distâncias não foi previamente calculada, e, então, constrói essa matriz e a armazena no arquivo `.matrix` no diretório de saída.

O comando `load` certifica que os parâmetros passados são corretos e que os arquivos de entrada existem. Em seguida, carrega o conjunto de exemplos para a memória principal para que seja executado o *clustering*. O comando `view` simplesmente lista as opções selecionadas pelo usuário.

modo cluster: nesse modo de operação o usuário pode executar o *clustering* considerando as definições feitas no modo `settings`. Há apenas um comando nesse modo: `run`. A execução desse comando, faz com que o aplicativo DH*Cluster* seja executado sobre o conjunto de dados selecionado e os resultados armazenados no diretório de saída especificado pelo parâmetro `-op` no modo `settings`.

Os arquivos gerados recebem o nome de acordo com o conjunto de dados selecionado. Desse modo, os arquivos com extensão `.link`, `.result`, `.tree` e `.matrix` são criados no diretório de saída especificado no modo `settings`. O nome desses arquivos segue o formato:

```
<dataset>.<-m>.<-d>.<extensão>
```

onde, `<dataset>` é o nome do conjunto de dados, `<-m>` e `<-d>` são os parâmetros configurados no modo `settings`, respectivamente, ao algoritmo e medida de similaridade utilizados e, `<extensão>` é a extensão do arquivo. Além desses arquivos, o arquivo `<dataset>.dat` também é escrito no diretório de saída especificado pelo usuário. Esse arquivo contém os dados no formato aceito pelo aplicativo DH*Cluster*.

modo int: nesse modo de operação é possível realizar a análise e interpretação dos resultados gerados pelo algoritmo de *clustering*. Um fator que diferencia o *clustering* hierárquico das outras abordagens é a maneira com que o resultado pode ser analisado, *i.e.*, a flexibilidade em relação à analise dos diferentes níveis de granularidade ou densidade dos agrupamentos faz com que, por meio do *clustering* hierárquico, seja possível encontrar as estruturas intrínsecas que estão embutidas nos dados. As outras abordagens, por outro lado, não são capazes de identificar essas estruturas, pois utilizam parâmetros globais de densidade ou funções matemáticas que não consideram os diferentes níveis de granularidade dos agrupamentos (Metz and Monard, 2006). Nesse contexto, um corte horizontal no dendograma em diferentes níveis gera diferentes agrupamentos. Com bases nesses agrupamentos, pode-se tentar descobrir relações ou padrões entre os exemplos que pertencem aos clusters identificados e atribuir conceitos a esses clusters. Nos casos em que não for possível identificar um conceito geral para todo o cluster, pode-se refinar a análise subdividindo o cluster em outros menores. Esse procedimento pode ser realizado nesse modo de operação do SHELL*Cluster*. Para isso, foram implementados seis comandos, descritos a seguir.

1. `setroot`: define o nó raiz a partir do qual será realizado o corte no dendograma. Sua sintaxe é:

   ```
   setroot <root>
   ```

   onde, `<root>` é um valor inteiro que identifica o número de um nó do dendograma. Esse valor deve estar no intervalo $[N+1,\ 2\times N-1]$, onde N é o número de exemplos do conjunto de dados. O valor padrão para esse parâmetro é a raiz absoluta do dendograma.

2. `setk`: define o número de clusters que deve ser gerado pelo corte no dendograma. Sua sintaxe é:

   ```
   setk <k>
   ```

   onde, `<k>` deve ser um valor inteiro. O valor padrão para esse parâmetro é 5 (cinco).

3. `cuttree`: a partir do resultado do *clustering* hierárquico, o comando `cuttree` divide o dendograma considerando os parâmetros especificados pelos comandos `setroot` e `setk`. Além do corte no dendograma, esse comando cria um novo conjunto de dados supervisionado, no qual o atributo classe corresponde ao cluster ao qual cada exemplo foi atribuído. É importante observar que apenas os exemplos pertencentes aos clusters presentes nos nós abaixo do nó raiz indicado pelo parâmetro `<root>` são utilizados para a criação desse novo conjunto de dados.

   Os arquivos gerados recebem o nome de acordo com o nome do conjunto de dados. Por exemplo, se o conjunto de dados `random` for utilizado, os arquivos criados por esse comando receberão o nome de:

   ```
   random.<-m>.<-d>.root<root>.k<k>.names;
   random.<-m>.<-d>.root<root>.k<k>.data; e
   random.<-m>.<-d>.root<root>.k<k>.test.
   ```

   onde `<-m>` e `<-d>` são os parâmetros correspondentes ao algoritmo utilizado e à medida de distância, respectivamente, e, `<root>` e `<k>` referenciam o nó raiz e o número de clusters utilizados para o corte no dendograma. Esses arquivos são formatados de acordo com a sintaxe DSX.

4. `runinducer`: esse comando executa o indutor selecionado pelo usuário com seus parâmetros padrão. Para isso, utiliza o conjunto de dados criado pelo comando `cuttree`. O resultado da indução é colocado no diretório de saída dos experimentos especificados pelo parâmetro `-op` no modo `settings`. A sintaxe desse comando é:

   ```
   runinducer <inducer>
   ```

   onde, `<inducer>` pode ser `C4.5`, `C4.5rules` ou `CN2`.

5. `viewrules`: esse comando apenas apresenta as regras geradas pelo indutor selecionado.

6. `viewreport`: esse comando apresenta o conteúdo do arquivo `report`, que contém as informações da execução do *clustering*.

modo batch: esse modo de operação foi implementado para facilitar ao usuário a execução do *clustering* sobre vários conjuntos de dados de uma única vez. Assim, basta ao usuário definir um arquivo contendo os nomes dos conjuntos de dados e outro arquivo contendo os parâmetros de configuração do *clustering* para que o SHELL*Cluster* execute o *clustering* sobre os conjuntos de dados especificados. Três comandos estão disponíveis nesse modo:

1. `dsfile`: esse comando define o endereço e o nome do arquivo que contém a relação dos conjuntos de dados e do arquivo que contém os parâmetros para execução de *clustering*. O arquivo com a relação dos conjuntos de dados é identificado pela extensão `.ds`. Seu formato é bastante simples, deve conter apenas um nome do conjunto de dados por linha e a última linha do arquivo deve ter a palavra `close`, indicando o fim da lista de arquivos. De maneira análoga, o arquivo de parâmetros é identificado pela extensão `.p` e contém a palavra chave `close` na última linha. Os parâmetros devem ser especificados, um em cada linha do arquivo, segundo a sintaxe: `<parâmetro>: <valor>`. Um exemplo dos arquivos `.ds` e `.p` é mostrado nas Tabelas 16 e 17, respectivamente.

```
bupa
vehicle
german
hungaria
pima
close
```

Tabela 16: Arquivo `.ds` para a execução do *clustering* no modo `batch`.

```
-ip: ~/workspace/experiments/input/
-op: ~/workspace/experiments/output/
-d:  s
-m:  a
close
```

Tabela 17: Arquivo `.p` para a execução do *clustering* no modo `batch`.

   A sintaxe para execução desse comando é:

   ```
   dsfile <path> <file name>
   ```

   onde, `<path>` é o endereço do diretório onde estão os arquivos com extensão `.ds` e `.p`; e `<file name>` é o nome desses arquivos sem extensão.

2. `run`: esse comando executa o *clustering* sobre os conjuntos de dados listados no arquivo `.ds` utilizando os parâmetros definidos no arquivo `.p`. Assim como nas execuções do *clustering* no modo `cluster`, os arquivos gerados pelo aplicativo DH*Cluster* a partir da execução desse comando são gravados no diretório de saída definido pelo parâmetro `-op`, ou no diretório padrão caso não tenha sido definido no arquivo de parâmetros. A sintaxe para execução desse comando é

```
run
```

3. `job`: freqüentemente, é necessário executar o *clustering* sobre vários conjuntos de dados e com diferentes parâmetros de configuração distintos. Essa necessidade é suprida pelo comando `job`, cuja sintaxe é

   ```
   job <path>/<file name>
   ```

   onde, `<path>/<file name>` indica o endereço completo para o arquivo com extensão `.job`. Nesse arquivo, a primeira linha contém o endereço do diretório onde estão armazenados os arquivos com a listagem de conjuntos de dados (`.ds`) e parâmetros (`.p`) para a execução do *clustering*.

   A sintaxe definida para esse parâmetro é: `path: <path>`, onde `<path>` é o endereço. Nas linhas seguintes, devem estar os nomes desses arquivos `.ds` e `.p`, um em cada linha. É importante observar que na última linha do arquivo `.job` deve estar a palavra chave `close`, indicando que não há mais arquivos a serem considerados.

   Para ilustrar, considere o exemplo apresentado na Tabela 18, na qual três experimentos são especificados:

   - `exp_single_link`;
   - `exp_complete_link`; e
   - `exp_average_link`.

   ```
   path:  /home/metzz/workspace/dhc/
   exp_single_link
   exp_complete_link
   exp_average_link
   close
   ```

   Tabela 18: Arquivo `.job` utilizado no modo `batch`.

   Desse modo, no diretório `/home/metzz/workspace/dhc/` devem estar os arquivos `.ds` e `.p` para cada um dos experimentos.

Uma ilustração do módulo de *clustering* do DISCOVER é apresentada na Figura 10. Nesse diagrama, a execução do *clustering* procede da seguinte maneira: inicialmente o conjunto de dados no formato DSX é carregado para a memória por meio dos procedimentos implementados na biblioteca DOL, e, então, submetido a um conversor para transformação dos dados para o formato reconhecido pelo aplicativo DH*Cluster* (`.dat` e `.mask`). Em seguida, esse novo conjunto de dados é carregado pelo aplicativo para a memória juntamente com os parâmetros de execução do aplicativo DH*Cluster* para realização do *clustering*. Esse processo resulta em dois arquivos que armazenam o dendograma segundo a estrutura *Pointer Representation* (arquivos `.result` e `.link`). Além disso, o aplicativo DH*Cluster* cria um arquivo `.matrix` que armazena a matriz de distâncias utilizada no processo de *clustering*, caso ela não tenha sido criada anteriormente. Após, o arquivo `.result` é submetido a um conversor que converte as informações armazenadas nesse arquivo para o

formato utilizado pelo SHELL*Cluster* e armazena essa informação no arquivo `.tree`. Nesse estágio, o dendograma já foi criado, basta então analisar o resultado obtido pelo aplicativo DH*Cluster*. Para isso, é feito um corte horizontal no dendograma por meio dos métodos implementados no módulo CLUSTER.PM, que cria um novo conjunto de dados supervisionado no qual o cluster é utilizado como atributo classe dos exemplos. Em seguida, esse novo conjunto de dados é submetido à um indutor para geração de regras que possam auxiliar no processo de interpretação dos resultados e atribuição de um significado útil aos clusters.

A descrição detalhada da estrutura de dados *Pointer Representation* pode ser encontrada em (Metz and Monard, 2006) e o formato da matriz de distâncias escrita no arquivo `.matrix` é apresentado neste trabalho na Seção 4.

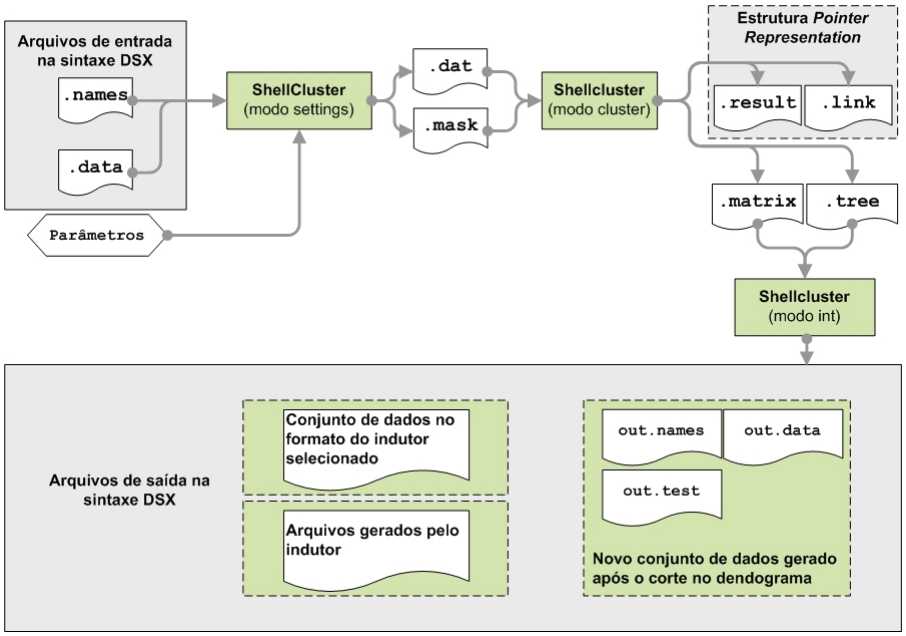

Figura 10: Módulo de *clustering* hierárquico do DISCOVER.

6 Considerações finais

A quantidade de dados armazenados em forma digital aumenta diariamente em uma rapidez assustadora. Muitas das organizações que produzem esses dados em suas tarefas cotidianas percebem o valor que eles representam e investem no desenvolvimento de pesquisas e na elaboração de estratégias que permitam tirar algum proveito desse grande acúmulo de dados. Para isso, de alguma maneira, há necessidade de transformá-los em informações que possam gerar conhecimento útil aos processos de tomada de decisão da organização, conseqüentemente, obter alguma vantagem nesse mundo tão competitivo.

Desse modo, muitas ferramentas computacionais são desenvolvidas com objetivo de facilitar a análise desses dados. Entretanto, em virtude do grande volume de dados armazenados eletronicamente, as técnicas e ferramentas tradicionais para análise, tais como planilhas eletrônicas, consultas às bases de

dados e geração de gráficos, não são capazes de extrair adequadamente o conhecimento embutido nos dados. Surge, então, a necessidade de técnicas mais eficientes para manipulação desses volumes de dados e extração de informações que possam ser úteis ao processo de tomada de decisão. Devido a essa e outras necessidades, diversos métodos têm sido desenvolvidos com objetivo de analisar grandes conjuntos de dados para descobrir padrões ou conhecimento de maneira automática ou semi-automática. Tais métodos têm contribuído para o surgimento de sistemas computacionais capazes de adquirir novos conhecimentos, novas habilidades e novas maneiras de organizar o conhecimento existente (Alpaydin, 2004). Esses sistemas são produto de uma sub-área da Inteligência Artificial (IA), aprendizado de máquina, cujos algoritmos estão sendo amplamente utilizados em aplicações de MD.

Nesse contexto, diversas ferramentas e ambientes de *software* foram propostos. Neste trabalho é apresentado o projeto de um módulo de AM não-supervisionado, mais especificamente para a realização do *clustering* hierárquico, integrado ao ambiente DISCOVER, o qual permite a realização de diversas tarefas necessárias ao processo de extração de conhecimentos em aplicações de MD.

Com a implementação do módulo de *clustering* hierárquico descrito neste trabalho e sua integração ao DISCOVER, é possível a realização de experimentos em um ambiente único e padronizado que dispõe de ferramentas para utilizar técnicas de pré-processamento de dados e pós-processamento de conhecimento, assim como a integração com outras técnicas de AM. Além disso, a elaboração de um módulo para esse ambiente, considerando os padrões de desenvolvimento adotados, permite que outros algoritmos sejam integrados de maneira bastante simples, desde que conversores sejam implementados para que as entradas e saídas desses algoritmos também atendam a esses padrões. Com isso, tem-se um sistema de fácil utilização, grande aplicabilidade e com muitas facilidades de expansão.

Referências Bibliográficas

Alpaydin, E. (2004). *Introduction to Machine Learning*. MIT Press. 39

Baranauskas, J. A. and Batista, G. E. A. P. A. (2000). O projeto DISCOVER: idéias iniciais. Comunicação pessoal. 2, 24

Batista, G. E. A. P. A. and Monard, M. C. (2003). Descrição da arquitetura e do projeto do ambiente computacional DISCOVER LEARNING ENVIRONMENT — DLE. Technical Report 187, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_187.pdf. 24, 26, 27, 28

Batista, G. E. A. P. A. and Monard, M. C. (2006). The DISCOVER OBJECT LIBRARY — DOL user's manual. Technical report, ICMC-USP. (em preparação). 25

Clark, P. and Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In *Proc. 5th European Conf. on Machine Learning*, volume 482 of *LNAI*, pages 151–163. Springer-Verlag. 25

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3(4):261–283. 25

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38. Series B. 12

Demsar, J. and Zupan, B. (2004). Orange: From experimental machine learning to interactive data mining. White paper, http://www.ailab.si/orange/doc. 1

Everitt, B. S. (1993). *Cluster Analysis*. Edward Arnold, 3 edition. 5, 11

Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2002a). Cluster validity methods: Part I. *SIGMOD Rec.*, 31(2):40–45. 6

Halkidi, M., Batistakis, Y., and Vazirgiannis, M. (2002b). Cluster validity methods: Part II. *SIGMOD Rec.*, 31(3):19–27. 6

Hoon, M. J. L., Imoto, S., and Miyano, S. (2005). *The C Clustering Library*. The University of Tokyo, Institute of Medical Science, Human Genome Center, Tokyo. http://bonsai.ims.u-tokyo.ac.jp/~mdehoon/software/cluster. 6

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice Hall. 5, 11

Karypis, G. (2003). *CLUTO: a clustering toolkit*. University of Minnesota, Department of Computer Science, Minneapolis. http://www-users.cs.umn.edu/~karypis/cluto/download.html. 2

Kohavi, R., Sommerfield, D., and Dougherty, J. (1996). Data mining using MLC++: A machine learning library in C++. *International Journal on Artificial Intelligence Tools*, 4(6):234–245. 1

Liu, H. and Motoda, H. (1998). *Feature Selection for Knowledge Discovery and Data Mining*. Kluver Academic Press, Boston. 5

Lutz, M. (2001). *Programming in Python*. O'Reilly & Associates, 2 edition. 23

Metz, J. and Monard, M. C. (2006). Estudo e análise das diversas representações e estruturas de dados utilizadas nos algoritmos de clustering hierárquico. Technical Report 269, ICMC-USP. http://www.icmc.usp.br/~biblio/download/RT_269.pdf. 2, 3, 5, 15, 16, 19, 23, 34, 38

PERL (1999). *Programming in PERL*. Morgan Kaufman. 23, 30

Prati, R. C. (2003). O framework de integração do sistema DISCOVER. Master's thesis, Instituto de Matemática e Computação, Universidade de São Paulo, Brasil. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-20082003-152116/. 24

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106. 25

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufman. San Francisco, CA. 25

Rathjens, D. (2000). *MineSet™ 3.0 Enterprise Edition*. Silicon Graphics. http://www.sgi.com/software/mineset/mineset_data.html. 1

Sibson, R. (1973). SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 1(16):30–34. 15

Team, R. D. C. (2005). *R: a language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.r-project.org. 2

Welch, B., Jones, K., and Hobbs, J. (2003). *Practical Programming in Tcl and Tk*. Prentice Hall, 4 edition. 23

Witten, I. H. and Frank, E. (2000). *Data mining: Pratical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufman. 1, 12, 13