Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Estudo e Comparação de Métodos para Estimação de Números de Grupos em Problemas de Agrupamento de Dados

André Fontana
Murilo Coelho Naldi

Nº 340

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Março/2009

Estudo e Comparação de Métodos para Estimação de Números de Grupos em Problemas de Agrupamento de Dados

André Fontana
Murilo Coelho Naldi

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-mails: fon@grad.icmc.usp.br, murilocn@icmc.usp.br

Resumo

O volume de dados gerado em certos cenários é cada dia maior e a capacidade de armazenamento destes dados ultrapassa a habilidade humana de análise e extração de conhecimento dos mesmos. Mineração de Dados é a área que pesquisa a extração de informações e conhecimento em bases de dados. Um dos problemas centrais desta área é o agrupamento não-supervisionado de dados que tem como objetivo particionar os dados em grupos de objetos (instâncias ou amostras) mais similares entre si do que com relação aos objetos dos demais grupos, segundo alguma medida de similaridade ou dissimilaridade.

Um problema ainda hoje crítico no que tange à tarefa de agrupamento diz respeito à estimação do número mais natural de grupos contidos em um determinado conjunto de dados. A maioria dos algoritmos requerem que o número de grupos seja fornecido pelo usuário. Existem, em contrapartida, diferentes metodologias sistemáticas para tentar estimar o número mais adequado de grupos a partir da supervisão e análise do comportamento de critérios de desempenho de múltiplas execuções desses algoritmos a partir de diferentes números de grupos. Não são claras, no entanto, as reais diferenças existentes entre essas diferentes metodologias em termos de eficácia e eficiência computacional. Esse projeto tem como objetivo investigar algumas das técnicas de múltiplas execuções de algoritmos de agrupamento e suas diferenças em termos de complexidade e desempenho.

Palavras-Chave: mineração de dados, agrupamento, critérios de validação, estimação de número de grupos, desempenho.

Impresso em 11 de março de 2009

Sumário

Lista de figuras iii

Lista de tabelas iv

1 Introdução 1

2 Introdução a Mineração de Dados 1
2.1 Necessidade e motivação . . . 1
2.2 Definição e aspectos principais . . . 2
2.3 Supervisionada / não supervisionada . . . 2
2.4 Classificação . . . 3
2.5 Agrupamento . . . 3
2.6 Classificação x Agrupamento . . . 4
2.7 Bases de dados . . . 4
2.7.1 Tipos de atributos . . . 4
2.7.2 Representação de bases de dados como tabelas de objetos por atributos . . . 5
2.7.3 Seleção de atributos relevantes para solucionar um determinado problema . . . 5
2.8 Noção de similaridade / dissimilaridade . . . 6
2.9 Algoritmos de Agrupamento . . . 7
2.10 Qualidade de uma partição . . . 9
2.10.1 Critério de razão das variâncias . . . 10
2.10.2 *Bayesian Information Criterion* (BIC) . . . 11
2.10.3 Critérios silhueta . . . 12
2.10.4 Critério Jaccard . . . 13
2.11 Problemas em bases de dados que afetam a qualidade de uma partição . . . 14
2.12 Tipos de algoritmos de agrupamento . . . 16
2.12.1 Algoritmos hierárquicos com métricas de integração . . . 17
2.12.2 *DIvisive ANAlysis* (DIANA) . . . 20
2.12.3 K-means (K-médias) . . . 21
2.12.4 *Bisecting k-means* . . . 23
2.12.5 *X-means* . . . 25

3 Comparação de metodologias de estimação de número de grupos 26
3.1 Bisecting K-means como estimador de número de grupos . . . 26
3.2 Implementações do Bisecting K-means . . . 27
3.3 Problemas encontrados no Bisecting K-means . . . 27
3.4 Diferenças entre implementações do X-means e seu artigo original . . . 29
3.4.1 Parâmetros a serem definidos pelo usuário . . . 29
3.4.2 Diferenças entre artigo e implementação em ansi C . . . 30
3.4.3 Diferenças entre artigo e implementação em Java . . . 31
3.4.4 Diferenças entre implementação em Java e implementação em ansi C 32
3.5 Implementações originais do X-means . . . 33
3.5.1 Características comuns de ambas versões . . . 33
3.5.2 X-means com BIC . . . 35
3.5.3 X-means com Silhueta . . . 36
3.6 Bases de dados . . . 36

3.7 Metodologia experimental 37
3.8 Resultados 38
3.9 Conclusões 46

Referências **48**

Lista de Figuras

1 Transição de dados físicos para dados digitais 2
2 Dados Representados em uma tabela de objetos / atributos 6
3 Relação de distância com dissimilaridade e similaridade 6
4 Representação gráfica das Distâncias Manhatan e Euclidiana 8
5 Relação entre similaridade de objetos e agrupamento de objetos 8
6 Funcionamento de um Algoritmo de Agrupamento 9
7 Esboço da função de desempenho 10
8 Mínimo local com qualidade discutível 10
9 Representação de um outlier em uma base de dados de duas dimensões . . 15
10 grupos com diferentes números de objetos em uma mesma base de dados . 16
11 Grupos com formatos e distribuições diferentes em uma mesma base de dados 16
12 Classificação de algoritmos de agrupamento 17
13 Exemplo de Dendrograma 18
14 Single-Link para identificação de shapes 18
15 Desvantagem do Single-link em bases de dados com ruídos 19
16 Contraste entre Complete-link e Average-link 20
17 Exemplo de Execução do algoritmo K-means 22
18 Exemplo de execução do Bisecting K-means 24
19 Avaliação de uma divisão do Bisecting K-means por meio do BIC 27
20 Divisão do Bisecting K-means: o grupo em destaque que possui sete grupos conhecidos será dividido utilizando K-means com parâmetro $k = 2$. 28
21 Divisão do Bisecting K-means: os dois novos grupos resultantes da divisão. Alguns dos objetos de dois grupos conhecidos foram separados de seus grupos originais e não serão unidos pelo resto da execução do algoritmo. . . 29
22 *Boxplots* relativos as médias do índice Jaccard para os algoritmos estudados. 39
23 *Boxplots* relativos as médias dos tempos de execução obtidos para os algoritmos estudados. 43

Lista de Tabelas

1 Comparações entre Agrupamento e Classificação. 4
2 Oito diferentes versões do Bisecting K-means. 28
3 Diferenças entras as médias dos valores de Jaccard obtidos pelos algoritmos estudados. 42
4 Diferenças entras as médias dos tempos de execução obtidos pelos algoritmos estudados. 45

1 Introdução

O volume de dados gerado em certos cenários é cada dia maior, por exemplo, em instituições financeiras, transações em empresas, dados científicos, dados textuais, etc. Porém, a capacidade de armazenamento destes dados ultrapassa a habilidade humana de análise e extração de conhecimento dos mesmos. Mineração de Dados (Data Mining) (Witten and Frank, 2005) é um campo de pesquisa que se destina à extração de informações e conhecimento em bases de dados.

O agrupamento não-supervisionado de dados (clustering) (Kaufman and Rousseeuw, 1990; Everitt et al., 2001) é um dos problemas centrais na área de data mining, cuja solução muitas vezes constitui o objetivo final da tarefa de mineração em si, mas em muitos casos pode também ser parte da solução de outros problemas relacionados, como classificação e extração de regras. O objetivo do agrupamento de dados é particionar os dados em grupos de objetos (instâncias ou amostras) mais similares entre si do que com relação aos objetos dos demais grupos, segundo alguma medida de similaridade ou dissimilaridade. A literatura sobre o problema de agrupamento de dados é extensa, sendo que por várias décadas diversos algoritmos de natureza, características e propósitos distintos têm sido propostos e investigados (Jain et al., 1999).

Um problema ainda hoje crítico no que tange à tarefa de agrupamento diz respeito à estimação do número mais natural de grupos contidos em um determinado conjunto de dados. A maioria dos algoritmos, em particular os mais populares, tradicionais e bem estabelecidos tanto no meio acadêmico quanto na prática, requerem que o número de grupos seja fornecido pelo usuário. Exemplos são os populares algoritmos das k-médias (k- means) (Witten and Frank, 2005). Existem, em contrapartida, diferentes metodologias sistemáticas para tentar estimar o número mais adequado de grupos a partir da supervisão e análise do comportamento de critérios de desempenho de múltiplas execuções desses algoritmos a partir de diferentes números de grupos, tais como, por exemplo, a avaliação de critérios relativos de validade (Jain and Dubes, 1988). Não são claras, no entanto, as reais diferenças existentes entre essas diferentes metodologias em termos de eficácia e eficiência computacional, dentre outros fatores, tanto no âmbito teórico como experimental. Esse trabalho tem como objetivo investigar algumas das técnicas de múltiplas execuções de algoritmos de agrupamento e suas diferenças em termos de qualidade, complexidade e desempenho.

2 Introdução a Mineração de Dados

2.1 Necessidade e motivação

Antes de introduzimos os conceitos do que vem a ser Mineração de Dados (Witten and Frank, 2005), vamos apresentar os panoramas do mundo atual que ajudam a entender quais foram as necessidades que motivaram os primeiros estudos dessa ciência.

Nossa realidade é marcada por uma utilização cada vez mais freqüente de bases de dados por diversos setores de nossa sociedade e para diversas aplicações. Empresas que possuíam grandes quantidades de dados físicos, armazenados em "gavetas e armários", migraram para o mundo da tecnologia, investindo em um armazenamento digital desses dados. O motivo principal é a busca por uma maior organização desses dados, que conseqüentemente proporciona um maior controle da empresa aos seus administradores.

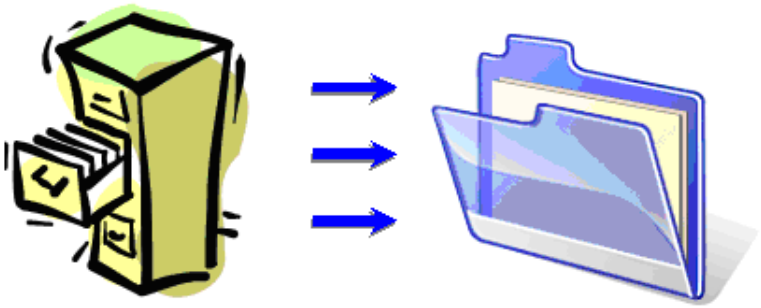


Figura 1: Transição de dados físicos para dados digitais

A busca de como otimizar a organização de dados incentivou pesquisas, que deram origem a tecnologias capazes de solucionar com considerável eficiência esse problema, como a Data Warehouse*.

Por outro lado, uma vez solucionada a questão da organização dos dados, surgiu uma nova necessidade: pesquisas por tecnologias capazes de transformar simples dados em informações e conhecimento, ferramentas poderosas para o desenvolvimento de qualquer empresa, por exemplo. Os resultados dessas pesquisas deram então origem a ferramentas de exploração hoje conhecidas como Mineração de Dados.

2.2 Definição e aspectos principais

Poderíamos interpretar o termo mineração como o ato de obter itens preciosos (de importante valor) através de técnicas de extração e refinamento dos recursos úteis de uma fonte de recursos rica, porém bruta. Associando esse significado ao panorama de uma base de dados, teríamos a fonte de recursos como sendo a própria base de dados, os itens preciosos como sendo as informações que estão escondidas por trás desses dados e as técnicas de extração como sendo, por sua vez, as aplicações práticas das disciplinas a que essa ciência está relacionada.

Ainda poderíamos, de forma mais formal, dizer que Mineração de Dados é uma área de pesquisa interdisciplinar, que envolve conhecimentos de Banco de Dados, Estatística e Inteligência Artificial, dentre outros, e que tem como objetivo aplicar os conhecimentos dessas áreas em ferramentas que sejam capazes de obter preciosas informações a partir de bases de dados.

2.3 Supervisionada / não supervisionada

As técnicas de Mineração de Dados podem ser divididas em supervisionadas ou não-supervisionadas. As características que as diferem serão discutidas a seguir:

- A Mineração de Dados supervisionada possui uma fase de treinamento que orienta os seus próximos passos através de informações obtidas previamente de alguma forma e possui um objetivo bem definido. Uma de suas técnicas é conhecida como classificação.

*Warehouse: tecnologia que permite armazenar as informações, anteriormente dispersas, através da identificação, compreensão integração e agregação dos dados

- A Mineração de Dados não-supervisionada não possui qualquer orientação prévia ou objetivo definido, em geral ela trabalha através de heurísticas que auxiliam na descoberta de informações na base de dados, que sejam capazes de responder perguntas relevantes que nem foram formuladas ou apresentadas previamente. Uma de suas técnicas é conhecida como agrupamento.

2.4 Classificação

Classificação é uma técnica de Mineração de Dados supervisionada, que tem como objetivo classificar objetos entre um conjunto previamente estipulado de classes. Objetos rotulados em uma mesma classe devem ter características em comum entre si.

Inicialmente um conjunto de dados que já teve seus objetos classificados (rotulados) previamente é apresentado a um algoritmo de classificação em uma fase de treinamento. Esse algoritmo irá então aprender, com essa base, quais as características que cada grupo de objetos de uma mesma classe tem em comum e quais são as características que grupos de objetos de classes diferentes não tem em comum. A partir dessas informações o algoritmo irá construir uma política de discriminação capaz de classificar novos objetos entre uma das classes estipuladas através de suas características.

Um exemplo de aplicação de classificação é um sistema de validação de compras por cartão de crédito. O algoritmo de classificação é previamente treinado por um grupo de objetos classificados entre bons pagadores (clientes que honraram seus pagamentos) e maus pagadores (clientes que não honraram seus pagamentos). A partir disso o algoritmo gera um modelo do que vem a ser os padrões de cliente bons pagadores e de clientes maus pagadores, com esse modelo, quando o algoritmo for apresentado a um novo cliente com características definidas mas que não possui uma classificação como bom pagador ou mau pagador, ele deverá ser capaz de classificar o novo cliente como bom ou mau pagador, com boa margem de precisão, afim de decidir se uma loja deve aceitar ou não que este cliente efetue um pagamento com cartão de crédito.

2.5 Agrupamento

Agrupamento ou *Clustering* é uma técnica de mineração de dados não supervisionada que tem como objetivo descobrir padrões em objetos de dados, gerando uma partição da base de dados que contenha grupos (clusters) de objetos com características comuns entre si. É importante ressaltar que esta técnica por si só não determina o significado dos grupos formados, tarefa esta que cabe a um especialista de domínio. Para o problema de categorização de espécies de flores, um exemplo de especialista de domínio seria um botânico.

Por ser uma aplicação não supervisionada, agrupamento pode não possuir qualquer informação prévia da base de dados da qual deve gerar partições, possivelmente nem mesmo o número de agrupamentos existente na base. Para realizar a partição de uma base de dados, um algoritmo de agrupamentos conta com heurísticas que auxiliam o particionamento dos objetos. Essas heurísticas geralmente têm relação com o conceito de similaridade entre os objetos, que será estudado no próximo capitulo.

Um exemplo de aplicação de agrupamento é a sua capacidade de formar grupos de pacientes com diagnósticos similares, baseado na análise de seus sintomas. Um outro exemplo é a segmentação de mercado em grupos de consumidores com perfis de compra similares.

2.6 Classificação x Agrupamento

A Tabela 1 mostra as principais diferenças entre as técnicas de Mineração de Dados classificação e agrupamento.

Classificação	**Agrupamento**
Número de Categorias Definido	Número de Categorias Não Definido
Supervisionado	Não Supervisionado
Categorias previamente definidas	Categorias não são previamente definidas
Classificar um novo objeto, entre as categorias já definidas.	Agrupar objetos conforme as suas similaridades: objetos de mesmo grupo têm alta similaridade, enquanto objetos de grupos diferentes têm baixa similaridade.

Tabela 1: Comparações entre Agrupamento e Classificação.

Algoritmos de agrupamentos também pode ser empregado como o primeiro passo para se fazer uma classificação.

2.7 Bases de dados

Introduziremos agora os conceitos mais fundamentais para se trabalhar com agrupamento: a definição do que são objetos e atributos de uma base de dados.

De maneira bem simples, podemos dizer que objetos são as instâncias de uma base de dados e que atributos são os dados que caracterizam esses objetos. É importante ressaltar que todos os objetos possuem valores independentes para cada um dos atributos presente na base de dados.

Dessa forma, se tivéssemos uma base de dados com informações sobre carros, por exemplo, diríamos que cada um dos carros representados é um objeto e que cada característica (cor, peso, ano, modelo, velocidade máxima), que possui valores para cada carro, é um atributo.

2.7.1 Tipos de atributos

- Atributos numéricos: São atributos que possuem valores numéricos. Eles ainda podem ser classificados como contínuos ou discretos.
  - Continuo: Um atributo numérico contínuo pode possuir qualquer valor numérico real, podendo portanto assumir valores fracionários ou mesmo negativos, dependendo da natureza da grandeza que está representando. Alguns exemplos desse tipo de atributo são: medidas físicas (peso, temperatura, altura, velocidade,...), o saldo de uma conta no banco, entre outros.
  - Discreto: Um atributo numérico discreto pode possuir apenas uma quantidade finita ou infinita mas contável de valores numéricos. Alguns exemplos desse tipo de atributo são: número de alunos em uma sala, idade de uma pessoa, número de rodas de um automóvel, número de grãos de areia em uma praia, entre outros.
- Atributos Binários: São atributos que podem possuir dois valores. Esses valores usualmente tem sentido oposto. São utilizados para descrever características que

possuem comportamentos extremos assumindo um valor ou outro e nunca um caso intermediário. Alguns exemplos desse tipo de atributo: características que tem valor sim ou não (tem asas, possui conta universitária,...), nível de sinais digitais, entre outros. Se os valores forem numéricos (por exemplo 0 e 1), podem ser considerados como um caso particular de atributos numéricos discretos. Caso contrário podem ser considerados como caso particular de atributos categóricos.

- Atributos Categóricos: São atributos não numéricos que caracterizam objetos dentre um número determinado de categorias. Podem ser classificados ainda como ordinais ou nominais.

  - Ordinais: Atributos categóricos ordinais são atributos categóricos em que as suas categorias possuem relações de ordem entre si. Nesse sentido, é possível analisar os graus de "parentescos" entre duas categorias. Utilizando uma analogia das classes econômicas presentes em nossa sociedade, poderíamos dizer que membros da classe alta têm maior grau de parentesco com membros da classe média do que com os membros da classe baixa. Além da hierarquia de classes econômicas, outros exemplos de atributos categóricos ordinais são: níveis de satisfação de determinada tarefa (muito satisfatória, satisfatória, parcialmente satisfatória, insatisfatória), nível de escolaridade (ensino fundamental, ensino médio, ensino superior,...), entre outros.
  - Nominais: Atributos categóricos nominais são atributos cujos valores pertencem a um conjunto finito de categorias que não possuem relações de ordem entre si. Alguns exemplos desse tipo de atributo são: profissão (estudante, bombeiro, carpinteiro,...), tipos de materiais (vidro, plástico, papel,...), entre outros.

2.7.2 Representação de bases de dados como tabelas de objetos por atributos

Geralmente em aplicações de agrupamento é comum representamos uma base de dados por uma tabela de objetos por atributos, onde cada linha da tabela corresponde a um objeto e cada coluna a um atributo. Podemos, então, enxergar a função de agrupamento como a de preencher uma nova coluna da tabela que contenha valores de rótulos que indicam a quais grupos cada objeto pertence.

A figura 2 ilustra uma base de dados de modelos de carros representada por uma tabela, onde as linhas representam os carros (objetos) da base de dados, as quatro primeiras colunas indicam os atributos dos carros (modelo, cor, ano de fabricação, velocidade máxima) e a quinta coluna corresponde à coluna dos rótulos dos grupos aos quais os objetos serão particionados após a execução de um algoritmo de agrupamento;

2.7.3 Seleção de atributos relevantes para solucionar um determinado problema

Para realizar a tarefa de agrupamento é necessário avaliar a similaridade entre os objetos que serão agrupados. Essa similaridade depende diretamente dos atributos que estão sendo utilizados. Dessa forma, a qualidade de uma partição dos dados em grupo depende da relevância dos atributos selecionados para gerá-la. Logo, um trabalho de agrupamento com qualidade deve conter uma seleção de atributos relevantes à aplicação desejada. Porém, essa seleção não é trivial, uma vez que agrupamento é uma técnica de Mineração de Dados não supervisionada e que, portanto, não pressupõe conhecimentos prévios da base de dados a ser particionada.

Objetos \ Atributos	Modelo(Cat. Nominal)	Cor(Cat. Nominal)	Ano(Num. Discreto)	Velocidade Máx(Num. Contínuo)	Rótulos
	Ferrari	Vermelho	2005	360Km/h	
	Gol	Azul	1998	180Km/h	
	Porshe	Amarelo	2000	320Km/h	
	A4	Vermelho	2008	200Km/h	
	. .	. .	. .	. .	. .

Figura 2: Dados Representados em uma tabela de objetos / atributos

Um exemplo simples de seleção de atributos é a exclusão de um atributo que tenha valores similares para todos os objetos, o que obviamente não contribui em nada para o processo de particionamento e que, ainda, minimiza o desempenho de uma medida de dissimilaridade entre os objetos.

Alguns trabalhos de pesquisa estudam mais a fundo técnicas de como realizar essa seleção. Tais técnicas não serão aqui discutidas por não ser esse o foco deste documento, mas o leitor interessado pode consultar, por exemplo, o trabalho de Devaney and Ram (1997).

2.8 Noção de similaridade / dissimilaridade

Para realizar a tarefa de particionamento é necessário utilizar algum critério de medida de similaridade entre objetos, ou seja, um critério que avalie o quanto dois objetos são parecidos e se devem, ou não, ser colocados em um mesmo grupo. Uma abordagem comum para comparação de dois objetos é a associação de uma função de distância (calculada através dos valores dos atributos de dois objetos) ao conceito de dissimilaridade. Nesse caso, dois objetos possuem alta dissimilaridade (baixa similaridade) se a distância entre eles for alta e possuem baixa dissimilaridade (alta similaridade) se a distância entre eles for baixa. Essa relação está ilustrada na Figura 3.

Figura 3: Relação de distância com dissimilaridade e similaridade

Existem formas diferentes de se calcular a dissimilaridade entre dois objetos com diferentes tipos de atributos. O desafio aumenta quando são analisados mais de um tipo de atributo ao mesmo tempo. Por isto, este relatório abrange apenas dissimilaridades entre objetos com atributos numéricos e contínuos.

Existem várias métricas de distâncias entre objetos com atributos numéricos. Cada uma possui melhores resultados em determinadas aplicações. Por exemplo, a distância Euclidiana produz bons resultados avaliando objetos que pertencem a grupos com formatos aproximadamente circulares, mas não é tão eficiente trabalhando com objetos que pertencem a grupos com formatos elípticos muito alongados. A tabela 1.2 descreve diferentes critérios de calculo de distâncias entre objetos com atributos numéricos e as principais

aplicações de cada um deles. Sendo $D_{i,j}$ o valor da distância entre os objetos i e j, n é o número de atributos da base de dados, $x_{i,j}$ é o valor do atributo j para o objeto i e r é o índice de Minkowski, alguns exemplos de distância entre objetos com atributos numéricos são:

- Distâncias de Minkowski: as distâncias de Minkowski são derivadas da Equação (1), de acordo com um valor escolhido para r, com $1 \leq r < \infty$.

$$D_{i,j} = (\sum_{z=1}^{n} (\mathbf{x}_{iz} - \mathbf{x}_{jz})^r)^{1/r} \qquad (r \geq 1) \tag{1}$$

  - Distância Manhatan: A distância Manhatan tem esse nome pois se inspira no espaço geográfico das ruas de Manhatan, onde para se locomover de um prédio a outro deve-se caminhar pelas as ruas, que se encontram nos eixos horizontais ou verticais. Dessa forma a distância entre dois objetos se dá pela soma das distâncias entre os valores de cada atributo desses dois objetos, como na Equação 2.

$$D_{i,j} = \sum_{z=1}^{n} (\mathbf{x}_{iz} - \mathbf{x}_{jz}) \tag{2}$$

  - Distância Euclidiana: A distância Euclidiana mede o comprimento do segmento de reta entre os dois objetos como pontos no espaço, como na Equação 3.

$$D_{i,j} = (\sum_{z=1}^{n} (\mathbf{x}_{iz} - \mathbf{x}_{jz})^2)^{1/2} \tag{3}$$

- Distância do Cosseno: A distância do Cosseno mede indiretamente a proporção entre os valores de cada atributo de um objeto, dada pela Equação 4. Essa distância é muito utilizada em mineração de textos, pela sua capacidade de avaliar a proporção com que as palavras aparecem em um texto.

$$D_{i,j} = 1 - \frac{\sum_{z=1}^{n} \mathbf{x}_{iz} \cdot \mathbf{x}_{jz}}{\sqrt{\sum_{z=1}^{n} \mathbf{x}_{iz}^2} \cdot \sqrt{\sum_{z=1}^{n} \mathbf{x}_{jz}^2}} \tag{4}$$

A Figura 4 ilustra a diferença entre as distâncias Euclidiana e Manhatan. Temos dois objetos representados em vermelho em um gráfico de duas dimensões e em (a) a representação gráfica da Distância Manhatan; e em (b) a representação gráfica da Distância Euclidiana.

2.9 Algoritmos de Agrupamento

Gerar grupos de objetos é o objetivo central de um algoritmo de agrupamento. Intuitivamente, podemos definir um grupo de objetos como um conjunto de objetos que possuem características semelhantes entre si. Expandindo um pouco mais para abranger os conceitos de similaridade e dissimilaridade podemos dizer que objetos de mesmo

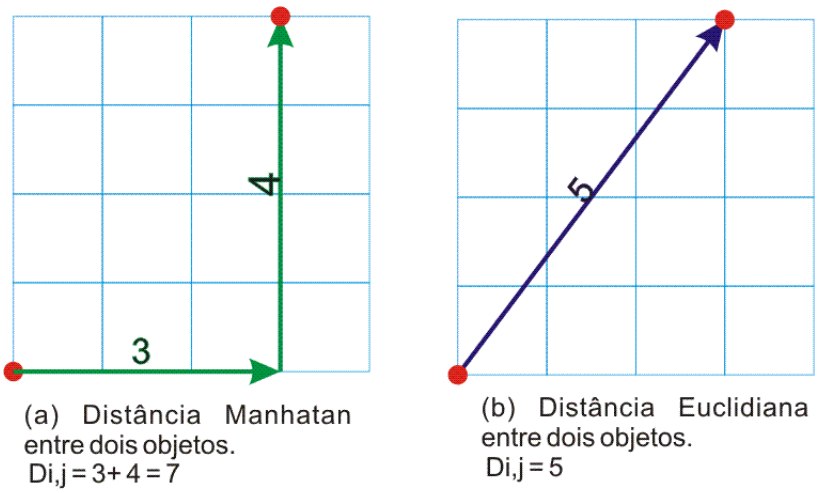


Figura 4: Representação gráfica das Distâncias Manhatan e Euclidiana

grupo possuem alta similaridade / baixa dissimilaridade e que objetos de grupos diferentes possuem baixa similaridade / alta dissimilaridade.

A Figura 5 ilustra a relação de similaridade entre objetos com os grupos de objetos a serem formados:

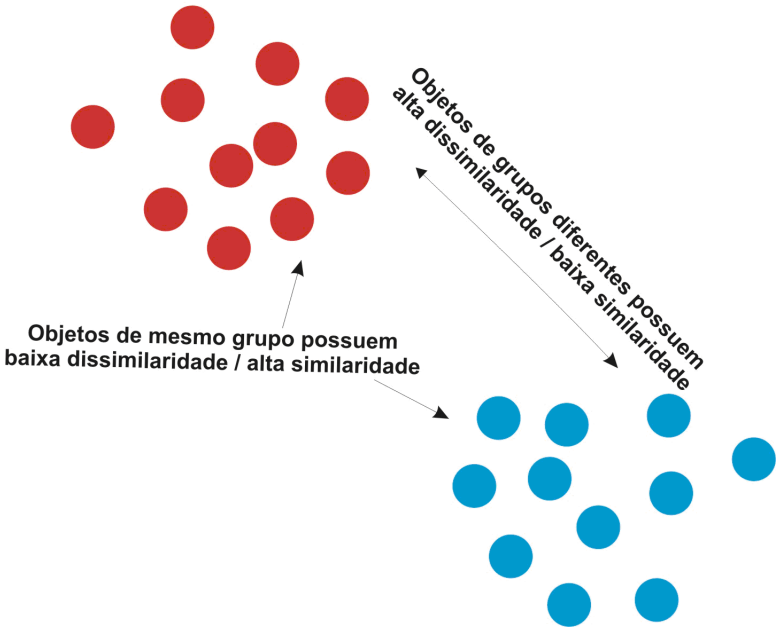


Figura 5: Relação entre similaridade de objetos e agrupamento de objetos

Algoritmos de agrupamento têm um objetivo em comum, gerar partições de uma base de dados através da construção de uma nova coluna na tabela objetos / atributos contendo o rótulo do grupo ao qual esse objeto pertence. Para realizar essa tarefa os algoritmos utilizam diferentes heurísticas de particionamento que são auxiliadas pelo conceito de similaridade. A figura 6 ilustra o funcionamento de um algoritmo de agrupamento.

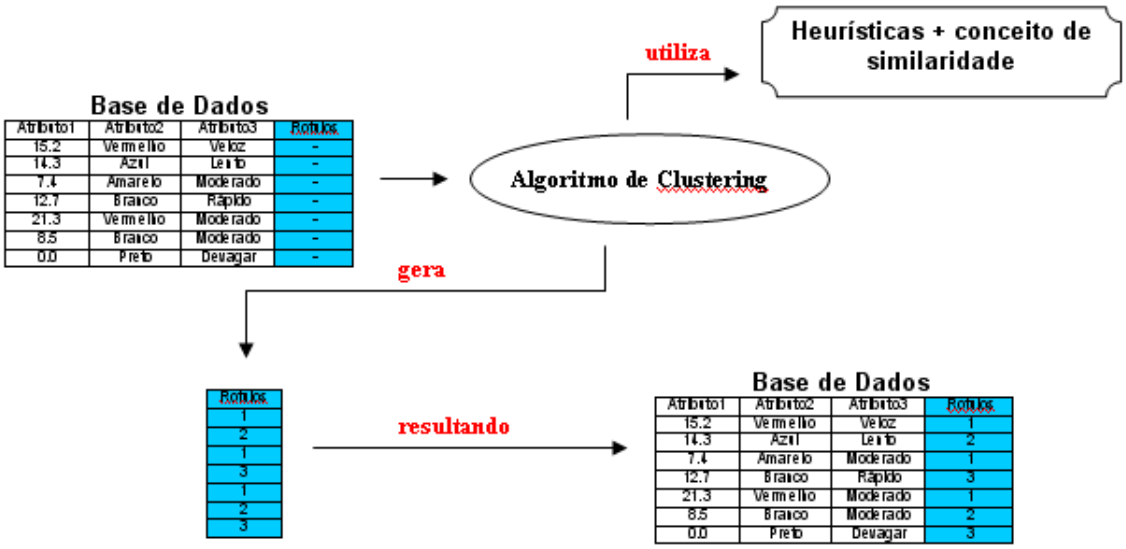


Figura 6: Funcionamento de um Algoritmo de Agrupamento

2.10 Qualidade de uma partição

Um dos grandes desafios de se trabalhar com agrupamento é o desconhecimento, presente na maioria dos casos, de qual a partição ideal a ser gerada para uma determinada base de dados. Sem essa informação não é trivial avaliar a qualidade de uma partição gerada.

Sobre esse aspecto existem funções de desempenho que avaliam a qualidade das partições, utilizando heurísticas que tem relação com o conceito de similaridade e dissimilaridade. Essas funções retornam valores numéricos de quão bom uma partição foi gerada. Interpretar esses valores em termos absolutos pode ser um novo desafio, uma vez que a variação deles muda conforme a base de dados que foi particionada. Uma base de dados pode ter partições que em geral recebem valores de qualidade muito próximos e cuja sensibilidade de variação deve ser mais bem estudada, enquanto outras podem obter resultados bem distantes e que não exigem um grande grau de sensibilidade para serem avaliados.

O processo de particionamento de uma base de dados pode ser visto como a otimização de uma função que retorna valores de desempenho conforme a qualidade de sua partição e cujos valores possuem mínimos e máximos locais e globais.

Sobre esse panorama enxergamos a qualidade das partições geradas por algoritmos iterativos com heurísticas de otimização como os pontos de mínimos locais dessa função e passamos a entender o desafio de gerar a melhor partição como sendo a tarefa de obter como retorno o valor de mínimo global dessa função (levando em consideração de que os valores retornados pela função de desempenho tenham uma relação de quanto menor, melhor).

A Figura 7 ilustra o esboço de uma função de desempenho que avalia a qualidade das partições.

Cada região em destaque é um vale que contém um ponto de mínimo local. Essas regiões correspondem aos pontos em que um algoritmo de otimização poderá ficar "preso" (ou convergir) e que, em muitos casos, apesar de serem pontos de mínimo, possuem qualidade discutível em relação ao mínimo global. Esse é o caso da região em destaque

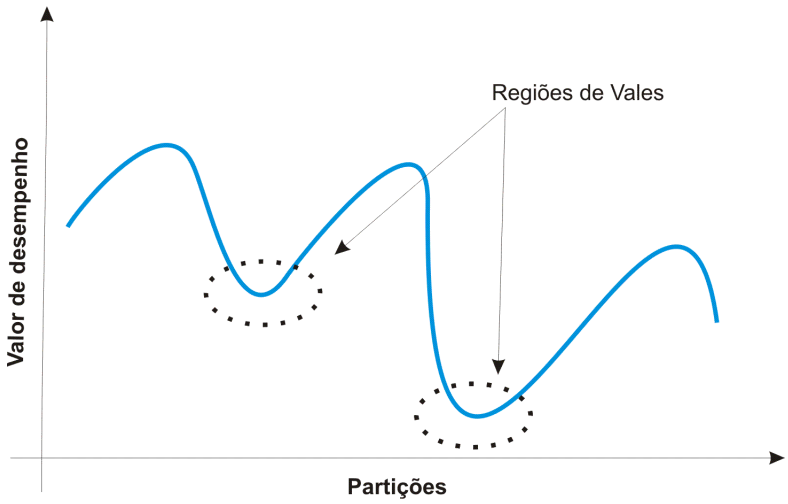


Figura 7: Esboço da função de desempenho

no gráfico de função de desempenho ilustrado pela Figura 8.

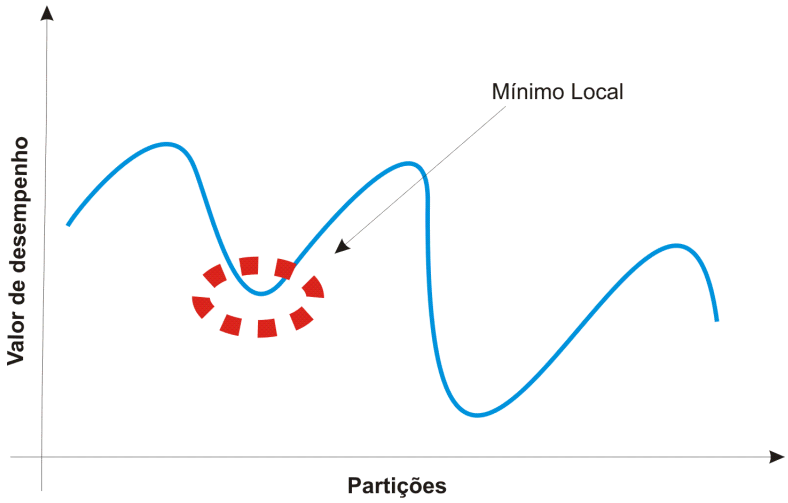


Figura 8: Mínimo local com qualidade discutível

Existem formas diferentes de fugir dos mínimos locais e buscar a convergência para o mínimo global. Uma delas é a múltipla execução de um algoritmo de otimização com valores iniciais gerados aleatoriamente, tendo em vista a heurística de que se o número de execuções tender para infinito a chance de pelo menos uma das inicializações ocorrer na região do mínimo global tenderá para 1(100%).

Alguns critérios de validade (qualidade) de agrupamentos serão apresentados a seguir, fortemente baseados na descrição feita em (Naldi, 2009).

2.10.1 Critério de razão das variâncias

O critério de razão das variâncias (*VRC*, do inglês *Variance Ratio Criterion*) é um índice que valoriza a homogeneidade interna e o isolamento externo dos grupos (Calinski and Harabasz, 1974). Esse índice é baseado em duas funções objetivo: a soma da distância quadrática interna aos grupos (*WGSS*, do inglês *within-group sum of squares*) e a soma da distância quadrática entre grupos (*BGSS*, do inglês *between-groups sum of squares*) (Duda et al., 2001). Dado um agrupamento, sua *WGSS* será a soma das distâncias quadráticas entre os objetos contidos em um mesmo grupo, calculada pela Equação (5), enquanto

sua $BGSS$ será a soma das distâncias quadráticas entre os objetos contidos em grupos distintos, calculada pela Equação (6):

$$WGSS = \sum_{i=1}^{k} d_{intra}(C_i) \tag{5}$$

$$BGSS = \sum_{i=1}^{k} d_{inter}(C_i), \tag{6}$$

em que k é o número de grupos da partição e $d_{intra}(C_i)$ e $d_{inter}(C_i)$ são as distâncias *intra*-grupo e *inter*-grupo relativas ao grupo C_i, dadas pelas Equações (7) e (8) respectivamente:

$$d_{intra}(C_i) = \sum_{y=1}^{|C_i|} \sum_{l=y+1}^{|C_i|} d(\mathbf{x}_{iy}, \mathbf{x}_{il}) \tag{7}$$

em que $|C_i|$ é a cardinalidade do conjunto C_i que representa o i-ésimo grupo, os padrões $\mathbf{x}_{iy}$ e $\mathbf{x}_{il}$ são o y-ésimo e o l-ésimo objetos contidos no grupo C_i.

$$d_{inter}(C_i) = \sum_{j \neq i} \sum_{y=1}^{|C_i|} \sum_{l=1}^{|C_j|} d(\mathbf{x}_{iy}, \mathbf{x}_{jl}) \tag{8}$$

em que $|C_j|$ é a cardinalidade do conjunto C_j, os padrões $\mathbf{x}_{iy}$ e $\mathbf{x}_{jl}$ são o y-ésimo e o l-ésimo objetos contidos nos grupos C_i e C_j, respectivamente, com $i \neq j$. É esperado que grupos compactos e separados possuam pequenos valores de $WGSS$ e grandes valores de $BGSS$. O índice VRC é dado pelo resultado da Equação (9), para n_o objetos do conjunto de dados:

$$VRC = \frac{BGSS}{WGSS} \times \frac{(n_o - k)}{(k - 1)} \tag{9}$$

Quanto melhor a partição, maior será o valor da razão entre $BGSS$ e $WGSS$. O termo de normalização $(n_o - k)(k - 1)$ previne o aumento monotônico do índice em relação ao número de grupos, o que faz do VRC um índice de otimização em relação a k. Isso significa que, se utilizado em um agrupamento com número não conhecido (não fixo) de grupos, o VRC pode ajudar a determinar este número para conjunto de dados. Também pode ser utilizado para comparar os resultados de diferentes algoritmos aplicados ao mesmo conjunto de dados.

2.10.2 Bayesian Information Criterion (BIC)

O *Bayesian Information Criterion* (BIC) (Mirkin, 2005), ou critério de (Schwarz, 1978) como também é conhecido, é um critério estatístico utilizado para seleção de modelos. Ele consiste em avaliar o balanceamento do ganho de informação de um modelo ao passo que o modelo ganha complexidade, por meio do aumento do número de seus parâmetros, ou seja, quanto mais informação o modelo tiver melhor será o critério. Por outro lado, quanto mais parâmetros o modelo tiver pior será o critério. O BIC é calculado por meio da Equação 10:

$$BIC = \ln(\hat{l}) - \frac{p}{2} \cdot \ln(N). \tag{10}$$

Alguns autores também apresentam o cálculo do BIC por meio da Equação 11, que equivale ao dobro da Equação 10:

$$BIC = 2 \cdot \ln(\hat{l}) - p \cdot \ln(N), \tag{11}$$

onde $\ln(\hat{l})$ é o logaritmo da estimativa da máxima verossimilhança, p é o número de parâmetros da partição, N é o número de objetos da partição. A forma como o logaritmo de máxima verossimilhança e o número de parâmetros são calculados para o problema de agrupamento variam na literatura.

2.10.3 Critérios silhueta

Os critérios do tipo silhueta definem a qualidade dos agrupamentos com base na proximidade entre os objetos de um determinado grupo e na distância desses objetos ao grupo mais próximo. Publicado por Rousseeuw (1987), o critério silhueta original é calculado para cada objeto de um grupo, mostrando quais objetos estão bem situados no mesmo e quais seriam situados melhor em outro grupo. O critério silhueta pode ser calculado com qualquer medida de dissimilaridade/similaridade.

Seja um objeto $\mathbf{x}_i$ e o grupo C_A tal que $\mathbf{x}_i \in C_A$. Seja $a(\mathbf{x}_i)$ a dissimilaridade média do objeto $\mathbf{x}_i$ em relação a todos os demais objetos do grupo C_A e $b(\mathbf{x}_i)$ a menor dissimilaridade média de $\mathbf{x}_i$ dos objetos de um outro grupo C_B, $C_B \neq C_A$. A silhueta de um objeto $\mathbf{x}_i$ é obtida pela Equação (12).

$$silhouette(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{max\{a(\mathbf{x}_i), b(\mathbf{x}_i)\}} \tag{12}$$

Se $\mathbf{x}_i$ for o único objeto do grupo a que pertence, então $silhouette(\mathbf{x}_i)$ se torna indefinido e uma escolha neutra seria assumir $silhouette(\mathbf{x}_i) = 0$ (Rousseeuw, 1987). O resultado obtido pela Equação (12) estará no intervalo [-1,1]. Se um objeto está bem situado em seu grupo, sua silhueta será positiva, caso contrário, se o objeto estiver mais próximo de outro grupo, ela será negativa. Logo, se o algoritmo utilizado garantir que os objetos estejam no grupo mais próximo, o resultado da Equação (12) estará no intervalo [0,1].

Como a silhueta depende apenas do agrupamento resultante e não do algoritmo de agrupamento empregado, ela pode ser usada para executar uma análise dos grupos obtidos, para comparar partições geradas por diferentes algoritmos, ou diferentes execuções do mesmo algoritmo, em um mesmo conjunto de dados. Para isso, é possível calcular a silhueta de um grupo ou partição (*ASWC*, do inglês *Average Silhouette Width Criterion*), por meio da média das silhuetas de seus objetos. A *ASWC* é dada pela Equação (13), para um grupo C_j a partir de seu i-ésimo objeto $\mathbf{x}_{ji}$, e pela Equação (14), para uma partição π.

$$ASWC(C_j) = \sum_{i=1}^{|C_j|} \frac{silhouette(\mathbf{x}_{ji})}{|C_j|} \tag{13}$$

$$ASWC(\pi) = \sum_{i=1}^{n_o} \frac{silhouette(\mathbf{x}_i)}{n_o} \tag{14}$$

Quanto maior o valor de $ASWC(\pi)$, melhor o posicionamento dos objetos dentro dos seus grupos, o que pode ser usado para determinar o melhor número k de grupos, por exemplo. Para isso, um algoritmo de agrupamento pode ser utilizado para gerar partições do conjunto de dados com diversos valores de k e comparar o $ASWC$ obtido entre as partições. Essa comparação pode ser feita por meio de um gráfico ($ASWC(\pi) \times k$), onde um pico indica o valor de k mais apropriado, provavelmente o maior valor para $ASWC(\pi)$.

Uma versão simplificada do critério silhueta é proposta por Hruschka et al. (2004a), onde há uma modificação nos cálculos de $a(\mathbf{x}_i)$ e $b(\mathbf{x}_i)$. Nesta versão, $a(\mathbf{x}_i)$ é a dissimilaridade entre o objeto $\mathbf{x}_i$ e o centróide $\mathbf{c}_A$ do grupo C_A, tal que $\mathbf{x}_i \in C_A$. Além disso, $b(\mathbf{x}_i)$ é a menor dissimilaridade entre $\mathbf{x}_i$ e o centróide $\mathbf{c}_B$ de um outro grupo C_B, $C_B \neq C_A$. Esta implementação visa reduzir o custo computacional da execução do critério de $O(n_o^2)$ para $O(n_o)$ e, segundo os autores, mantém uma qualidade próxima do critério original, o que é confirmado nos exaustivos experimentos de Vendramin et al. (2009). Hruschka et al. (2004b) mostram que, alternativamente à versão original e simplificada do critério silhueta, uma versão modificada pode ser utilizada, substituindo a silhueta dada pela Equação (12) pela silhueta dada pela Equação (15), em que ε é uma pequena constante necessária para o calculo de $silhouette(\mathbf{x}_i)$ quando $a(\mathbf{x}_i) = 0$.

$$silhouette(\mathbf{x}_i) = \frac{b(\mathbf{x}_i)}{a(\mathbf{x}_i) + \varepsilon} \tag{15}$$

A versão modificada aparenta ser mais sensível a pequenas mudanças em $a(\mathbf{x}_i)$ e $b(\mathbf{x}_i)$, que por sua vez corresponde a mudanças significativas no agrupamento. Porém, o critério modificado não mais retorna resultados no intervalo [-1,+1]. Além disso, uma versão híbrida pode ser facilmente obtida a partir da combinação das versões simplificada e modificada da silhueta.

O critério silhueta é mais apropriado para agrupamentos volumétricos com grupos compactos e separados. Vendramin et al. (2008, 2009) mostra que o critério silhueta obtém resultados comparáveis ou melhores do que o critério VRC, e significantemente melhores do os critérios DB, Dunn e variantes para partições com estas características. Porém, ela resulta em larguras tendenciosas contra grupos potencialmente sobrepostos, favorecendo agrupamentos disjuntos. Além disso, ela nem sempre obtém bons resultados para grupos com formatos arbitrários (Rousseeuw, 1987).

2.10.4 Critério Jaccard

Indices de validação externos objetivam mensurar quanto um agrupamento obtido se aproxima da estrutura conhecida dos dados. Para isso, os índices externos mais tradicionais comparam a partição resultante π_r com uma partição referência π_g ("gold standard "), construída com base na intuição ou conhecimento a priori sobre a estrutura real dos dados. Estes índices se baseiam em informações a respeito da relação entre os objetos de ambas as partições.

Não é possível determinar a equivalência entre grupos de partições distintas unicamente por meio de seus rótulos. Para estabelecer esta equivalência, é necessário considerar

a relação entre seus objetos. Por este motivo, os índices de validação externos apresentados aqui consideram a relação entre pares de objetos para determinar esta equivalência.

Dado o par de objetos $\mathbf{x}_i$ e $\mathbf{x}_j$ pertencente ao conjunto de dados X, tal que $i \neq j$, os grupos $C_r \in \pi_r$ e $C_g \in \pi_g$, consideremos as quatro situações apresentadas a seguir:

- 11: $(\mathbf{x}_i \in C_r)$ e $(\mathbf{x}_j \in C_r)$, $(\mathbf{x}_i \in C_l)$ e $(\mathbf{x}_j \in C_l)$
- 01: $(\mathbf{x}_i \in C_r)$ e $(\mathbf{x}_j \notin C_r)$, $(\mathbf{x}_i \in C_l)$ e $(\mathbf{x}_j \in C_l)$
- 10: $(\mathbf{x}_i \in C_r)$ e $(\mathbf{x}_j \in C_r)$, $(\mathbf{x}_i \in C_l)$ e $(\mathbf{x}_j \notin C_l)$
- 00: $(\mathbf{x}_i \in C_r)$ e $(\mathbf{x}_j \notin C_r)$, $(\mathbf{x}_i \notin C_l)$ e $(\mathbf{x}_j \in C_l)$

Seja np_t o número de todas as possíveis combinações de pares de objetos do conjunto de dados ($np_t = n_o(n_o - 1)/2$), np_{11} o número de pares de objetos na situação 11, np_{01} o número de pares de objetos na situação 01, np_{10} e np_{00} os números de pares de objetos nas situações 10 e 00, respectivamente. Algumas destas variáveis serão utilizadas nos índices a serem definidos nesta seção.

O índice Jaccard, também conhecido como coeficiente de similaridade de Jaccard (1908), consiste em um índice utilizado para comparar a similaridade entre conjuntos de amostras. Quando utilizado para calcular a probabilidade de dois objetos pertencerem ao mesmo grupo em ambas as partições, o índice Jaccard é dado pela Equação (16)

$$Jaccard(\pi_r, \pi_g) = \frac{np_{11}}{np_{11} + np_{01} + np_{10}} \tag{16}$$

O coeficiente de Jaccard resulta em valores entre [0,1], de forma que quanto mais diferentes as partições, mais próximo de 0 será o valor retornado e partições idênticas resultam no valor 1.

2.11 Problemas em bases de dados que afetam a qualidade de uma partição

Algumas Bases de Dados possuem certas características que dificultam o processo de particionamento. Discutiremos nesta seção algumas dessas características que serão citadas no decorrer deste trabalho.

- Ocorrência de Outliers: outliers são objetos que possuem alta dissimilaridade com todos os demais objetos da base e que, portanto, não se encaixam bem em nenhum grupo de objetos. A ocorrência de outliers é muito comum em bases de dados, por isso algoritmos de agrupamento devem tratar esses tipos de objetos, trabalhando com heurísticas que não se prendem a informação de um único objeto para realizar a formação de grupos. Assim, se houver outliers em uma base de dados, esses tenderão a ser ignorados devido à sua contribuição ser relativamente pequena em relação ao total de objetos da base. A Figura 9 ilustra um exemplo de ocorrência de outlier em uma base de dados de duas dimensões.
- Grupos com números de objetos muito diferentes em uma mesma base de dados: não existe uma regra para determinar quantos objetos devem pertencer a cada grupo. De fato, grupos podem ser formados com um único objeto (chamados de

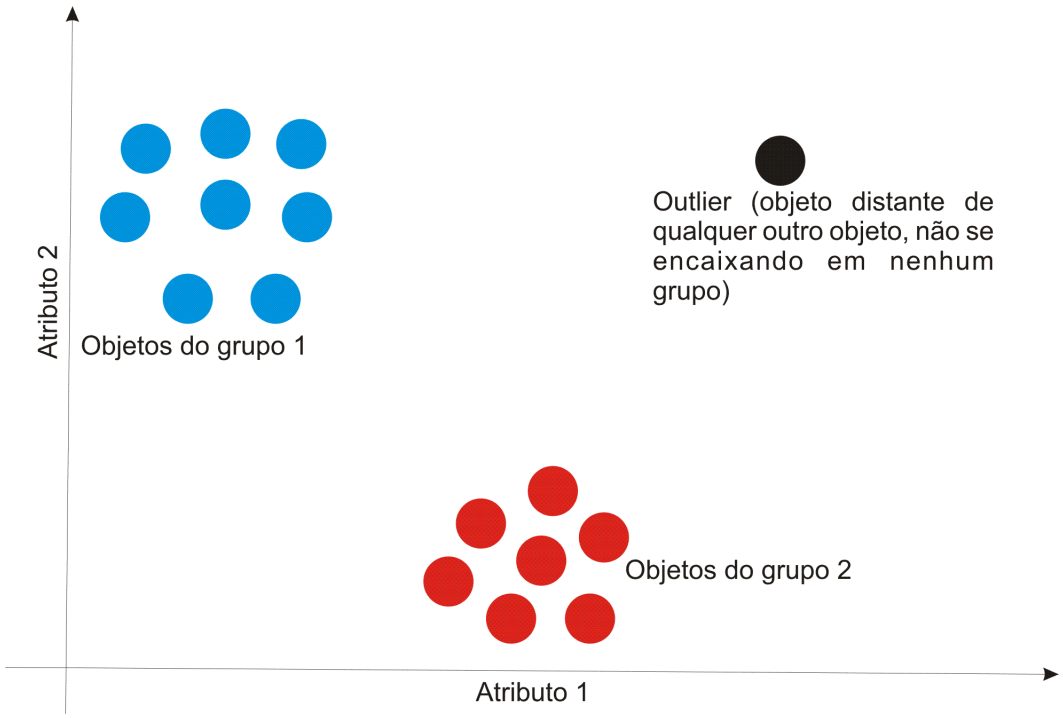


Figura 9: Representação de um outlier em uma base de dados de duas dimensões

singletons), com dezenas ou centenas e assim por diante. O problema é que se houver uma variação muito grande no número de objetos de cada grupo em uma base de dados, cria-se uma dificuldade para os algoritmos a particionarem com acurácia, pois, nesses casos, os grupos com baixa densidade (grupos com número pequeno de objetos) podem acabar sendo ignorados pelos algoritmos de particionamento. A Figura 10 ilustra um exemplo de base de dados que contém grupos com diferentes números de objetos.

- Bases de dados com valores de atributos de alguns objetos não determinados ou errados: é muito comum encontrar bases de dados que possuam um ou mais objetos com valores de atributos incompletos (denominados "valores ausente") ou incoerentes. Isso acontece devido à presença de ruídos nas bases de dados e ao fato que bases que possuem grande número de objetos quase sempre são esparsas.

  Bases de dados que apresentam objetos assim são claramente difíceis de serem particionadas de forma apropriada.

  Um exemplo comum de dados com ruído e valores ausentes advém de bases com dados medidos por sensores, que normalmente são afetados por ruídos e podem estar ausentes no caso do sensor ficar inoperante por algum período. Outro exemplo clássico de ruídos em dados ocorre em bases, cujos dados foram digitados por um usuário, devido aos eventuais erros de digitações.

- Grupos com diferentes características (distribuição e / ou formato) em uma mesma base de dados: grupos podem ainda apresentar formatos e distribuições distintas. Eles podem, por exemplo, ter formatos circulares e bem concentrados, ou elípticos com os objetos espalhados. O problema de agrupamento pode se tornar difícil quando encontramos grupos com formatos muito diferentes em uma mesma base de dados. Isso acontece por que, geralmente, os algoritmos trabalham com heurísticas que são capazes de identificar grupos de determinados formatos, mas que apresentam dificuldade quando encontram grupos de formatos variados. A figura 11 ilustra um

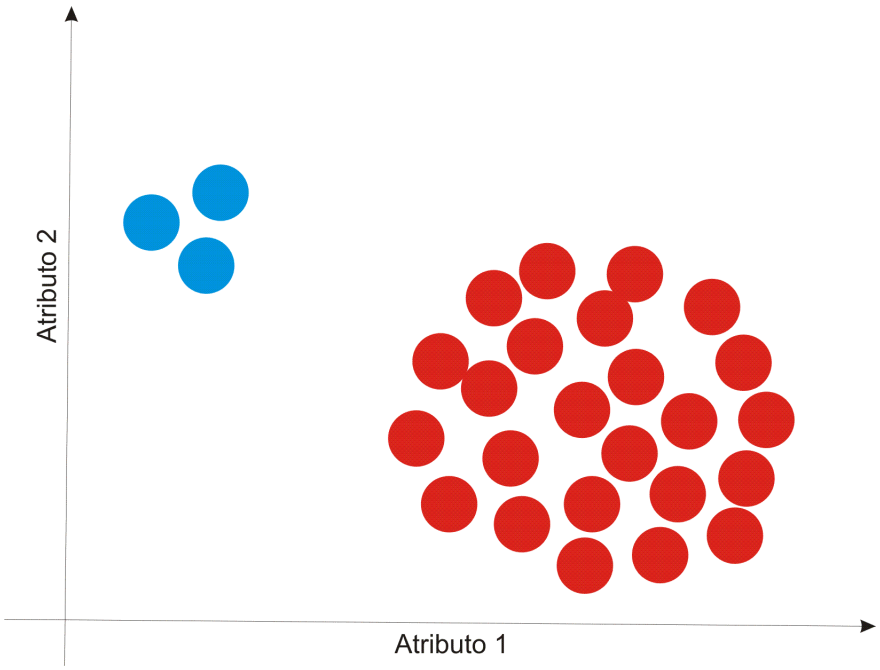


Figura 10: grupos com diferentes números de objetos em uma mesma base de dados

exemplo de base de dados que contém grupos com diferentes distribuições e formatos em uma mesma base de dados.

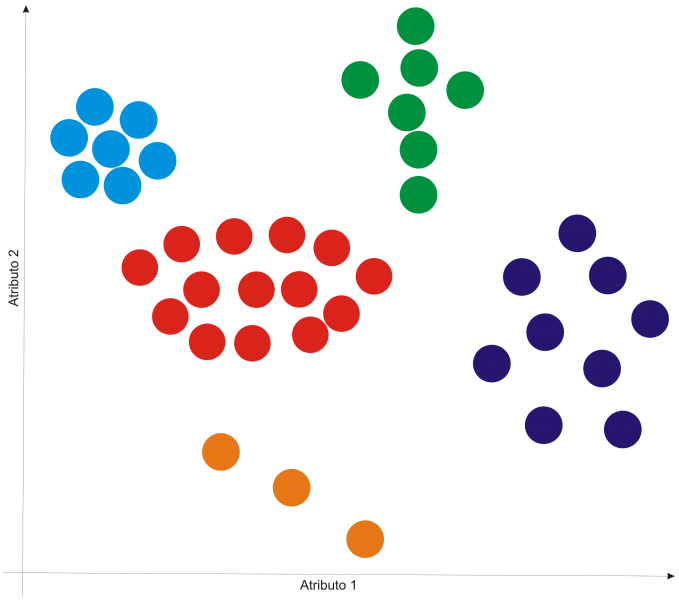


Figura 11: Grupos com formatos e distribuições diferentes em uma mesma base de dados

2.12 Tipos de algoritmos de agrupamento

Algoritmos de agrupamento são geralmente classificados como particionais, hierárquicos ou híbridos. A Figura 12 Ilustra essa classificação e algumas de suas sub-classificações:

Nesta seção, apresentaremos as definições do que são algoritmos de agrupamento hierárquicos e particionais, bem como alguns dos algoritmos mais populares de cada um desses tipos, descrevendo seu funcionamento, heurísticas relacionadas e tempo de execução.

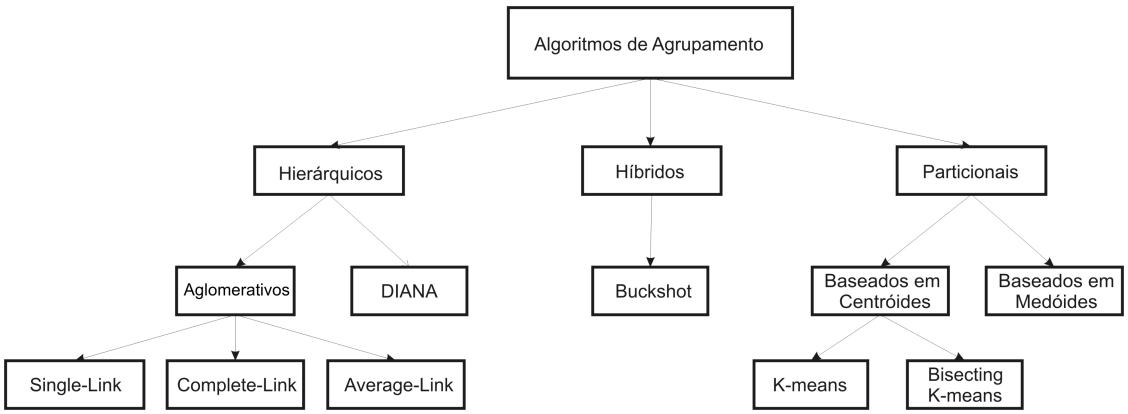


Figura 12: Classificação de algoritmos de agrupamento

2.12.1 Algoritmos hierárquicos com métricas de integração

Algoritmos hierárquicos produzem um conjunto de partições de dados através de aglomerações ou divisões de grupos. Algoritmos aglomerativos unem grupos iterativamente até que um único grupo contendo todos os objetos da base de dados seja formado, enquanto, algoritmos divisivos quebram iterativamente grupos, em dois, até que cada objeto forme o seu próprio grupo. A decomposição resultante, que é uma árvore de grupos, é chamada de dendrograma. O dendrograma fornece uma visualização de como um algoritmo gera uma saída. A sua análise permite verificar se a saída gerada é ruim ou não.

A Figura 13 (c) ilustra um dendrograma para uma base bidimensional ilustrada em 13 (a) Em 13 (c) a linha da esquerda representa uma régua que mede os valores de dissimilaridade entre os grupos que estão sendo aglomerados, enquanto a linha em vermelho indica o valor máximo de dissimilaridade entre os grupos selecionado pelo usuário/especialista para que a aglomeração entre os grupos seja realizada; Sendo assim a Figura 13 (b) ilustra os grupos formados pelo AGNES até a última aglomeração aconselhada pelo dendrograma. Neste exemplo a dissimilaridade é calculada segundo o critério Average-link.

Um exemplo típico de algoritmo aglomerativo pode ser dividido em dois passos:

1. Inicialização: O algoritmo inicia com N grupos (1 grupo para cada objeto);
2. Aglomeração: Os dois grupos mais similares são agrupados (aglomerados);

O passo 2 é repetido N-1 vezes de forma que na última iteração um grupo que contenha todos os objetos da base de dados seja gerado, resultando em complexidade $O(N^2 \cdot \log N)$. Existem três versões deste algoritmo com diferenças em relação as métricas de integração entre dois grupos utilizados, sendo eles: Single-link, Average-link e Complete-link.

O critério Single-link trata a similaridade entre dois grupos como sendo igual a similaridade entre os dois objetos mais similares daqueles grupos. Muito utilizado para identificar grupos em bases de dados que possuem grupos com características de shapes, como circunferências, retas, entre outras. A Figura 14 ilustra alguns grupos com características de shapes.

Um problema do critério Single-link consiste em seu baixo desempenho em bases de dados com ruídos. Devido ao critério de similaridade do algoritmo consistir em avaliar a distância entre os objetos mais semelhantes, uma possível "ponte de objetos" provocada

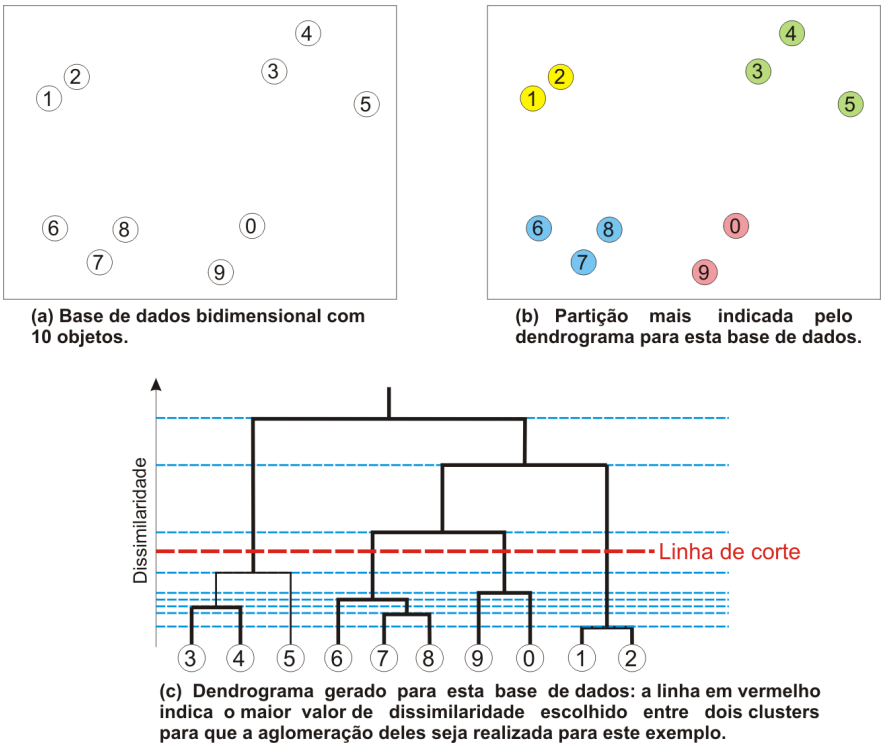


Figura 13: Exemplo de Dendrograma

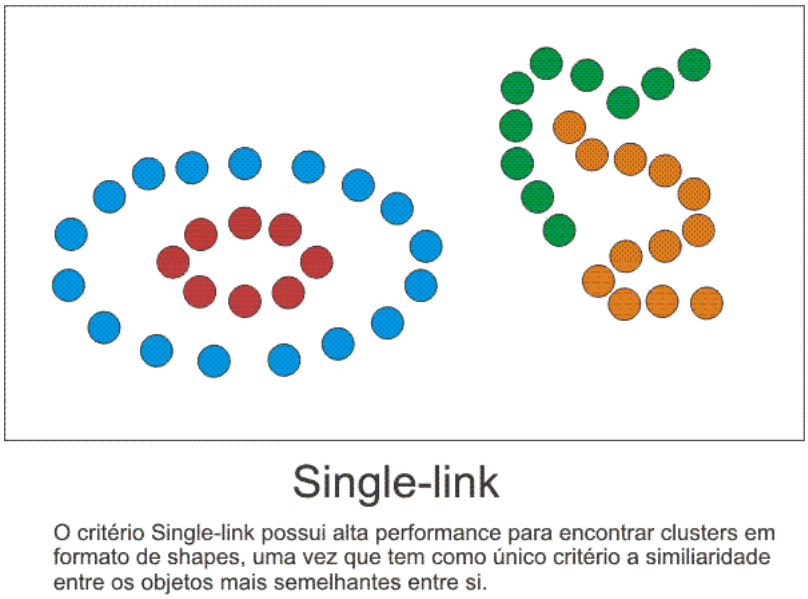


Figura 14: Single-Link para identificação de shapes

por ruídos da base de dados poderia unir dois grupos distintos como ilustra a Figura 15 inspirada em Pantel (2003). Esta desvantagem desaconselha o uso do critério de similaridade Single-link em bases de dados com ruídos.

Em 15 (a) uma base de dados com 3 grupos conhecidos e com ruídos; em 15(b) os grupos formados pelo algoritmo utilizando Single-link como critério de similaridade; Em 115 (c) os grupos formados pelo algoritmo utilizando Complete-link como critério de similaridade.

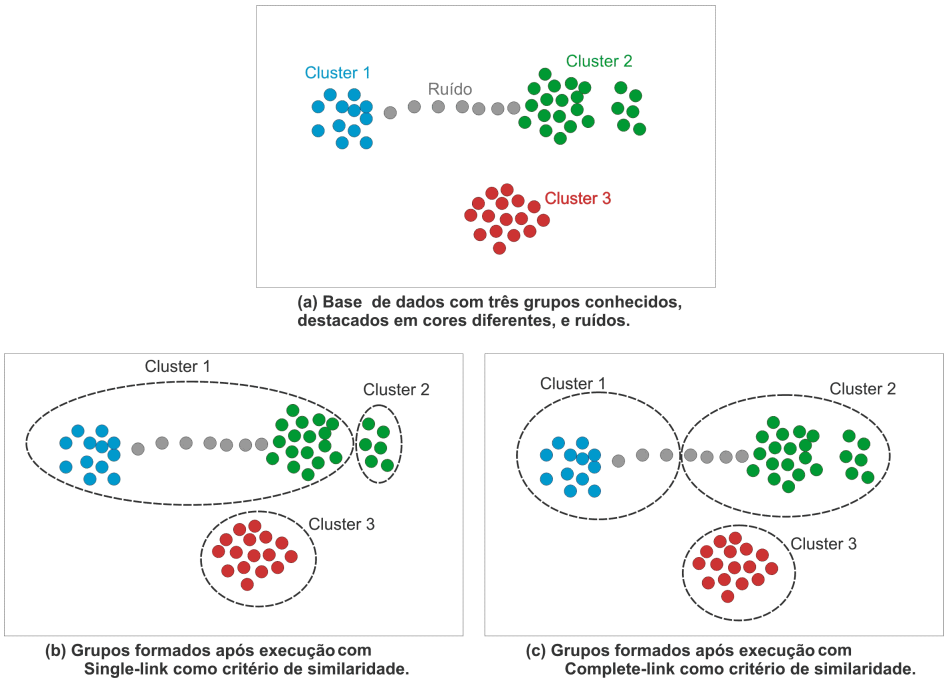


Figura 15: Desvantagem do Single-link em bases de dados com ruídos

A partir da Figura 15 fica claro que o critério Single-link não é a melhor escolhar para analisar similaridade entre grupos de bases de dados com ruídos, sendo nesse caso mais recomendado utilizar os critérios Clomplete-link ou Average-link. Agora, caso o objetivo seja localizar grupos com características de shapes, o critério Single-link é o mais indicado.

O critério Complete-link trata a similaridade entre dois grupos como sendo igual à similaridade entre os dois objetos menos similares daqueles grupos.

O critério Average-link trata a similaridade entre dois grupos como sendo igual à média das similaridades entre cada par de objetos, onde um par de objetos contém um objeto de cada grupo.

O Average-link retorna resultados parecidos com o complete-link, mas está menos suscetível a erros, pois elimina problemas gerados por casos onde existe um objeto de um grupo que não tem uma similaridade tão grande em relação aos demais objetos do grupo. A Figura 16 ilustra um exemplo em que temos dois grupos A e B com 10 e 7 objetos, respectivamente. Dentre os objetos de A existe um com baixa similaridade aos objetos de B, enquanto os outros 9 possuem alta similaridade aos objetos de B. Se aplicássemos o critério de complete-link, constataríamos que o grupo A tem baixa similaridade a B, mas se aplicássemos o critério de average-link constataríamos que o grupo A tem alta similaridade em relação ao grupo B, como já era esperado.

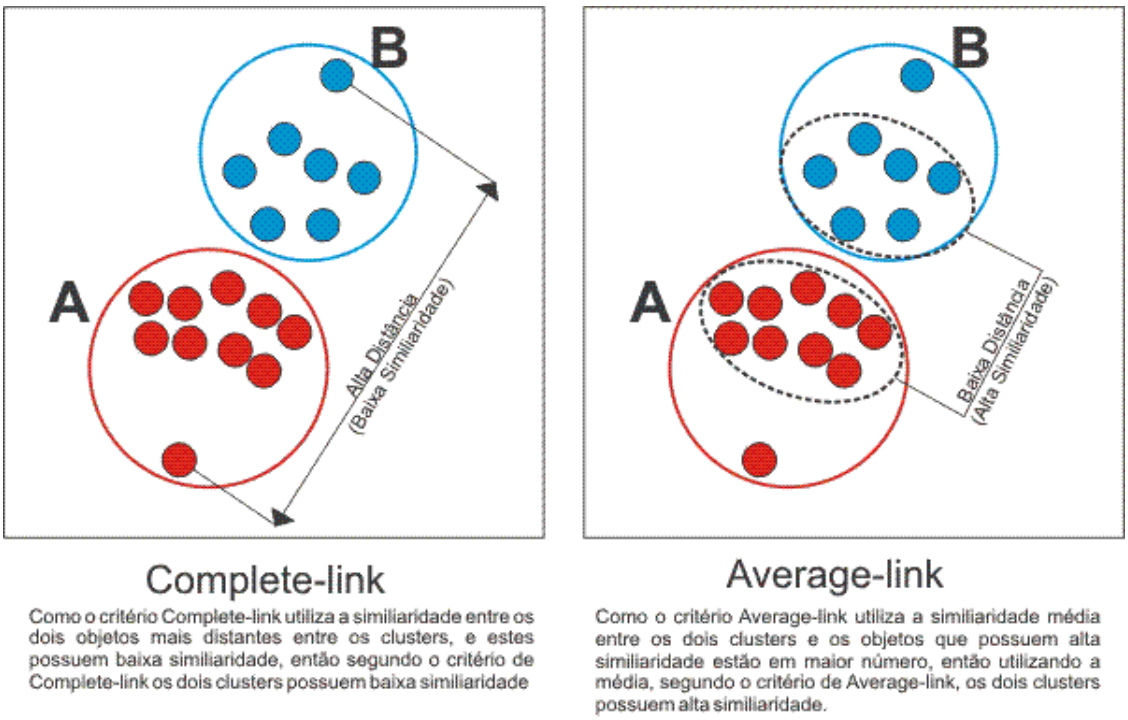


Figura 16: Contraste entre Complete-link e Average-link

2.12.2 DIvisive ANAlysis (DIANA)

Proposto por Kaufman and Rousseeuw (1990), o algoritmo DIANA é um exemplo comum de algoritmo divisivo hierárquico. Ele inicia-se com um único grupo contendo todos os objetos e utiliza a heurística de dividir o maior grupo iterativamente, o que resulta em uma complexidade $O(n_o^2 \log n_o)$. Considerando o diâmetro de um grupo como a maior distância entre dois de seus objetos, o algoritmo pode ser descrito pelos seguintes passos:

1. O algoritmo é inicializado com um único grupo que contenha todos os objetos do conjunto de dados.
2. O grupo com maior diâmetro C_l é selecionado.
3. Do grupo selecionado, escolhe-se o objeto $\mathbf{x}_e$ com a menor similaridade média em relação aos outros objetos.
4. O objeto $\mathbf{x}_e$ é inserido no grupo de divisão C_d.
5. Encontra-se o objeto $\mathbf{x}_f \in C_l$, cuja similaridade média aos objetos de C_d é máxima.
6. Se a similaridade média de $\mathbf{x}_f$ aos objetos de C_d for maior do que sua similaridade com C_l, então $\mathbf{x}_f$ será adicionado em C_d e volta-se ao Passo 5; Caso contrário, nada é feito.
7. O algoritmo pára se todos os grupos forem formados por um único objeto. Caso contrário, retorna-se ao Passo 2.

Depois de executado, cada elemento irá pertencer ao seu próprio grupo. O algoritmo possui uma métrica para avaliar quão necessária foi uma certa divisão. Esta métrica é

chamada de DC (abreviação do termo em inglês Divisive Coeficient, ou em português Coeficiente de Divisão). Baixos valores de DC indicam que foi realizada uma divisão desnecessária. DC é dado por:

$$DC = \frac{\sum_{\mathbf{x} \in X} diametro(\mathbf{x})}{diametro(X)} \tag{17}$$

em que X é o conjunto de dados, $diametro(\mathbf{x})$ é o diâmetro do último grupo no qual o objeto $\mathbf{x}$ pertenceu antes da divisão para formar o grupo de um elemento e $diametro(X)$ é o diâmetro do conjunto de dados. Quanto maior for o coeficiente de divisão, mais forte será a estrutura agrupada. Por este motivo, é possível escolher uma partição dentre a hierarquia formada por meio dos valores do coeficiente de divisão. Se a divisão feita em um nível da hierarquia reduzir o valor do coeficiente de divisão, esta divisão pode ser desconsiderada.

2.12.3 K-means (K-médias)

K-means utiliza o conceito de centróides como protótipos representativos dos grupos. O centróide representa o centro de um grupo, sendo calculado pela média de todos os objetos do grupo.

O funcionamento do K-means pode ser descrito pelos seguintes passos:

1. Atribuem-se valores iniciais para os protótipos seguindo algum critério, por exemplo, sorteio aleatório desses valores dentro dos limites de domínio de cada atributo;
2. Atribui-se cada objeto ao grupo cujo protótipo possua maior similaridade com o objeto;
3. Recalcula-se o valor do centróide (protótipo) de cada grupo, como sendo a média dos objetos atuais do grupo;
4. Repete-se os passos 2 e 3 até que os grupos se estabilizem;

K-means possui uma complexidade computacional equivalente a $O(N * k * T)$, pois, a cada iteração (T), é calculada a distância entre os N objetos até cada um dos k centróides. Se considerarmos nessa medida de complexidade o número de atributos, passaríamos para $O(N * k * T * n)$, já que cada comparação entre dois objetos tem complexidade computacional $O(n)$.

A Figura 17 ilustra um exemplo de execução do algoritmo K-means. Em 17 (a) temos a execução do passo 1 do algoritmo, a inicialização dos protótipos; Em 17 (b) temos a execução do passo 2, a atribuição de cada objeto ao grupo cujo protótipo possui maior similaridade ao objeto; Em 17 (c) temos a execução do passo 3, o recálculo dos valores dos protótipos, como sendo a média dos objetos atualmente presentes em cada grupo; Em 17 (d) temos a preparação para a repetição dos passos 2 e 3, o desvinculamento dos objetos dos grupos a que pertenciam; em 17 (e) e 17 (f) a repetição dos passos 2 e 3, respectivamente.

O algoritmo continuaria a repetir os passos 2 e 3 enquanto ocorrer alterações nos grupos.

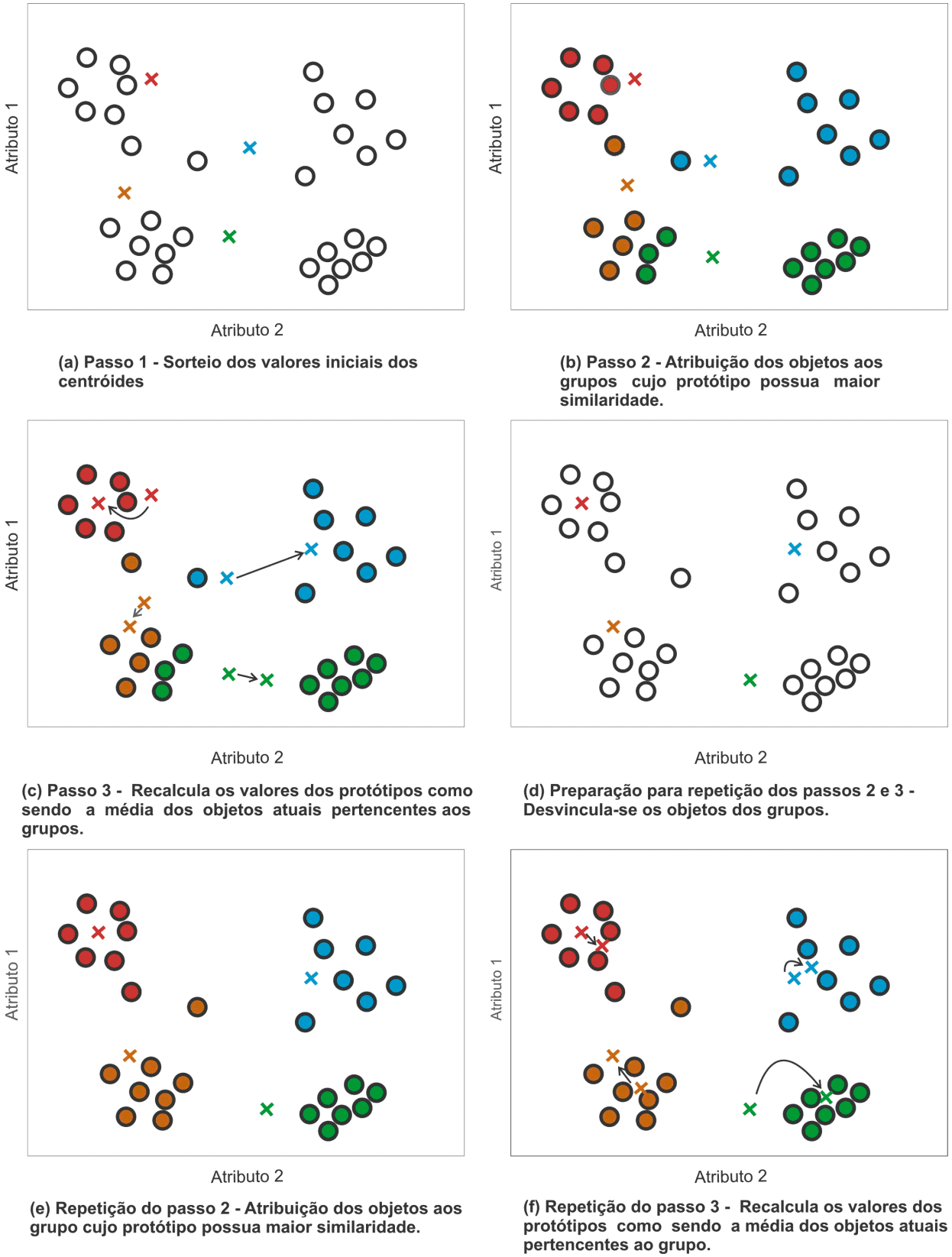


Figura 17: Exemplo de Execução do algoritmo K-means

Determinar o número de grupos k de forma prática geralmente é um procedimento baseado em dois passos. Primeiramente o algoritmo é executado múltiplas vezes, de forma que o número de grupos k das partições geradas varie dentro de um intervalo pré-definido, para diferentes inicializações de protótipos. Segundo, é necessário utilizar um índice de validação para estimar a qualidade das partições geradas. Alguns destes índices são apresentados na Seção 2.10.

Uma vez que tenha sido escolhida uma medida apropriada para avaliar a qualidade relativa das partições, é possível estudar estratégias para executar k-médias com o número de grupos a ser determinado. Um dos métodos consiste em executar o algoritmo k-médias para um número crescente de grupos k, dentro de um determinado intervalo. Para cada valor de k, um número $n_{\pi,k}$ de partições devem ser geradas com diferentes inicializações. A partição final é escolhida dentre o conjunto obtido, por meio da qualidade aferida pelo índice de validação relativo. Este tipo de execução é aqui chamado de Múltiplas Execuções Ordenadas de k-médias (*OMRk*, do inglês *Ordered Multiple Runs of k-means*) (Naldi et al., 2009).

Estima-se que a complexidade do algoritmo *OMRk* seja $O(N \cdot k_{max}^2)$ (Naldi et al., 2009).

2.12.4 Bisecting k-means

O algoritmo *Bisecting k-means* (Steinbach et al., 2000) consiste em uma variação hierárquica do algoritmo k-médias (Seção 2.12.3). Em cada iteração, ele seleciona um grupo e o divide, de forma a gerar uma hierarquia. Seu funcionamento pode ser descrito pelos passos:

1. O algoritmo é inicializado com um único grupo que contenha todos os objetos do conjunto de dados.
2. Seleciona-se um grupo C_s.
3. Divide-se C_s por meio da aplicação de k-médias, com $k = 2$.
4. Repete-se o passo anterior por *inter* vezes e seleciona-se a divisão que produza o agrupamento com a maior similaridade entre objetos e os centros de seus grupos.
5. Repete-se os passos 2,3,4 até que o número de grupos desejado seja alcançado.

O grupo C_s por ser escolhido de diversas formas, seja pelo maior tamanho (e.g., número de objetos ou diâmetro) ou menor similaridade entre os objetos e centróide (e.g. volume). O número de vezes que o algoritmo k-médias é aplicado por iteração, *inter*, e o número de grupos da partição final são parâmetros a serem definidos pelo usuário. O *Bisecting k-means* pode resultar em uma única partição ou em uma hierarquia e sua complexidade varia com o método de seleção de C_s. Se a seleção for feita a partir da quantidade de objetos do grupo, a complexidade do algoritmo é linear ($O(N)$) (Steinbach et al., 2000). O uso de métodos mais sofisticados, como diâmetro ou volume, pode aumentar a complexidade do algoritmo.

Neste caso, existem várias formas de se definir o diâmetro de um grupo, uma delas poderia ser: a distância entre o par de objetos mais distantes entre si que pertençam ao grupo.

A Figura 18 ilustra as duas primeiras iterações do algoritmo Bisecting K-means para uma base de dados. Em 18 (a) temos a inicialização da partição com um único grupo que contém todos os objetos, em 18 (b) e 18 (c) temos, respectivamente, o resultado da primeira e da segunda iteração do algoritmo. É importante ressaltar que nas demais iterações do algoritmo, (não ilustradas), os grupos continuarão a se dividir até que o número de grupos pré-determinado seja alcançado.

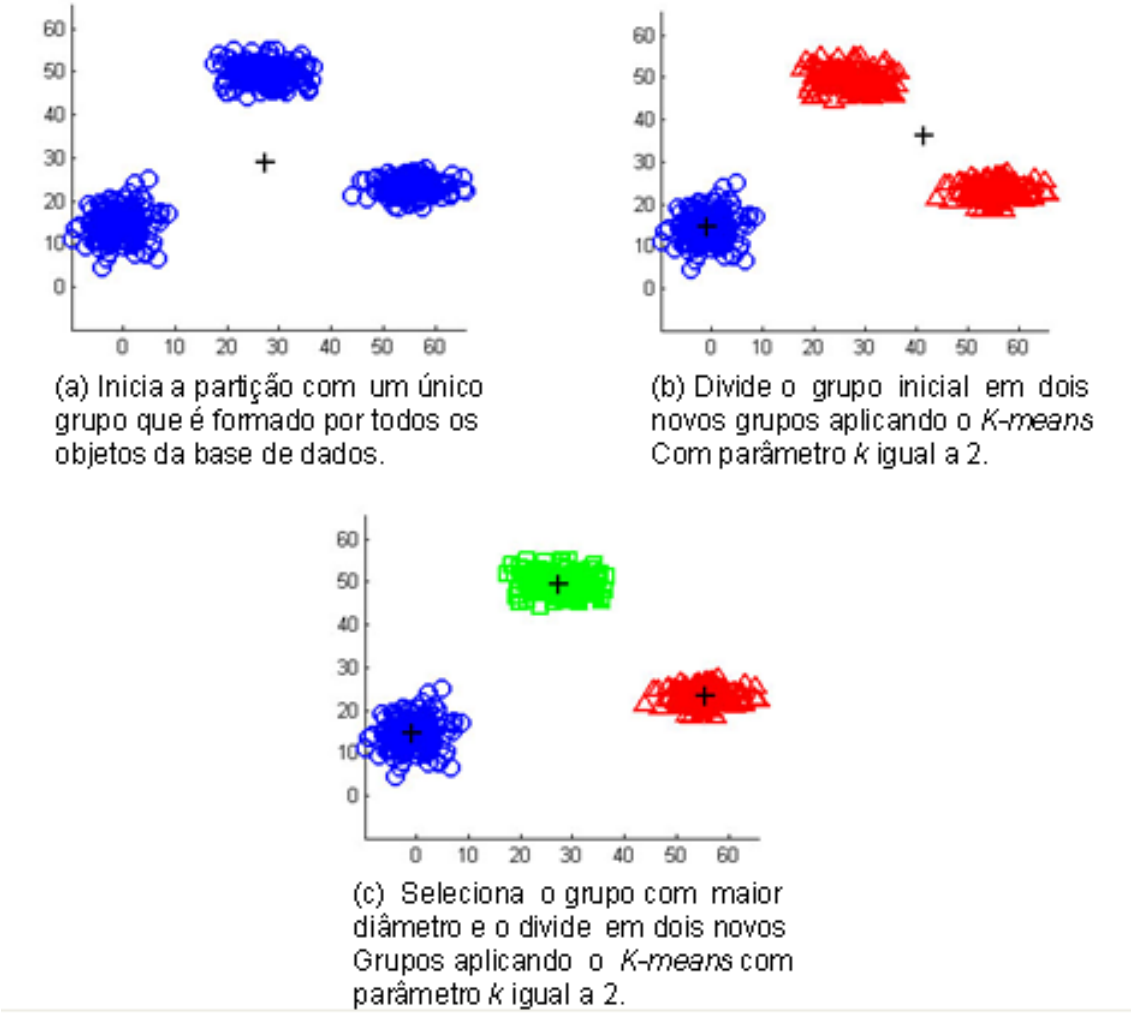


Figura 18: Exemplo de execução do Bisecting K-means

Caso C_s for selecionado por seu diâmetro e o algoritmo for executado até formar o dendrograma completo, ou seja, até forma grupos com um único objeto, sua complexidade pode ser estimada em $O(N^2 \cdot T)$. Neste caso o algoritmo realiza $N-1$ divisões durante sua total execução, (uma divisão para cada objeto da base de dados a mais que um), cada uma dessas divisões executa K-means e portanto tem complexidade computacional proporcional ao número de objetos no grupo a ser dividido. Peguemos o pior caso, onde cada divisão de um grupo gere um singleton e outro grupo com todos os demais objetos. As execuções do K-means com parâmetro k igual a 2 teriam uma complexidade computacional que poderia ser estimada como:

$$O((N*2*T)+((N-1)*2*T)+((N-2)*2*T)+...+(2*2*T))$$

que ao isolarmos o termo $(2*T)$ pode ser simplificada em:

$$O((2+3+...+(N-1)+N)*2*T)$$

se somado e subtraido o termo $(1 * 2 * T)$, resulta em:

$$O((1 + 2 + 3 + ... + (N - 1) + N) * 2 * T) - (1 * 2 * T)$$

mas, como a relação $O(1 + 2 + 3 + ... + (N - 1) + N) = O(N^2)$ é conhecida, pode ser simplificada em:

$$O(N^2 - 1 * 2 * T)$$

e como, $O(N^2 - 1 * 2 * T) = O(N^2)$, reestimamos a complexidade do algoritmo em:

$$O(N^2 * T * 2).$$

Como 2 é uma constante, pode ser removida do cálculo da complexidade resultando em uma complexidade assintótica $O(N^2 * T)$. O algoritmo Bisecting k-means possui o mesmo problema de outros algoritmos hierárquicos, no qual uma vez que um grupo seja formado sua estrutura se propaga pelas iterações posteriores.

2.12.5 X-means

O algoritmo *X-means* foi proposto por Pelleg and Moore (2000) para gerar uma partição do conjunto de dados por meio do uso de k-médias. O algoritmo recebe como parâmetro de entrada a base de dados a ser particionada e um intervalo de número de grupos, aqui representado como $[k_{min}, k_{max}]$ e pode ser descrito pelos seguintes passos:

1. Inicializa-se o algoritmo por meio da aplicação de k-means ao conjunto de dados, de forma a gerar uma partição com k_{min} grupos.
2. A partição resultante é avaliada, armazenando-a em seguida.
3. Realiza-se a divisão de cada grupo em dois, por meio de k-means.
4. Cada divisão efetuada é avaliada e as divisões que geraram as melhores partições são mantidas, de forma a não gerar mais do que k_{max} grupos.
5. Caso nenhuma partição seja mantida, divide-se uma proporção dos grupos formados inicialmente (usualmente a metade), a partir das divisões que resultem nas melhores avaliações. Respeita-se o limite de k_{max} grupos.
6. O algoritmo k-means é aplicado para o refinamento da partição resultante.
7. Se a partição resultante for a melhor avaliada, ela é armazenada.
8. Caso a partição resultante possua k_{max} grupos o algotimo pára, retornando-se a partição melhor avaliada. Caso contrário, o algoritmo retorna ao Passo 3.

O critério *BIC* é utilizado por Pelleg and Moore (2000) para avaliar as partições durante a execução do algoritmo. Os protótipos fornecidos ao k-means são iniciados na mesma posição do centróide original do grupo a ser dividido e, em seguida, são deslocados no sentido oposto na direção escolhida aleatoriamente dentro do espaço de dados. O valor do deslocamento deve ser proporcional ao tamanho da região do grupo. Em geral essas divisões são avaliadas pelo critério BIC que indica se a divisão realizada gerou melhora, ou não, para a qualidade da partição.

3 Comparação de metodologias de estimação de número de grupos

Muitos dos algoritmos de agrupamento de dados que são propostos na literatura necessitam que o número de grupos seja previamente fornecido pelo o usuário. Como essa informação é usualmente desconhecida pelo usuário, a estimação automática do número de grupos presentes em uma base de dados é um importante foco de estudo da área de agrupamentos de dados.

Este trabalho tem como objetivo estudar e comparar metodologias de estimação do número de grupos de bases de dados, particularmente as metodologias que tenham como base o algoritmo clássico K-means. Por este motivo, as metodologias de estimação de número de grupos estudadas neste trabalho são variações propostas do algoritmo Bisecting K-means (Steinbach et al., 2000), o algoritmo X-means (Pelleg and Moore, 2000) e variações propostas, e múltiplas execuções do algoritmo K-means.

3.1 Bisecting K-means como estimador de número de grupos

Neste trabalho modificamos o Bisecting K-means para que ele tenha a capacidade de avaliar quando uma divisão de um grupo era necessária, ou não. Para esta avaliação utilizamos o critério BIC (Seção 2.10.2), ou uma versão simplificada da silhueta, que comparam a qualidade do grupo antes da divisão a ser avaliada com a qualidade dos dois novos grupos gerados.

A Figura 19 ilustra a utilização do critério BIC para avaliar uma divisão do Bisecting K-means, para o BIC quanto maior for o seu valor melhor é a partição avaliada, assim o critério avalia a divisão necessária se o BIC após a divisão tiver um valor maior do que o BIC antes da divisão.

Como o valor do BIC após a divisão é maior que o valor calculado antes da divisão, podemos dizer que, para este caso, segundo o critério BIC a divisão era necessária, ou seja, gerou melhora na qualidade da partição.

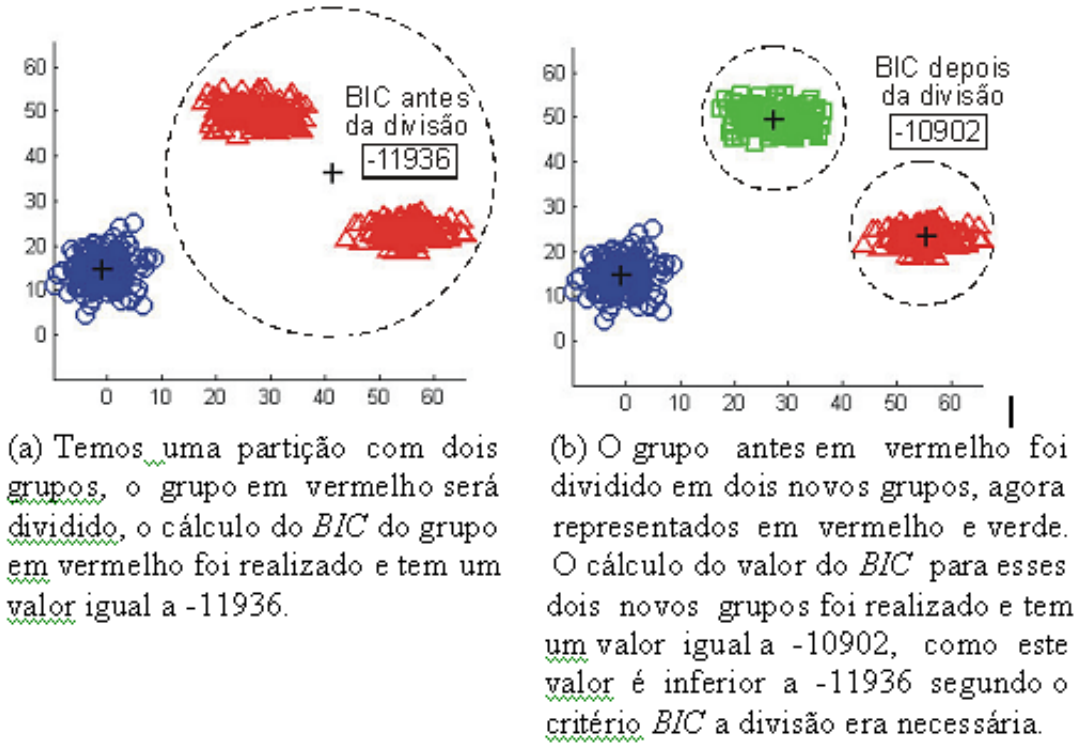
Quanto à forma de calcular o diâmetro dentre várias possibilidades trabalhamos com dois cenários:

- a) O diâmetro é calculado com o valor igual a duas vezes a maior distância entre o centróide e seus objetos;
- b) O diâmetro é calculado com o valor igual a duas vezes a média das distâncias entre o centróide e seus objetos;

As duas formas de calcular o diâmetro foram mantidas, pois a primeira considera apenas o objeto mais distante ao centróide tem uma tendência a calcular com valores mais altos os diâmetros de grupos que possuam *outliers*, indicando a divisão desses grupos como prioridade, o que, pode ser um fator positivo em alguns casos. Já a segunda é mais robusta a presença de *outliers*, pois possui uma maior consistência por ser calculada por meio da média das distâncias dos objetos ao centróide.

Também definimos dois cenários para o critério de parada para o algoritmo:

- i) Avalia-se a divisão do grupo com maior diâmetro, se não houver melhora a divisão não é efetuada e finaliza-se a execução;



(a) Temos uma partição com dois grupos, o grupo em vermelho será dividido, o cálculo do *BIC* do grupo em vermelho foi realizado e tem um valor igual a -11936.

(b) O grupo antes em vermelho foi dividido em dois novos grupos, agora representados em vermelho e verde. O cálculo do valor do *BIC* para esses dois novos grupos foi realizado e tem um valor igual a -10902, como este valor é inferior a -11936 segundo o critério *BIC* a divisão era necessária.

Figura 19: Avaliação de uma divisão do Bisecting K-means por meio do BIC

- ii) Avalia-se a divisão de todos os grupos em ordem decrescente de diâmetro. Se não houver melhora na divisão de nenhum deles, ela não é efetuada e finaliza-se a execução.

Optamos por utilizar o parâmetro α, que indica o número de múltiplas execuções do K-means ao realizar cada divisão, com o valor igual a 1. Essa opção tem como objetivo possibilitar uma comparação mais justa do Bisecting K-means com o algoritmo X-means, que realiza apenas uma execução do K-means em cada uma de suas divisões.

3.2 Implementações do Bisecting K-means

No total, foram implementadas oito versões do Bisecting K-means capazes de estimar o número de grupos de uma base de dados e gerar uma partição com esse número de grupos. As versões diferem-se em três características: a maneira como o diâmetro dos grupos é calculado; o critério de parada utilizado e o critério de validação das partições, utilizado para quantificar a melhora do uso de uma divisão de grupo.

As versões são geradas a partir de oito combinações possíveis dessas três características. A Tabela 2 contém as características de cada versão utilizada:

3.3 Problemas encontrados no Bisecting K-means

O Bisecting K-means por ser um algoritmo diviso possui o seguinte problema: uma vez que objetos que pertençam a um mesmo grupo sejam dividos em grupos distintos, eles nunca mais irão pertencer ao mesmo grupo, ou seja, a estrutura se mantém nas próximas iterações do algoritmo. A princípio pode ser dito que o uso de critérios de avaliação da qualidade utilizados na divisão previnem divisões inadequadas. Contudo, um problema

Versão	Cálculo do diâmetro	Critério de parada	Critério de avaliação
1	(a) Maior Distancia Objetos / Centróides	(i) Grupo com maior diâmetro não foi dividido	Silhueta
2	(a) Maior Distancia Objetos / Centróides	(ii) Todos os grupos não foram divididos	Silhueta
3	(b) Média das Distâncias Objetos / Centróides	(i) Grupo com maior diâmetro não foi dividido	Silhueta
4	(b) Média das Distâncias Objetos / Centróides	(ii) Todos os grupos não foram divididos	Silhueta
5	(a) Maior Distancia Objetos / Centróides	(i) Grupo com maior diâmetro não foi dividido	BIC
6	(a) Maior Distancia Objetos / Centróides	(ii) Todos os grupos não foram divididos	BIC
7	(b) Média das Distâncias Objetos / Centróides	(i) Grupo com maior diâmetro não foi dividido	BIC
8	(b) Média das Distâncias Objetos / Centróides	(ii) Todos os grupos não foram divididos	BIC

Tabela 2: Oito diferentes versões do Bisecting K-means.

acontece quando o grupo a ser dividido possui três ou mais grupos conhecidos contidos nele, pois por exemplo, ao se dividir um grupo que contenha três grupos conhecidos em dois, por exemplo, pode ser gerado dois grupos contendo um grupo conhecido mais uma parte dos objetos do terceiro grupo conhecido.

As figuras 20 e 21 ilustra uma divisão de um grupo com sete grupos reais pelo Bisecting K-means, em que na Figura 20 temos em destaque o grupo que será dividido pelo algoritmo. Na Figura 20 estão presentes os dois grupos gerados depois da divisão e em destaque temos os objetos de dois grupos reais que foram divididos dos seus grupos. Esses objetos não serão unidos novamente pelo resto da execução do algoritmo, o critério de validade provavelmente avaliará positivamente esta divisão, pois em a partição gerada é melhor com sete grupos reais divididos em dois grupos do que sete grupos unidos.

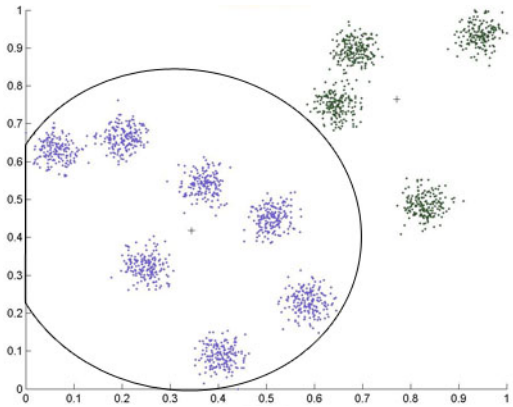


Figura 20: Divisão do Bisecting K-means: o grupo em destaque que possui sete grupos conhecidos será dividido utilizando K-means com parâmetro $k = 2$.

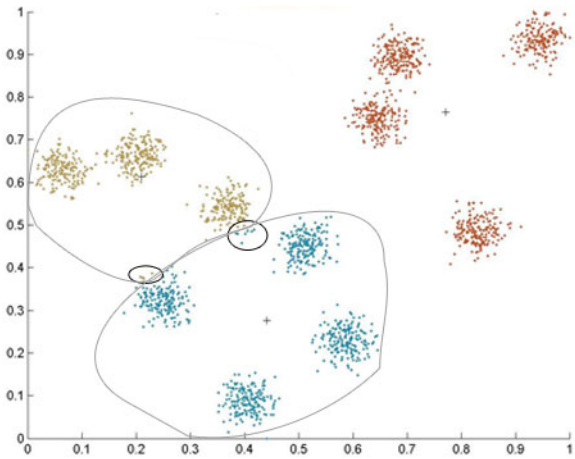


Figura 21: Divisão do Bisecting K-means: os dois novos grupos resultantes da divisão. Alguns dos objetos de dois grupos conhecidos foram separados de seus grupos originais e não serão unidos pelo resto da execução do algoritmo.

3.4 Diferenças entre implementações do X-means e seu artigo original

O algortimo X-means, descrito na Seção 2.12.5, foi originalmente publicado por Pelleg and Moore (2000) e sua implementação foi disponibilizada pelo autor em ansi C (`http://www.cs.cmu.edu/~dpelleg/kmeans.html`). Posteriormente, uma vesão de seu algoritmo foi adicionada a versão de desenvolvimento do conhecido programa Weka (Witten and Frank, 2005), disponibilizada para download em (`http://www.cs.waikato.ac.nz/ml/weka/`). Contudo, durante o desenvolvimento desta pesquisa foram encontradas informações nos versões dos códigos disponibilizados que não estavam presentes ou mostravam-se distintas das contidas em seu artigo original (Pelleg and Moore, 2000). Nesta seção serão apresentadas as principais diferenças encontradas entre as versões do algoritmo. Contudo, é possível que outras diferenças não relatadas possam existir.

3.4.1 Parâmetros a serem definidos pelo usuário

Apesar de não haver referência em (Pelleg and Moore, 2000), as implementações feita pelo autor do algoritmo em ansi C e a disponibilizada pelo Weka em Java possuem alguns parâmetros que precisam ser fornecidos pelo usuário para que ele seja executado. Nesta seção serão apresentados os principais parâmetros e repassar os valores padrões encontrados no arquivo README que vem em conjunto com a implementação ansi C.

- Número de grupos inicial (Parâmetros k (versão ansi C) e $minNumClusters$ (versão Java)): Número de grupos com que o algoritmo irá iniciar a partição antes que qualquer divisão seja realizada. O autor sugere que caso o usuário tenha algum conhecimento prévio do intervalo de número de grupos da base a ser particionada que ele forneça o menor valor do intervalo, caso contrário este parâmetro deve ser igual a 1.

- Máximo número de grupos (Parâmetros max_ctrs (versão ansi C) e $maxNumClusters$ (versão Java)): Número de grupos máximo ao qual o algoritmo não deve ultrapassar ao fazer suas divisões, este parâmetro serve como condição de parada do algoritmo. O autor sugere que caso o usuário tenha algum conhecimento prévio do intervalo de número de grupos da base a ser particionada que ele forneça o maior valor do intervalo.

- Número máximo de iterações locais do K-means (Parâmetros *max_iter* (versão ansi C) e *maxKmeansForChildren* (versão Java)): Número máximo de iterações do K-means a serem realizadas em cada uma das divisões dos grupos. O autor indica 200 como um bom valor para este parâmetro. Caso o usuário queira que o algoritmo execute mais rápido o autor recomenda a utilização de valores menores que 50, pois estes não geram grandes perdas de qualidade. Neste caso o autor indica 20 como um valor adequado.

- Número máximo de iterações globais do K-means (Parâmetros *max_iter* (versão ansi C) e *maxKmeans* (versão Java)): Número máximo de iterações do K-means a serem realizadas globalmente a cada iteração do X-means. O autor indica 200 como um bom valor para este parâmetro. Caso o usuário queira que o algoritmo execute mais rápido o autor recomenda a utilização de valores menores que 50, pois estes não geram grandes perdas de qualidade. Neste caso, o autor indica 20 como um valor adequado. A versão em ansi C utiliza o mesmo parâmetro para determinar o número de iterações locais e globais do K-means.

- Número de divisões (Parâmetros *num_splits* (versão ansi C) e *maxIterations* (versão Java)): Corresponde ao número de iterações do processo de divisão que o algoritmo irá realizar no máximo, essas iterações foram descritas na Seção 2.12.5 e correspondem aos passos 3 a 6, este parâmetro serve como condição de parada do algoritmo.

- *Cut off factor* (Parâmetros *cutoff_factor* (versão ansi C) e *cutOffFactor* (versão Java)): Determina qual proporção dos grupos será dividida no caso de nenhuma das divisões ter melhorado a qualidade da partição. Como este parâmetro trata-se de uma proporção ele deve pertencer ao intervalo [0,1], o autor indica 0,5 como o valor ideal para este parâmetro.

3.4.2 Diferenças entre artigo e implementação em ansi C

Nesta seção apresentaremos as diferenças encontradas entre a descrição do algoritmo em (Pelleg and Moore, 2000) e a implementação feita pelo autor em ansi C.

- Critério de parada: em (Pelleg and Moore, 2000) o único critério de parada do algoritmo é a formação de uma partição com um número de grupos igual à k_{max}. No entanto, é possível que em uma de suas iterações, o algoritmo gere uma partição com um número menor de grupos do que o necessário para satisfazer o critério de parada e possua apenas grupos que não deveriam ser divididos, segundo a avaliação (o critério BIC). Essa partição não seria em nenhuma iteração alterada pelo algoritmo, pois as partições formadas em suas divisões não resultarão em em uma melhor avaliação segundo o critério BIC. Isto resultaria em um ciclo infinito de iterações já que a condição de parada não seria atendida. Este problema é solucionado na implementação do algoritmo feita em ansi C fornecida pelo próprio autor. Ao se deparar com uma partição onde nenhum dos grupos irá gerar grupos com BIC maior, o algoritmo resultante divide uma proporção dos grupos atuais (usualmente a metade), se este valor não ultrapassar k_{max}. Caso ultrapasse, divide-se os grupos de forma a resultar em k_{max}. No entanto, Pelleg and Moore (2000) não deixam claro que este recurso é utilizado pelo algoritmo e, inclusive, afirmam que não é uma solução adequada para o algoritmo (página 3 do mencionado artigo). Além disso, a versão em ansi C possui um outro critério de parada que se trata do número de iterações máxima do X-means que o algoritmo irá realizar.

- Cálculo do BIC: o cálculo do BIC (Seção 2.10.2) difere em relação ao parâmetro p e na estimação da máxima verossimilhança. O valor de p corresponde ao número de parâmetros da partição e é calculado em (Pelleg and Moore, 2000) por $p = k - 1 + nk + 1$, enquanto sua implementação em ansi C calcula $p = k - 1 + nk + k$, em que k é o número de grupos e n é o número de atributos da base de dados. Na página 4 de (Pelleg and Moore, 2000), a estimativa de máxima verossimilhança é calculada pela Equação 18:

$$\sum_{i=1}^{k} \hat{l}(D_i) = -\left(\frac{N_i}{2}\ln(2\pi)\right) - \left(\frac{N_i \cdot n}{2}\ln(\hat{\sigma}^2)\right) - \left(\frac{N_i - k}{2}\right) + (N_i \ln N_i) - (N_i \ln N) \tag{18}$$

em que N_i é o número de objetos de i-ésimo grupo e $\hat{\sigma}^2$ é a estimação da variância, calculada por:

$$\hat{\sigma}^2 = \frac{1}{N-k}\sum_{i=1}^{N}(\mathbf{x}_i - \mu_{(i)})^2 \tag{19}$$

$\mathbf{x}_i$ é o i-ésimo objeto da base de dados, e $\mu_{(i)}$ é o centróide do grupo que contém o i-ésimo objeto. A versão em ansi C estima a máxima verossimilhança pela Equação 20 (arquivo split.c linhas: 525 a 549):

$$\sum_{i=1}^{k} \hat{l}(D_i) = -\left(\frac{N_i}{2}\ln(2\pi)\right) - \left(\frac{N_i \cdot n}{2}\ln(\hat{\sigma}^2)\right) - \left(\sum_{\mathbf{x}_j \in D} ||\mathbf{x}_j - \mu_i||\right) + (N_i \ln N_i) - (N_i \ln N) \tag{20}$$

Comparando a Equação 18 com a Equação 20, notamos que o terceiro parâmetro é diferente nas duas equações. Não foi encontrada nenhuma justificativa para esta mudança no cálculo da estimativa de máxima verossimilhança.

Na implementação em ansi C o valor da estimativa de máxima verossimilhança de um grupo singleton é definido como 0 (arquivo split.c linha: 534). Não há referência sobre este caso no artigo (Pelleg and Moore, 2000).

Todos os logaritmos presentes nas fórmulas do artigo correspondem a logaritmos neperianos, o que também não é indicado no artigo (Pelleg and Moore, 2000).

3.4.3 Diferenças entre artigo e implementação em Java

Nesta seção serão apresentadas as diferenças encontradas entre a descrição do algoritmo no artigo original (Pelleg and Moore, 2000) e a versão implementada em Java, disponibilizada no Weka (Witten and Frank, 2005).

- Inicialização do k-means: na implementação em Java os protótipos tem seus valores inicializados por meio do sorteio aleatório de objetos representativos (medóides). Não há descrição desta forma de inicialização no artigo original.

- Critério de parada: como descrito na Seção 3.4.2, o único critério de parada descrito no artigo original é a formação de uma partição com um número de grupos igual

à k_{max}. Porém, é possível que uma partição encontrada seja avaliada de maneira a não ser mais dividida durante a execução do algoritmo. Como descrito na Seção 3.4.2, esse problema gera um ciclo infinito de iterações já que a condição de parada não seriam atendidas.

Assim como na versão implementada em ansi C, esse problema é solucionado na versão desenvolvida para o Weka. Ao se deparar com uma partição onde nenhum dos grupos deve ser dividido, o algoritmo divide uma proporção dos grupos atuais (usualmente a metade), se este valor não ultrapassar k_{max}. Caso ultrapasse divide-se os grupos de forma a resultar em k_{max}. A versão em Java possui um outro critério de parada, que se trata do número de iterações máxima do X-means que o algoritmo irá realizar (função buildClusterer do código disponível em `http://www.cs.waikato.ac.nz/ml/weka/`).

- Cálculo do BIC: assim como na versão em ansi C, o parâmetro p é calculado por $p = k - 1 + nk + k$ (função calculateBIC). O calculo da função de verossimilhança também se diferencia do artigo original e é dado pela Equação 21 (função logLikelihoodEstimate):

$$\sum_{i=1}^{k} \hat{l}(D_i) = -\left(\frac{N_i}{2}\ln(2\pi)\right) - \left(\frac{N_i \cdot n}{2}\ln(\hat{\sigma}^2)\right) - \left(\frac{N_i - k}{2}\right) + (N_i \ln N_i) \tag{21}$$

de forma que $\hat{\sigma}^2$ seja calculado pela Equação 22 (função logLikelihoodEstimate):

$$\hat{\sigma}^2 = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_i - \mu_{(i)})^2 \tag{22}$$

Comparando a Equação 18 com 21 é possível notar que o terceiro parâmetro das equações é diferente e que o quinto parâmetro da 18 não existe na Equação 21, além do fato do cálculo da variância ser diferente nas duas equações. Assim como a versão em ansi C, a máxima verossimilhança de um grupo singleton é estimado como 0 (função logLikelihoodEstimate) e todos os logaritmos presentes nas fórmulas correspondem a logaritmos neperianos. Estas informações não estão descritas no artigo (Pelleg and Moore, 2000). Comparando a Equação 22 com a Equação 19, existe diferença no número de graus de liberdade utilizado no cálculo da variância.

- Normalização: não há qualquer referência no artigo sobreo uso de normalização dos dados no algoritmo. Contudo, o Weka aplica normalização dos dados para o intervalo $[0, 1]$ (função m_DistanceF.distance iniciada como m_DistanceF = new EuclideanDistance()) antes de agrupa-los. Esse fator interfere no resultado do algoritmo, já que o critério BIC apresenta grande variação com a normalização dos dados.

3.4.4 Diferenças entre implementação em Java e implementação em ansi C

Nesta seção apresentaremos as diferenças encontradas entre a implementação feita pelo autor em ansi C e a implementação em Java disponibilizada no Weka.

- Cálculo do BIC: na versão implementada em ansi C a estimativa de máxima verossimilhança é calculada pela Equação 20 (arquivo split.c linhas: 525 a 549), enquanto

que na versão em Java a estimativa de máxima verossimilhança é calculada pela Equação 21 (função logLikelihoodEstimate). Também existe diferença nos graus de liberdade no cáculo da variança entre as duas versões, dadas pelas Equações 19 e 22 respectivamente.

- Normalização: diferentemente da versão ansi C, o Weka aplica normalização dos dados para o intervalo $[0, 1]$ (função m_DistanceF.distance iniciada como m_DistanceF = new EuclideanDistance()) antes de agrupa-los, o que faz com que o BIC apresente grande variação com a normalização dos dados.

3.5 Implementações originais do X-means

Nesta seção descreveremos as versões originais de X-means implementadas. O objetivo destas versões é gerar versões do algoritmo que possam ser comparadas, em termos de tempo de execução, com outros algoritmos implementados nesta plataforma. Outra preocupação foi que esta versão fosse fiel a descrição do algoritmo feita em (Pelleg and Moore, 2000), porém, como muitos detalhes necessários para implementação não estavam claros, recorremos aos códigos das versões implementada pelo autor em ansi C e a versão Java disponibilizada na Weka.

Devido a existência de diferentes formas de calcular a estimativa de máxima verossimilhança nas versões do X-means (discutidas na Seção 3.4), optamos por realizar este cálculo da forma com que esta indicado no livro (Mirkin, 2005), pois esta referência aborda o cálculo deste índice de uma forma tradicionalmente abordada na literatura.

Outra implementação do X-means também foi implementada, utilizando a versão simplificada da silhueta (Seção 2.10.3) no lugar do critério BIC. Assim, foram implementadas duas versões do X-means que diferem apenas no critério de validade utilizado para avaliar as divisões realizadas pelo algoritmo. Nesta seção, serão descritas inicialmente as características em comum de ambas versões e, em seguida, serão descritas as particularidades de cada versão, referentes aos critérios de validade e aos ajustes necessários para utilizá-los.

3.5.1 Características comuns de ambas versões

Ambas versões implementadas do X-means possuem como parâmetros de entrada: a base de dados a ser particionada; um intervalo do número de grupos ao qual a base possui $[k_{min}, k_{max}]$, um valor pra o *cut_off_factor* que deve variar entre $[0, 1]$ e corresponde a proporção dos grupos que serão divididos quando a divisão de nenhum grupo for avaliada positivamente pelo critério de validade utilizado; e o número de iterações máximo que o K-means realizará em cada execução.

Ambos algoritmos podem ser resumidos nos seguintes passos:

1. Gera-se uma partição inicial executando K-means com 2 grupos.
2. Armazena-se a partição inicial e o valor resultante do critério de validade nas variáveis *melhor_partição* e *melhor_critério*, respectivamente.
3. A partição inicial é considerada a partição atual.
4. Calcula-se o valor do critério de validade para cada um dos grupos da partição atual e armazena o resultado no vetor **cvgp** (critério de validade dos grupos pais).

5. Realiza a divisão de cada grupo da partição atual em dois novos grupos por meio do K-means, de forma que os novos rótulos e novos centróides gerados sejam armazenados.

6. Calcula-se o valor do critério de validade para cada par de grupos gerados pelas divisões e armazena esses valores no vetor **cvgf** (critério de validade dos grupos filhos).

7. Calcula-se a melhoria que a divisão de cada grupo gerou, armazenando o resultato em no vetor **vmqg** (vetor de melhoria da qualidade dos grupos). O valor da melhoria é armazenado na i-ésima posição deste vetor, sendo calculado pela diferença entre os valores contidos na mesma posição dos vetores **cvgf** e **cvgp** (**vmqg** = **cvgfi** - **cvgpi**).

8. Gera-se um vetor **gsd** (grupos a serem divididos) ordenado de forma decrescente pelos valores de **vmqg** dos grupos pertencentes a partição atual.

9. Checa se existe ao menos um valor de **vmqg** positivo. Se sim vai para o passo 10, caso contrário vai para o passo 18.

10. Checa se **vmqg** do primeiro grupo do vetor **gsd** é positivo. Se for realiza a divisão do grupo utilizando as informações armazenadas no passo 5, caso contrário executa o passo 13.

11. Remove o primeiro elemento da lista **vmqg**.

12. Checa se o número de grupos atual é menor do que k_{max}, se for reexecuta o passo 9, caso contrário executa o passo 13.

13. Executa K-means globalmente na nova partição, aprimorando a estrutura.

14. Calcula o valor do critério de validade utilizado na nova partição. Se o valor calculado for maior que a variável *melhor_critério*, armazena o valor calculado na variável *melhor_critério* e a nova partição na variável *melhor_partição*.

15. Checa se o número de grupos é menor que k_{max}, se sim executa o passo 16, caso não executa o passo 23.

16. Armazena a nova partição como partição atual.

17. Volta ao passo 4.

18. Determina um valor de grupos a ser dividido que será igual a uma proporção dos grupos atuais equivalente ao parâmetro *cut_off_factor*, armazena esse valor na variável *nd* (número de divisões).

19. Divide o primeiro grupo do vetor **gsd** utilizando as informações armazenadas no passo 5.

20. Remove o primeiro grupo do vetor **gsd** e decrementa *nd* em 1.

21. Checa se o número de grupos atual ultrapassou k_{max}. Se sim executa o passo 13, caso contrário executa o passo 22.

22. Checa se *nd* é maior que zero, se sim retorna ao passo 19, caso contrário retorna ao passo 13.

23. Retorna a partição armazenada na variável *melhor_partição*.

Segundo Pelleg and Moore (2000), os protótipos fornecidos ao K-means ao realizar a divisão de cada grupo (Passo 5) são obtidos a partir da posição do centróide original do grupo a ser dividido e, em seguida, são deslocados no sentido oposto em uma direção escolhida aleatoriamente. O que não fica claro é o valor do deslocamento desses centróides ao longo da direção escolhida que, segundo o artigo, deve ser proporcional ao tamanho da região do grupo. Para solucionar essa questão, o código do autor em ansi C foi consultado e viu-se que o valor do deslocamento é igual ao valor do desvio padrão do grupo a ser dividido.

A partir das observações feitas, decidiu-se calcular o valor dos protótipos iniciais de cada grupo a ser dividido pelas equações a seguir:

$$\mathbf{prototype1} = \mathbf{centroid} + \sigma \cdot \mathbf{direction} \tag{23}$$

$$\mathbf{prototype2} = \mathbf{centroid} + \sigma \cdot \mathbf{direction} \tag{24}$$

em que **centroid** é o centróide do grupo a ser divido, σ é o desvio padrão dos objetos do grupo em relação ao centróide e **direction** é um vetor unitário cuja direção é sorteada aleatóriamente.

3.5.2 X-means com BIC

O próprio Dan Pelleg afirma em sua lista de discussão sobre X-means (`https://www.autonlab.org/mailman/archives/public/kmeans/2004/000039.html`) que, de fato há um erro no cálculo do BIC em seu artigo (Pelleg and Moore, 2000). Por este motivo, a versão do X-means implementada em Matlab utiliza o critério BIC para avaliar cada divisão realizada pelo algoritmo, segundo descrito em (Mirkin, 2005). Essa avaliação é feita calculando-se o valor do BIC do grupo a ser dividido e o valor do BIC no conjunto dos dois grupos resultantes da divisão, comparando os valores obtidos. Se o segundo for maior que o primeiro, a divisão é avaliada positivamente. Caso contrário, ela é avaliada negativamente.

Nesta versão foi possível utilizar uma otimização (no que diz respeito ao tempo computacional) do algoritmo que esta descrita no artigo (Pelleg and Moore, 2000). A técnica utilizada consiste em manter uma variável binária para cada grupo, que possui valor 0 indicando que o grupo não foi dividido na iteração anterior do algoritmo e 1 indicando que o grupo é proveniente de uma divisão na iteração anterior. Não será feita uma nova tentativa de divisão caso algum grupo possua um valor 0 para essa variável, a não ser que o grupo seja alterado em uma execução global do K-means. Nesse caso, a alteração faz com que a variável seja setada novamente para 1. Foi possível utilizar essa otimização porque o valor critério BIC para um grupo só sofre alterações se esse grupo sofrer alterações, não dependendo dos outros grupos da partição.

Contudo, não foi implementado o uso de *kd-trees* (Pelleg and Moore, 2000), uma heurística de otimização do K-means proposta pelo autor como "opcional". Também não foi utilizado o uso de *constraints clustering* (Pelleg and Baras, 2007), uma característica adicionada ao código desenvolvido pelos autores depois de sua publicação em (Pelleg and Moore, 2000). O não uso destas técnicas tem como objetivo a padronização do K-means a ser utilizado por todos os algoritmos comparados.

3.5.3 X-means com Silhueta

Uma versão orignal do algoritmo *X-means* foi implementada com o uso de uma versão simplificada da silhueta (Seção 2.10.3) para avaliar cada divisão realizada pelo algoritmo. Entretanto, o calculo do valor da silhueta de um grupo é dependente das dissimilaridades entre seus objetos e os outros grupos, portanto não é possível calcular a silhueta de um grupo sem essa informação. Sendo assim, não é possível utilizar a silhueta da mesma forma com que o critério BIC na avaliação da divisão dos grupos (Seção 2.12.5).

Por este motivo, a silhueta foi utilizada para avaliar as divisões dos grupos por meio de outra heurística, descrita pelos passos a seguir:

1. Calcula-se o valor da silhueta da partição atual (com todos os grupos).
2. Realiza a divisão do grupo escolhido em dois novos grupos. Isto gera uma partição auxiliar que será igual a partição atual, exceto pela substituição do grupo que foi dividido pelos dois novos grupos gerados.
3. Calcula-se o valor da silhueta da partição auxiliar.
4. Compara o valor das silhuetas calculadas. Se a silhueta do passo 3 for maior que a do passo 1, avalia a divisão positivamente e mantém-se a divisão. Caso contrário, a divisão não é efetuada.

Nesta versão não utilizamos a otimização feita para a versão com BIC (Seção 3.5.2), pois essa otimização assume que o critério utilizado não depende dos outros grupos da partição, o que não é verdade para o critério silhueta.

3.6 Bases de dados

Um conjunto de bases de dados, foi artificialmente gerado para executarmos experimentos com os algoritmos estudados, conforme descrição feita em (Milligan and Cooper, 1985) e em (Vendramin et al., 2008). Estas bases possuem grupos disjuntos com formatos levemente elipsoidais, aproximadamente esféricos.

O conjunto é composto por 108 tipos de bases se diferem em relação ao número de atributos, quantidade de grupos e na forma de balanceamento dos objetos entre cada um dos grupos. Os tipos são essencialmente todas as combinações possíveis dentre as variações dos número de atributos $n = 2, ..., 10$, número de grupos $k = 4, 6, 8, 10$ e três formas distintas de balanceamento, ou seja, a proporção com que os objetos são divididos entre os grupos. Cada tipo de base é replicado 3 vezes, totalizando 324 bases de dados com 1000 objetos cada.

As três formas de balanceamentos utilizadas são: (1) dividir os objetos igualmente entre os grupos, (2) um grupo menor que contenha 10% dos objetos em um grupo e o restante dos objetos distribuídos igualmente entre os outros grupos, (3) um grupo maior que contenha 60% dos objetos e o restante dos objetos distribuídos igualmente entre os outros grupos.

Devido a todos os algoritmos estudados terem relações com o K-means, tomamos o cuidado para que as bases de dados geradas tivessem uma estrutura que o K-means conseguisse reconhecer em pelo menos uma de suas inicializações. Assim garantimos que todas as 324 bases possuem ao menos uma inicialização de K-means (com o número de

grupos conhecido) que gera uma partição com um valor de jaccard igual ou superior a 0.8, por meio de sua aplicação do algoritmo com inicializações aleatórias.

3.7 Metodologia experimental

Como descrito anteriormente, este trabalho tem como objetivo investigar algumas das técnicas de múltiplas execuções de algoritmos de agrupamento e suas diferenças em termos de qualidade, complexidade e desempenho. A complexidade computacional dos algoritmos foi discutida juntamente com a descrição dos mesmos. Nesta seção será descrita a metodologia experimental utilizada para comparar a qualidade e o desempenho dos algoritmos descritos.

Foram comparados 14 versões dos algoritmos descritos neste trabalho:

1. Bisecting K-means com cálculo de diâmetro (a), critério de parada (i) e avaliação por silhueta (Seção 3.2).
2. Bisecting K-means com cálculo de diâmetro (a), critério de parada (ii) e avaliação por silhueta (Seção 3.2).
3. Bisecting K-means com cálculo de diâmetro (b), critério de parada (i) e avaliação por silhueta (Seção 3.2).
4. Bisecting K-means com cálculo de diâmetro (b), critério de parada (ii) e avaliação por silhueta (Seção 3.2).
5. Bisecting K-means com cálculo de diâmetro (a), critério de parada (i) e avaliação por BIC (Seção 3.2).
6. Bisecting K-means com cálculo de diâmetro (a), critério de parada (ii) e avaliação por BIC (Seção 3.2).
7. Bisecting K-means com cálculo de diâmetro (b), critério de parada (i) e avaliação por BIC (Seção 3.2).
8. Bisecting K-means com cálculo de diâmetro (b), critério de parada (ii) e avaliação por BIC (Seção 3.2).
9. OMRk com $n_{\pi,k} = 10$ e avaliação por silhueta (2.12.3).
10. OMRk com $n_{\pi,k} = 20$ e avaliação por silhueta (2.12.3).
11. X-means com avaliação por BIC (Seção 3.5.2).
12. X-means com avaliação por silhueta (Seção 3.5.3).
13. X-means original do autor com código em ansi C (`http://www.cs.cmu.edu/~dpelleg/kmeans.html`).
14. X-means original distribuido no Weka com código em Java (`http://www.cs.waikato.ac.nz/ml/weka/`).

Cada algoritmo implementado foi executado 30 vezes para cada uma das bases de dados (descritas na Seção 3.6), armazenando os valores de jaccard das partições geradas, o tempo de execução e o número de grupos da partição resultante.

As versões dos algoritmos Bisecting K-means, X-means e OMRk foram executadas com o número de grupos no intervalo entre $k_{min} = 2$ e $k_{max} = \sqrt{N}$. O uso de $k_{max} = \sqrt{N}$ como limite superior é comum na literatura (Pal and Bezdek, 1995; Pakhira et al., 2005; Campello et al., 2009), pois dificilmente o número conhecido de grupos ultrapassa este valor, o que é o caso das bases de dados utilizadas neste trabalho (Seção 3.6).

Todas as versões de X-means utilizam $cut_off_factor = 0.5$, parâmetro que determina a quantidade relativa de grupos a serem divididos caso nenhum divisão dos grupos gere melhora na avaliação. As versões em Ansi e Java possuem o parâmetro $maxIterations$, que corresponde ao número máximo de iterações do processo de divisão que o algoritmo irá realizar (vide Seção 3.4.1). O valor utilizado para este parâmetro é 200, pois este valor é alto o suficiente para permitir que o algoritmo seja finalizado por outros critérios de parada (k_{max} grupos sejam encontrados), uma vez que este parâmetro não foi implementado nas versões em Matlab.

Todos os algoritmos implementados utilizam de alguma forma o K-means em suas execuções. Com o objetivo de permitir uma comparação justa entre estes algoritmos, a mesma implementação do K-means foi utilizada. Nesta implementação, os protótipos dos grupos foram inicializados com os valores de seus atributos sorteados uniformemente no espaço dos dados. Optamos por esta forma de inicialização por obter bons resultados em bases de dados com grupos desbalanceados, o que é o caso de algumas das bases de dados geradas. Contudo, devido as diferenças entre plataformas, o K-means utilizado nas versões do X-means implementada pelo autor em ansi C e disponibilizada no Weka (algoritmos 13 e 14, respectivamente) foi mantido. Devido a estas diferenças, estes algoritmos não serão comparados com os demais em termos de tempo computacional.

Todos o K-means executados possuem o limite máximo de 5 iterações, incluindo os utilizados pelos algoritmos 13 e 14, pois evidência empírica sugere que este valor seja suficiente para que o algoritmo encontre soluções satisfatórias (Anderberg, 1973).

É importante lembrar que para a implementação do X-means em Java foi utilizada a função de dissimilaridade de distância euclidiana implementada pela Weka (*weka.core.EuclideanDistance*), que utiliza normalização dos dados dentro do intervalo $[0, 1]$. O mesmo tipo de normalização é utilizado em todos os outros algoritmos comparados.

Todos os experimentos de algoritmos implementados em Matlab foram executados em um mesmo computador. Suas principais características são: processador Intel Core(TM) 2 E8400 de 3 Ghz, memória ram de 4,0Gb, sistema operacional Ubuntu 8.04 64 bits Kernel Linux 2.6.24-23-generic. A versão utilizada do Matlab foi a 7.6.0 R2008a.

3.8 Resultados

Nesta seção serão apresentados os resultados obtidos por meio da metodologia descrita na Seção 3.7. Os 14 algoritmos enumerados na Seção 3.7 foram executados 30 vezes para cada uma das 324 bases de dados descritas na Seção 3.6. As partições resultantes foram comparadas com os grupos conhecidos por meio do critério externo Jaccard (Seção 2.10.4). Uma vez calculado o Jaccard a média dos valores de Jaccard das 30 partições obtidas nos experimentos, para cada uma das 324 bases e para cada um dos 14 algoritmos estudados,

foram construídos 14 *boxplots*. A Figura 22 representa esses boxplots utilizando linhas em vermelho para destacar as medianas, linhas azuis para delimitar a região entre o primeiro e o terceiro quartil, linhas tracejadas pretas para representar a área entre o primeiro e terceiro quartil para as regiões de outliers e cruzes em vermelho para representar pontos considerados *outliers*.

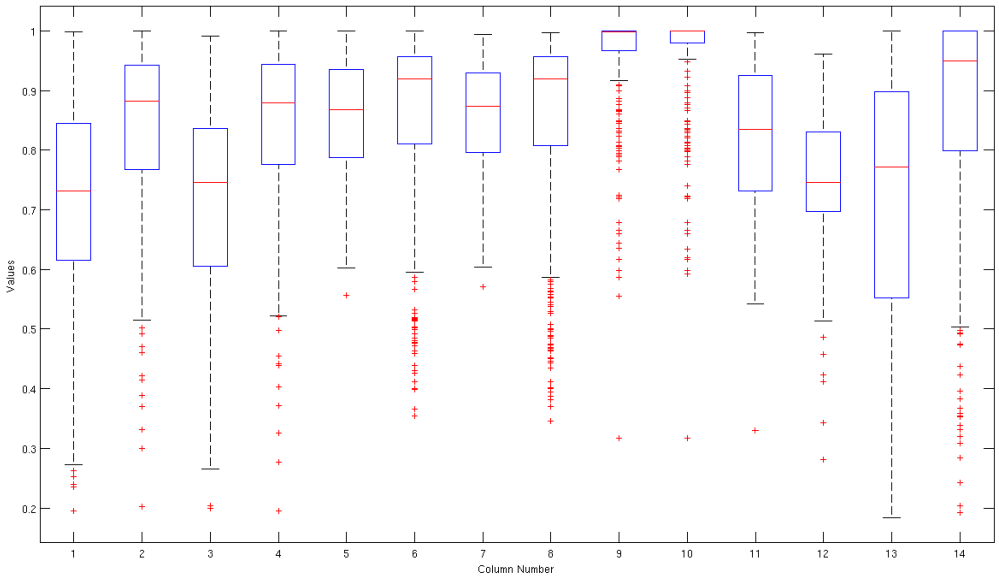


Figura 22: *Boxplots* relativos as médias do índice Jaccard para os algoritmos estudados.

Algumas algumas informações interessantes que podem ser observadas pela analise da na Figura 22:

- A versão do X-means implementada em Java e disponibilizada na Weka, representada como 14, obteve, para mais da metade das bases, valores de Jaccard entre 1 e 0.9 e, para mais de 75% delas, valores superiores a 0.8. O restante das bases resultaram em valores de Jaccard bem distribuídos, com presença de alguns *outliers* com valores até abaixo de 0.5. Isso acontece devido ao K-means implementado nesta versão possuir inicialização dos seus protótipos por meio de medóides, prejudicado ao particionar bases que possuem objetos desbalanceados entre os grupos.

- As versões do Bisecting K-means com silhueta, representadas como 1,2,3 e 4, apresentam grande diferença entre as versões 1 e 3 e as versões 2 e 4. As primeiras apresentam suas medianas abaixo dos primeiros quartis das últimas.

- As versões do Bisecting K-means com BIC, representadas como 5, 6, 7 e 8, geraram partições com valores de Jaccard acima de 0.6 em sua maioria. Todas versões tiveram, em geral, resultados muito parecidos. Contudo duas diferenças podem ser notadas das versões 6 e 8 para as versões 5 e 7 (versões com diferentes critérios de parada). A primeira diferença consiste na maior concentração de bases as quais obtiveram valores entre 0.9 e 1 das versões 6 e 8. A segunda diferença é que as versões 6 e 8 possuem muito mais *outliers* com baixos valores de Jaccard. estas difereças acontecem pelo pelo fato de que as versões 6 e 8 forçam mais a divisão dos grupos, o que gera partições com número de grupos superior ao real.

- As duas versões do OMRK-means (com $n_{\pi,k} = 10$ e 20), representadas como 9 e 10 respectivamente, obtiveram para mais da metade das bases estudadas valores de Jaccard em média próximos a 1 e mais de 75% entre 1 e 0.9. Com exceção de alguns *outliers* concentrados em valores superiores a 0.8, esse resultado se justifica pelo algoritmo efetuar uma busca exaustiva. É possível notar que a versão com $n_{\pi,k} = 20$ possui uma maior concentração de bases para as quais obteve partições com valores de Jaccard próximos a 1.

- A versão do X-means com silhueta, representada pelo número 12, obteve resultado pior que a versão com BIC, representada pelo número 11. Essa afirmação é constada pelo fato do terceiro quartil do algoritmo 12 se posicionar abaixo da mediana do algoritmo 11. Isso acontece pela heurística que utilizamos para a silhueta não ser adequada.

- A versão do X-means implementada em ansi C, representada como 13, é que possui maior distribuição dos valores de Jaccard para as bases estudadas, isso mostra que o algoritmo é pouco robusto para as diferenças entre as bases estudadas, obtendo bons resultados para algumas bases e ruins para outras.

- A versão do X-means implementada em Java e disponibilizada na Weka, representada como 14, obteve para mais da metade dessas bases valores de Jaccard entre 1 e 0.9 e para mais de 75% valores superiores a 0.8. O restante das bases obteve valores de Jaccard bem distribuídos, com presença de alguns outliers com valores até abaixo de 0.5. Isso ocorre devido ao K-means implementado nesta versão possuir inicialização dos seus protótipos por meio de medóides, sendo prejudicado ao particionar as bases que possuem desbalanceamento dos objetos entre os grupos.

Na Tabela 3 são apresentadas as diferenças entre as médias dos Jaccards obtidos pelos algoritmos, por ordem decrescente de valor de Jaccard. Os algoritmos são identificados pelos índices enumerados na Seção 3.7 e suas médias são apresentadas em destaque no vetor que se encontra abaixo da Tabela 3. Por exemplo, a célula da linha 2 e coluna 3 da tabela corresponde a diferença entre as médias das médias dos valores de Jaccard obtidos pelo algoritmo 9 e as médias dos valores de jaccard obtidos pelo algoritmo 14.

Dois testes de hipóteses foram utilizados para determinar se existe diferença estatística entre os valores de Jaccard obtidos pelos algoritmos. O primeiro foi o teste de análise de variância (ANOVA) (Neto, 2002; Walpole et al., 2006). Este teste assume que as amostras sejam obtidas de populações com distribuições normais. Contudo, mesmo que este pré-requisito não seja comprovado, a maioria dos estatísticos não fariam objeções ao uso desse teste (Demšar, 2006), em especial para grandes amostras. Um segundo pré-requisito do ANOVA é que as variáveis aleatórias testadas possuam variâncias semelhantes, o que não é garantido para o caso em estudo. Portanto, o teste não paramétrico de Friedman (Hollander and Wolfe, 1999) também foi utilizado para verificar se as diferenças entre os resultados encontrados possuem significância estatística. Ambos os testes foram aplicados com $\alpha = 5\%$ e as bases de dados foram dividas em blocos, cada um com partições oriundas das 30 execuções feitas para cada base. Uma vez que as hipóteses nulas foram rejeitadas para ambos testes, o que indica que as médias obtidas são significativamente distintas, testes *post-hoc* (Demšar, 2006) foram efetuados para encontrar quais diferenças foram significativas. A comparação das médias quando feita duas-a-duas não é satisfatória porque o nível de significância do teste é desvirtuado, pois quanto maior o número de comparações feitas, maior a probabilidade de se obterem rejeições por mera casualidade (Neto, 2002;

Walpole et al., 2006; Demšar, 2006). Por este motivo, a correção de Bonferroni (Demšar, 2006) é aplicada de modo a permitir o controle do nível de significância do teste, que no caso dos experimentos é de 95%.

No caso em que a diferença entre as médias possuir significância estatística para o teste ANOVA, a posição que contém o valor desta diferença assume colocação cinza claro (e.g. *valor*) na Tabela 3. Caso o teste possuir significância estatística para ambos os testes ANOVA e Friedman, a posição que contém o valor desta diferença assume colocação cinza escuro (e.g. *valor*) na mesma tabela.

Algumas aspectos gerados pela análise da Tabela 3 a serem discutidos são:

- As duas versões do OMRk (com $n_{\pi,k} = 10$ e 20) e representadas pelo número 9 e 10 respectivamente, tiveram os melhores desempenho em relação aos valores de Jaccard obtidos. Não houve diferença estatística entre as médias dos valores de Jaccard das duas versões, o que indica que a maioria das bases estudadas possui uma estrutura simples o suficiente para resultar em uma partição boa em 10 execuções do algoritmo K-means com o número de grupos conhecido.

- Comparando os valores obtidos pelos OMRk com outros algoritmos, o OMRk com $n_{\pi,k} = 20$ obteve valores de Jaccard melhores do que todos os outros algoritmos, confirmados por ambos os testes estatísticos. A versão do OMRk com $n_{\pi,k} = 10$ foi melhor do que todos os outros algoritmos, de acordo com os dois testes estatísticos utilizados, exceto pela versão do algoritmo X-means disponibilizada pelo Weka cuja diferença estatística não foi comprovada pelo teste de Friedman. A busca exaustiva executada por estes algoritmos resultou em uma qualidade superior do que a obtida pelos outros algoritmos.

- A versão do X-means em Java disponibilizada pela Weka e representada por 14 foi o terceiro algoritmo com maior Jaccard médio. Este resultado é estatisticamente comprovado em relação aos algoritmos com médias de Jaccard menores, por ambos os testes utilizados. Esta foi a única versão do X-means que obteve resultados melhores do que as versões do Bisecting K-means estudadas. A versão em C do X-means, representada pelo número 13, obteve um resultado muito abaixo do algoritmo 14, o que intensifica a relevância das diferenças dessas duas versões que foram apresentadas e discutidas neste texto.

- Tomando as versões do Bisecting K-means implementadas com o critério BIC, é possível notar que os algoritmos 5 e 7 possuem melhores resultados que os algoritmos 6 e 8, segundo o teste de Friedman. Isso indica que o critério de parada ii foi melhor do que o critério i no cenário em que a avaliação do desempenho é feita com o critério BIC. Utilizando o critério BIC, o algoritmo costuma encontrar o número de grupos conhecido ou supervalorizar esse valor. Ao utilizar um critério de parada que força a continuação da divisão dos grupos o número de grupos tende a ser supervalorizado, o que prejudica o seu desempenho avaliado pelo critério Jaccard.

- Em relação as versões do Bisecting K-means implementadas com o critério Silhueta, notamos que as algoritmos 2 e 4 possuem melhores resultados que os algoritmos 1 e 3, com relevância estatística para ambos os testes. Isto indica que o critério de parada i foi melhor do que o critério ii no cenário em que a avaliação do desempenho é feita por silhueta. O uso do critério silhueta o algoritmo costuma encontrar o número natural (conhecido) de grupos ou subestimá-lo. Ao utilizar um critério de

	10	**9**	**14**	**7**	**5**	**6**	**8**	**2**	**4**	**11**	**12**	**13**	**3**	**1**
10	0	-0.0038	-0.0886	-0.1007	-0.1045	-0.1094	-0.1125	-0.1213	-0.1213	-0.1412	-0.2070	-0.2370	-0.2438	-0.2489
9	0.0038	0	-0.0848	-0.0969	-0.1008	-0.1057	-0.1087	-0.1175	-0.1176	-0.1374	-0.2033	-0.2332	-0.2400	-0.2452
14	0.0886	0.0848	0	-0.0121	-0.0159	-0.0209	-0.0239	-0.0327	-0.0328	-0.0526	-0.1184	-0.1484	-0.1552	-0.1604
7	0.1007	0.0969	0.0121	0	-0.0038	-0.0087	-0.0118	-0.0206	-0.0206	-0.0405	-0.1063	-0.1363	-0.1431	-0.1482
5	0.1045	0.1008	0.0159	0.0038	0	-0.0049	-0.0080	-0.0168	-0.0168	-0.0366	-0.1025	-0.1325	-0.1393	-0.1444
6	0.1094	0.1057	0.0209	0.0087	0.0049	0	-0.0030	-0.0118	-0.0119	-0.0317	-0.0976	-0.1275	-0.1343	-0.1395
8	0.1125	0.1087	0.0239	0.0118	0.0080	0.0030	0	-0.0088	-0.0088	-0.0287	-0.0945	-0.1245	-0.1313	-0.1365
2	0.1213	0.1175	0.0327	0.0206	0.0168	0.0118	0.0088	0	-0.0000	-0.0199	-0.0857	-0.1157	-0.1225	-0.1277
4	0.1213	0.1176	0.0328	0.0206	0.0168	0.0119	0.0088	0.0000	0	-0.0198	-0.0857	-0.1157	-0.1224	-0.1276
11	0.1412	0.1374	0.0526	0.0405	0.0366	0.0317	0.0287	0.0199	0.0198	0	-0.0659	-0.0958	-0.1026	-0.1078
12	0.2070	0.2033	0.1184	0.1063	0.1025	0.0976	0.0945	0.0857	0.0857	0.0659	0	-0.0300	-0.0368	-0.0419
13	0.2370	0.2332	0.1484	0.1363	0.1325	0.1275	0.1245	0.1157	0.1157	0.0958	0.0300	0	-0.0068	-0.0119
3	0.2438	0.2400	0.1552	0.1431	0.1393	0.1343	0.1313	0.1225	0.1224	0.1026	0.0368	0.0068	0	-0.0052
1	0.2489	0.2452	0.1604	0.1482	0.1444	0.1395	0.1365	0.1277	0.1276	0.1078	0.0419	0.0119	0.0052	0
	0.9600	**0.9562**	**0.8714**	**0.8593**	**0.8555**	**0.8505**	**0.8475**	**0.8387**	**0.8387**	**0.8188**	**0.7530**	**0.7230**	**0.7162**	**0.7110**

Tabela 3: Diferenças entras as médias dos valores de Jaccard obtidos pelos algoritmos estudados.

parada que força a continuação da divisão dos grupos o número de grupos tende a ser aumentado compensando essa subestimação.

- Por último em relação às diferenças entre as versões implementadas do X-means, é possível notar que a versão do X-means com BIC (algoritmo 11) foi melhor que o algoritmo utilizando silhueta, representado por 12, segundo ambos os testes estatísticos utilizados. Isso indica que talvez a heurística para utilização da silhueta não é adequada. Além disso, os algoritmos 11 e 12 obtiveram resultados estatisticamente melhores que a versão em C (de número 13), para ambos os testes no caso do algoritmo 11 e apenas para o teste Friedman no algoritmo 12.

- Os resultados também indicam que, para as bases estudadas, a forma de calcular o diâmetro dos grupos das diferentes versões do Bisecting K-means não resultam em diferença com significancia estatística para nenhum dos testes utilizados. As versões do Bisecting K-means que utilizam o critério de parada ii resultaram em valores de Jaccard praticamente idênticos, diferenciando apenas em relação ao método de calculo do diâmetro. Estas versões só gerariam resultados diferentes se tivessem sua execução interrompida caso o limite superior de número de grupos (k_{max}) for alcançado, o que não ocorreu para as bases estudadas. Apenas a aleatoriedade presente no algoritmo preveniu que os resultados não fossem idênticos.

Além da qualidade das partições obtidas, também foram comparados os tempos computacionais de 12 dos 14 algoritmos estudados, todos implementados em Matlab. As versões do X-means em C e a em Java disponibilizada pela Weka foram excluídas por se tratarem de plataformas de desenvolvimento com características distintas.

Os tempos médios (em segundos) de execução dos algoritmos para cada base de dados são apresentados em forma de *boxplots* na Figura 23. Nesta figura, pode ser observado que as versões do algoritmo OMRk executaram em maior tempo e possuem tempos computacionais médios com maior variância do que os outros algoritmos.

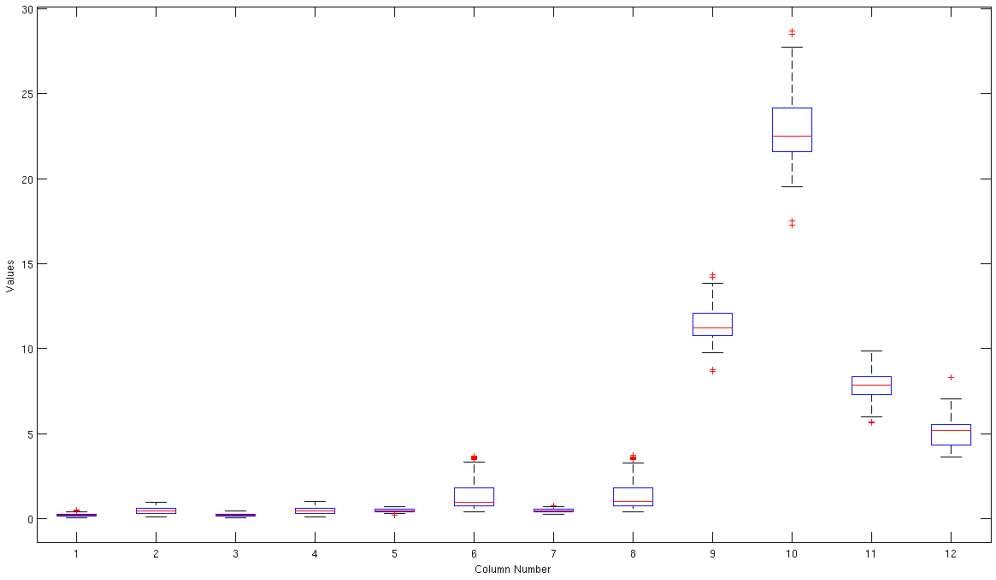


Figura 23: *Boxplots* relativos as médias dos tempos de execução obtidos para os algoritmos estudados.

Depois do OMRk, os algoritmos com os maiores tempos computacionais foram as versões de X-means. As versões do Bissecting K-means resultaram nos menores tempos de execução. Há um destaque para as versões representadas pelos números 6 e 8 que possuem tempos de execução maiores. Essa diferença nos tempos é justificada por essas versões utilizarem o critério ii de parada.

A dimensionalidade dos dados influência diretamente no tempo médio de execução dos algoritmos, uma vez que o tempo de execução do k-means é dependente da dimensão dos dados. Portanto, devido a grande variação da dimensão dos dados, é esperado que haja uma variação razoável da médias de tempos das execuções dos algoritmos para as bases estudadas.

A Tabela 4 apresenta a diferença entre as médias dos tempos computacionais obtidos por todas as execuções dos 12 algoritmos que tiveram seus tempos de execução medidos. Os algoritmos 13 e 14 não são passíveis de participarem desta comparação por serem implementados em condições diferentes. Logo abaixo, estas médias são apresentadas com destaque em um vetor, por ordem decrescente. O mesmo procedimento estatístico utilizado para validar as diferenças presentes na Tabela 3 foi utilizado para validar as diferenças da Tabela 4. Desta forma, as posições pintadas de cinza indicam a existência de significância estatística nas diferenças presentes. Não há casos em que os testes ANOVA e Friedman obtenham resultados distintos para as diferenças de tempo computacional.

Algumas observações relacionadas as diferenças entre os tempos computacionais e a avaliação por Jaccard serão feitas a seguir:

- Como visto anteriormente, os maiores tempos computacionais foram resultantes de execuções do algoritmo OMRk. Apesar de não haver diferença significativa entre as médias dos Jaccard obtidos pelas versões com $n_{\pi,k} = 10$ e $n_{\pi,k} = 20$, houve diferença estatística entre as médias dos tempos obtidos para estes algoritmos. Como era esperado, o algoritmo OMRk executou em aproximadamente o dobro do tempo da versão com $n_{\pi,k} = 10$.

- Depois das versões de OMRk, os algoritmos X-means foram os com maior tempo de execução, sendo que a versão com avaliação por meio de BIC obteve um tempo de execução menor do que a avaliada por silhueta (comprovado estatisticamente). Estes algoritmos também obtiveram valores para critério Jaccard piores do que a maioria das versões de Bisecting K-means.

- Os algortimos 5,6,7 e 8, que correspondem aos Bisecting kmeans com avaliação por BIC, resultam em Jaccards e tempos medianos em relação aos outros algoritmos. Contudo, é possível notar que o uso do critério de parada i obteve melhores Jaccards e tempos computacionais menores.

- Os algoritmos 1 e 3 (Bisecting kmeans com avaliação por silhueta e critério de parada i) foram algoritmos mais rápidos. Contudo também foram os algoritmos com piores resultados em termos de qualidade, segundo o critério Jaccard. Não há diferença estatística entre os resultados obtidos por estes algoritmos.

- As diferentes formas de calcular os diâmetros dos grupos utilizadas no Bisecting K-means não influeciaram no tempo computacional do algoritmo, com significância estatística. Isto pode ser observado pelas células correspondentes aos pares de algoritmos: (1 e 3), (2 e 4), (5 e 7) e (6 e 8). Esse resultado pode estar relacionado ao fato das diferentes formas de calcular o diâmetro terem a mesma complexidade

	10	**9**	**11**	**12**	**8**	**6**	**5**	**7**	**2**	**4**	**3**	**1**
10	0	-11.4817	-15.0948	-17.8915	-21.6069	-21.6311	-22.4533	-22.4549	-22.4629	-22.4652	-22.7215	-22.7253
9	11.4817	0	-3.6131	-6.4097	-10.1252	-10.1493	-10.9716	-10.9732	-10.9812	-10.9834	-11.2397	-11.2436
11	15.0948	3.6131	0	-2.7967	-6.5121	-6.5363	-7.3586	-7.3601	-7.3681	-7.3704	-7.6267	-7.6305
12	17.8915	6.4097	2.7967	0	-3.7155	-3.7396	-4.5619	-4.5635	-4.5715	-4.5737	-4.8300	-4.8339
8	21.6069	10.1252	6.5121	3.7155	0	-0.0241	-0.8464	-0.8480	-0.8560	-0.8583	-1.1146	-1.1184
6	21.6311	10.1493	6.5363	3.7396	0.0241	0	-0.8223	-0.8239	-0.8318	-0.8341	-1.0904	-1.0943
5	22.4533	10.9716	7.3586	4.5619	0.8464	0.8223	0	-0.0016	-0.0096	-0.0118	-0.2681	-0.2720
7	22.4549	10.9732	7.3601	4.5635	0.8480	0.8239	0.0016	0	-0.0080	-0.0103	-0.2666	-0.2704
2	22.4629	10.9812	7.3681	4.5715	0.8560	0.8318	0.0096	0.0080	0	-0.0023	-0.2586	-0.2624
4	22.4652	10.9834	7.3704	4.5737	0.8583	0.8341	0.0118	0.0103	0.0023	0	-0.2563	-0.2601
3	22.7215	11.2397	7.6267	4.8300	1.1146	1.0904	0.2681	0.2666	0.2586	0.2563	0	-0.0038
1	22.7253	11.2436	7.6305	4.8339	1.1184	1.0943	0.2720	0.2704	0.2624	0.2601	0.0038	0
	22.9463	**11.4646**	**7.8515**	**5.0549**	**1.3394**	**1.3153**	**0.4930**	**0.4914**	**0.4834**	**0.4811**	**0.2249**	**0.2210**

Tabela 4: Diferenças entras as médias dos tempos de execução obtidos pelos algoritmos estudados.

computacional e seus resultados não implicarem em diferenças na qualidade das partições geradas. Isto faz com que o algoritmo aplique uma quantidade aproximada de divisões.

3.9 Conclusões

Os experimentos executados permitem concluir que o algoritmo de busca exaustiva OMRk obteve resultados mais próximos da estrutura real do tipo de base estudado do que os outro algoritmos. Contudo, seu tempo computacional é elevado e soluções com qualidades próximas podem ser obtidas por meio do algoritmo Bisecting k-means, em especial se as soluções forem avaliadas com o critério BIC. Outro algoritmo que obteve bom desempenho foi o X-means distribuído juntamente com o programa Weka. Porém não foi possível comparar seu tempo computacional.

A versão implementada do algoritmo X-means é consistente com a descrição do algoritmo feitas no artigo (Pelleg and Moore, 2000) obteve resultados superiores a implementação do algoritmo feita em linguagem ansi C feita pelo autor e utilizada para as comparações no artigo citado, segundo a avaliação feita com valores de Jaccard. Foi comprovado experimentalmente que as diferentes versões do algoritmo (do autor, disponibilizada pelo Weka, com silhueta...) geram resultados com diferenças estatisticamente significantes.

Para o tipo de base de dados estudado, valores modestos para o parâmetro $n_{\pi,k}$ e critérios de parada menos conservadores, como o critério de parada i utilizado pelo Bisecting k-means, foram suficientes para encontrar partições de qualidade equivalente a outros algoritmos com buscas mais demoradas.

Referências

Anderberg, M. R. (1973). *Cluster Analysis for Applications*. Academic Press Inc.

Calinski, R. B. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, 3:1–27.

Campello, R. J. G. B., Hruschka, E. R., and Alves, V. S. (2009). On the efficiency of evolutionary fuzzy clustering. *Journal of Heuristics*, 15(1):43–75. doi: 10.1007/s10732-007-9059-6.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30.

Devaney, M. and Ram, A. (1997). Efficient feature selection in conceptual clustering. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 92–97, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Duda, R., Hart, P., and Stork, D. (2001). *Pattern Classification*. John Wiley & Sons.

Everitt, B., Landau, S., and Leese, M. (2001). *Cluster Analysis*. Arnold Publishers.

Hollander, M. and Wolfe, D. A. (1999). *Nonparametric Statistical Methods*. Wiley-Interscience.

Hruschka, E. R., Campello, R. J. G. B., and de Castro, L. N. (2004a). Evolutionary algorithms for clustering gene-expression data. In *Proc. IEEE Int. Conf. on Data Mining*, pages 403–406, Brighton/England.

Hruschka, E. R., Campello, R. J. G. B., and de Castro, L. N. (2004b). Improving the efficiency of a clustering genetic algorithm. In *Advances in Artificial Intelligence - IBERAMIA 2004: 9th Ibero-American Conference on AI, Puebla, Mexico, November 22-25. Proceedings*, volume 3315, page 861. Springer-Verlag GmbH, Lecture Notes in Computer Science.

Jaccard, P. (1908). Nouvelles recherches sur la distribution florale. *Bull. Soc. Vandoise des Sci. Nat.*, 44:223–270.

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

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323.

Kaufman, L. and Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons.

Milligan, G. W. and Cooper, M. C. (1985). An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50:159–179.

Mirkin, B. (2005). *Clustering for Data Mining: A Data Recovery Approach*. Chapman & Hal.

Naldi, M. C. (2009). Investigação de técnicas de computação evolutiva em problemas de agrupamento de dados. Monografia para exame de qualificação de doutorado - Orientador: Ricardo J. G. B. Campello.

Naldi, M. C., Campello, R. J. G. B., Hruschka, E. R., and Carvalho, A. C. P. L. F. (2009). Efficiency issues of evolutionary k-means. *Applied Soft Computing. Submetido.*

Neto, P. L. O. C. (2002). *Estatística.* Edgard Blucher.

Pakhira, M., Bandyopadhyay, S., and Maulik, U. (2005). A study of some fuzzy cluster validity indices, genetic clustering and application to pixel classification. *Fuzzy Sets Systems*, 155(2):191–214.

Pal, N. and Bezdek, J. (1995). On cluster validity for the fuzzy c-means model. *IEEE Transactions of Fuzzy Systems*, 3(3):370–379.

Pantel, P. A. (2003). *Clustering by Committee.* PhD thesis, Faculty of Graduate Studies and Research − University of Alberta.

Pelleg, D. and Baras, D. (2007). K-means with large and noisy constraint sets. Technical Report 253, IBM RESEARCH. `http://domino.research.ibm.com/library/cyberdig.nsf/1e4115aea78b6e7c85256b360066f0d4/08c39e5a60da9d278525731b00577324?OpenDocument`.

Pelleg, D. and Moore, A. (2000). X-means: extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464.

Steinbach, M., Karypis, G., and Kumar, V. (2000). A comparison of document clustering techniques. Technical Report 34, University of Minnesota.

Vendramin, L., Campello, R. J. G. B., and Hruschka, E. R. (2008). A robust methodology for comparing performances of clustering validity criteria. In *Simpósio Brasileiro de Inteligência Artificial (SBIA)*, volume 5249 of *Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 237–247, Salvador.

Vendramin, L., Campello, R. J. G. B., and Hruschka, E. R. (2009). On the comparison of relative clustering validity criteria. Sparks.

Walpole, R. E., Myers, R., and Myers, S. L. (2006). *Probability & Statistics for Engineers & Scientists.* Macmillan.

Witten, I. H. and Frank, E. (2005). *Data Mining - Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann Publishers, 2nd edition.