Capítulo 4

Análise de agrupamento de dados

A análise de agrupamento de dados é um dos principais problemas exploratórios investigados em mineração de dados. Ao longo das últimas décadas, diversos algoritmos e formulações foram propostos para os mais variados tipos de aplicação. Neste capítulo são apresentadas e discutidas alguns dos principais algoritmos da área, bem como alguns dos critérios para validação das soluções encontradas por eles. Portanto, esta parte está organizada conforme segue. Na Seção 4.1 será descrito o problema de agrupamento de dados. Na Seção 4.2 será discutido ... Na Seção 4.3 serão introduzidos os três algoritmos de agrupamento hierárquico mais conhecidos. Na Seção 4.4 será apresentado o algoritmo *k-means*. Na Seção 4.5 será explicado o algoritmo DBSCAN. Por fim, na Seção 4.7, serão propostos alguns exercícios.

4.1. Aprendizado de máquina não supervisionado

O problema de AM não supervisionado consiste em trabalhar sobre um conjunto de dados, sem utilzar rótulos ou qualquer tipo de informação (ou supervisão) sobre como as instâncias devem ser manipuladas [Zhu e Goldberg 2009]. Em outras palavras, o conjunto de dados tratado consiste apenas de instâncias sem qualquer informação de rótulo (ou classe) conhecida a priori. Algumas das principais tarefas neste cenário são [Zhu e Goldberg 2009]: detecção de novidades, redução de dimensionalidade e agrupamento de dados, sendo esta última de interesse do presente capítulo.

Em suma, considerando um conjunto de dados composto por n objetos descritos por m características, o problema de agrupamento de dados consiste em segmentar esse conjunto em k grupos, com $k << n$, de modo que objetos contidos em um mesmo grupo sejam similares entre si e dissimilares de objetos contidos nos demais grupos, segundo alguma medida de (dis)similaridade que leva em conta as m características disponíveis [Jain e Dubes 1988, Tan et al. 2006].

É importante salientar que não há uma definição universal para o que constitui um grupo [Everitt et al. 2001, Faceli et al. 2011], sendo esta dependente do algoritmo e aplicação estudados. Ademais, diversos algoritmos de agrupamento foram propostos ao longo das últimas décadas, baseadas em diferentes suposições. Por exemplo, o algoritmo *k-means*, amplamente conhecido e utilizado, foi proposto há mais de cinco décadas [Jain 2010].

Portanto, o presente capítulo possui um caráter totalmente introdutório, sem o ob-

jetivo de cobrir de maneira ampla o campo de análise de agrupamento de dados. Assim, apenas os algoritmos clássicos que buscam por partições rígidas nos dados serão discutidos. Formalmente, dado um conjunto $X = \{x_1, \cdots, x_n\}$, uma partição rígida consiste em uma coleção de subconjuntos $C = \{C_1, \cdots, C_k\}$, satisfazendo as seguintes propriedades [Xu e Wunsch 2005]:

- $C_1 \cup C_2 \cup \cdots \cup C_k = X$;
- $C_i \neq \emptyset, \quad \forall i \in [1,k]$; e
- $C_i \cap C_j = \emptyset, \quad \forall i, j \in [1,k]$ e $i \neq j$.

Exercício 4.1. Uma partição rígida, conforme previamente definida, é comumente referenciada na literatura como agrupamento particional exclusivo. Outros tipos de partições encontradas na literatura são: probabilístico, *fuzzy* ou particional não exclusivo. No primeiro caso, cada objeto possuirá uma probabilidade de pertencer a cada grupo com a restrição de que a soma de suas probabilidades é igual a 1. No segundo caso, cada objeto tem um grau de pertinência no intervalo $[0,1]$ para cada grupo. Por fim, no terceiro caso, cada objeto pode ser incluído em mais de um grupo. Escreva, para cada cenário, as respectivas propriedades, de modo similar ao apresentado anteriormente para o caso particional exclusivo.

4.2. Medidas de (dis)similaridade

A definição de uma medida de (dis)similaridade para um problema de agrupamento de dados é de grande importância, uma vez que ela será uma das principais responsáveis por definir a estrutura de grupos produzida. A escolha de uma medida é uma decisão importante e deve levar em conta diversos fatores. Dentre eles, um dos mais importantes envolve os tipos de atributos à disposição (quantitativo, qualitativo nominal, qualitativo ordinal, misto etc.).

Boa parte das medidas de dissimilaridade para objetos com atributos quantitativos são baseadas na distância de Minkowski. Considerando dois objetos x_i e x_j, essa distância é definida como:

$$d(x_i, x_j) = \left(\sum_{l=1}^{m} |x_{il} - x_{jl}|^p \right)^{\frac{1}{p}}. \tag{1}$$

A escolha do valor de p define variações para essa medida. Dentre elas, os três casos mais conhecidos são [Faceli et al. 2011]:

- Distância Manhattan ($p = 1$):

$$d(x_i, x_j) = \sum_{l=1}^{m} |x_{il} - x_{jl}|; \tag{2}$$

- Distância euclidiana ($p = 2$):

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{m} (x_{il} - x_{jl})^2}; \tag{3}$$

- Distância de Chebyshev ($p = \infty$):

$$d(x_i, x_j) = \max_{1 \leq l \leq m} |x_{il} - x_{jl}|. \tag{4}$$

Por sua vez, dentre as principais medidas de similaridade, pode-se citar a correlação de Pearson e o cosseno entre dois vetores [Faceli et al. 2011].

Para atributos qualitativos, uma das medidas de dissimilaridade mais conhecidas é a distância de Hamming, definida como:

$$d(x_i, x_j) = \sum_{l=1}^{m} I(x_{il} = x_{jl}), \tag{5}$$

sendo $I(\cdot)$ a função indicadora, a qual retorna valor igual a um quando a condição passada a ela é verdadeira e zero, caso contrário.

Em Python, diversas medidas de distância podem ser calculadas para um conjunto de dados por meio da função `pdist` do módulo `scipy.spatial.distance`[1]. Um exemplo de código é apresentado a seguir.

```
from scipy.spatial.distance import pdist, squareform
import pandas

# carregando iris.csv sem a coluna que contem
# os rotulos das classes
dados = pandas.read_csv('iris.csv', usecols=[0, 1, 2, 3])

# O metodo 'pdist' recebe como entrada um numpy.array.
# O atributo 'values' de um pandas.DataFrame retorna
# seus valores em tal formato.
dados = dados.values

# 'pdist' calcula as distancias entre todos os pares
# possiveis de objetos. O parametro 'metric' define
# qual medida de (dis)similaridade sera calculada.
distancias = pdist(dados, metric='euclidean')

# O metodo 'pdist' retorna um numpy.array contendo
# n * (n - 1) / 2 elementos. Para transforma-lo em
```

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html

```
# um numpy.array de dimensao n x n pode-se utilizar
# o metodo 'squareform'.
distancias = squareform(distancias)
```

Exercício 4.2. Considerando os três casos da distância de Minkowski apresentados nesta seção ($p = 1$, $p = 2$ e $p = \infty$), comente, para cada um, qual o formato da superfície gerada por todos os possíveis objetos equidistantes a um objeto central.

4.3. Algoritmos hierárquicos

Enquanto algoritmos particionais geram apenas uma partição, algoritmos hierárquicos produzem uma sequência de partições rígidas aninhadas, cada uma contendo uma quantidade diferente de grupos. Esses algoritmos podem seguir duas abordagens [Faceli et al. 2011]:

1. Abordagem aglomerativa: o algoritmo inicia com n grupos, cada um formado por um objeto diferente do conjunto de dados. Em cada passo, os dois grupos mais próximos, segundo algum critério pré-estabelecido, são unidos. Desse modo, o procedimento é repetido até que reste apenas um único grupo contendo todos os objetos.

2. Abordagem divisiva: o algoritmo inicia com apenas um grupo contendo todos os objetos. Em cada passo, algum dos grupos é dividido em dois novos grupos, segundo algum critério pré-estabelecido. Todo o procedimento é realizado até que sejam formados n grupos, cada um contendo apenas um objeto do conjunto de dados original.

Os algoritmos hierárquicos clássicos da literatura são baseados na abordagem aglomerativa e podem ser sumarizados pelo Algoritmo 1.

Algoritmo 1: agrupamento aglomerativo hierárquico.

Entrada: matriz de dissimilaridade $S \in \mathbb{R}^{n \times n}$
Saída : agrupamento hierárquico de S
1 **enquanto** *todos os objetos não estiverem em um único grupo* **faça**
2 atualizar as distâncias entre todos os pares possíveis de grupos;
3 encontrar os dois grupos C_i e C_j mais próximos;
4 unir C_i e C_j em um novo grupo;
5 **fim**

À execução do passo 3 do algoritmo acima dá-se o nome de *linkage* (ligação). A principal diferença entre vários dos algoritmos aglomerativos está no modo como a atualização das distâncias entre os grupos é feita (passo 2). Os métodos mais conhecidos são:

- *Single-linkage*: a distância $d(C_i, C_j)$ entre dois grupos é dada pela distância mínima

entre os seus objetos. Ou seja:

$$d(C_i, C_j) = \min_{\substack{x_a \in C_i \\ x_b \in C_j}} d(x_a, x_b). \tag{6}$$

- *Complete-linkage*: a distância $d(C_i, C_j)$ entre dois grupos é dada pela distância máxima entre os seus objetos. Ou seja:

$$d(C_i, C_j) = \max_{\substack{x_a \in C_i \\ x_b \in C_j}} d(x_a, x_b). \tag{7}$$

- *Average-linkage*: a distância $d(C_i, C_j)$ entre dois grupos é dada pela distância média entre os objetos de diferentes grupos. Ou seja:

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{\substack{x_a \in C_i \\ x_b \in C_j}} d(x_a, x_b). \tag{8}$$

Por fim, uma das principais vantagens de algoritmos de agrupamento hierárquicos advém da representação dos seus resultados por meio de dendrogramas. Um dendrograma é uma representação gráfica em formato de árvore que apresenta a hierarquia de partições obtidas. Na Figura 4.1 é apresentado um exemplo de dendrograma, utilizando o método *complete-linkage*, para o conjunto de dados artificial `blobs.csv`, fornecido como material suplementar, que contém 20 instâncias separadas em três grupos correspondentes a três distribuições normais multivariadas. No eixo horizontal é apresentado o índice de cada objeto. No eixo vertical é apresentado o valor de distância quando cada par de objetos foi unido. As possíveis partições são definidas por meio de cortes no dendrograma. Um exemplo de corte corresponde à linha horizontal pontilhada[2].

Em Python, a biblioteca SciPy fornece as implementações dos três algoritmos de agrupamento hierárquico citados anteriormente, além de uma função para a plotagem de dendrogramas. Um exemplo de código, utilizando o conjunto `blobs.csv` é apresentado a seguir.

```
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import fcluster
from scipy.cluster.hierarchy import dendrogram
from matplotlib import pyplot
import pandas

# carregando blobs.csv sem a coluna 'label'
dados = pandas.read_csv('blobs.csv', usecols=[0, 1])

# O metodo 'linkage' recebe como entrada um numpy.array.
```

[2]Neste exemplo, o corte sugerido segmenta os três grupos corretamente.

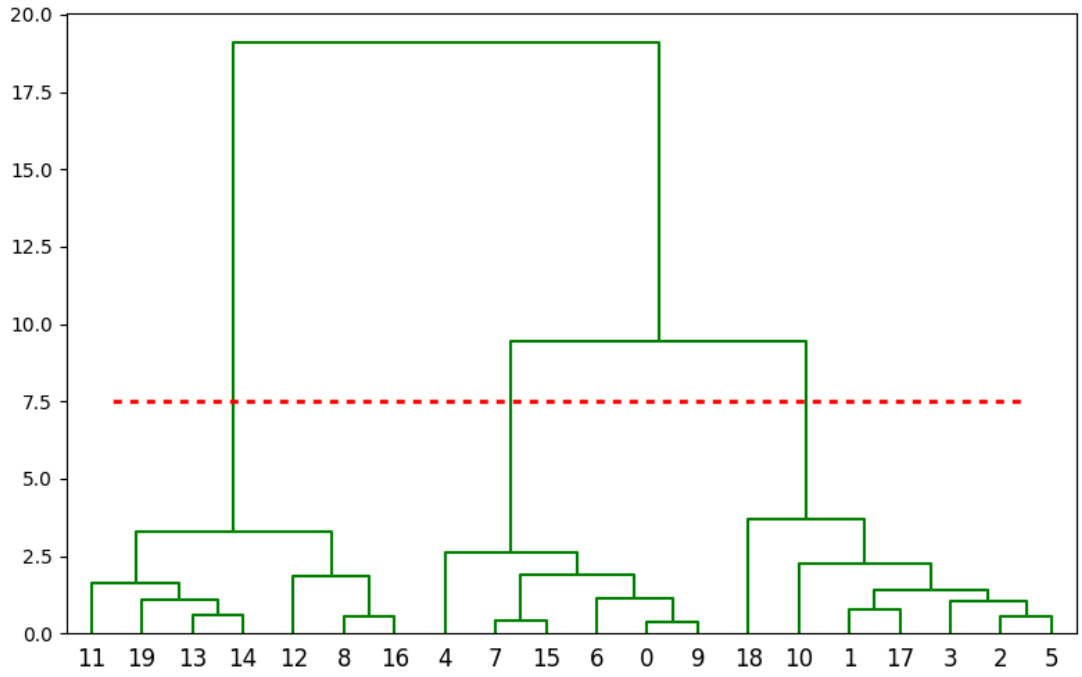

Figura 4.1: Exemplo de dendrograma para um conjunto de dados gerado a partir de três distribuições normais multivariadas.

```
# O atributo 'values' de um pandas.DataFrame retorna
# seus valores em tal formato.
dados = dados.values

# Aplicando o agrupamento hierarquico aos dados.
# O parametro 'method' define qual algoritmo sera
# utilizado.
# Varios metodos de agrupamento estao disponiveis. Para
# os exemplos deste material sera utilizado
# method='average', method='complete' ou
# method='single'.
# O parametro 'metric' define a medida de
# (dis)similaridade a ser utilizada. Para uma lista
# de medidas disponiveis recomenda-se a documentacao
# do metodo scipy.spatial.distance.pdist.
h = linkage(dados, method='complete', metric='euclidean')

# Para plotar o dendrograma utiliza-se o metodo
# 'dendrogram'.
dendrogram(h)
pyplot.show()
```

```
# O metodo fcluster recebe como entrada uma
# hierarquia gerada pelo metodo linkage e extrai
# grupos da mesma segundo algum criterio.
# Abaixo serao extraidos os grupos por meio de um limiar
# de distancia. Segundo o dendrograma da Figura
# 4.1, o qual foi gerado atraves do metodo
# complete-linkage, um limiar de 7.5 parece ser
# adequado.
# A variavel 'rotulos' sera um numpy.array
# onde o valor contido em rotulos[i] indica o rotulo
# do objeto i.
rotulos_dist = fcluster(h, t=7.5, criterion='distance')

# Caso o criterio escolhido seja o numero de
# grupos, o metodo fcluster estima sozinho um valor
# de distancia de modo que t grupos sejam formados.
# Por exemplo, para extrair 3 grupos:
rotulos_k = fcluster(h, t=3, criterion='maxclust')
```

Exercício 4.3. Considerando um conjunto de dados $X = \{x_1, \cdots, x_5\}$ e a matriz de distâncias

$$\begin{bmatrix} 0 & 3 & 10 & 1 & 4 \\ 3 & 0 & 2 & 2 & 3 \\ 10 & 2 & 0 & 8 & 8 \\ 1 & 2 & 8 & 0 & 8 \\ 4 & 3 & 8 & 8 & 0 \end{bmatrix}$$

Nessa matriz, cada elemento a_{ij} indica a distância entre x_i e x_j. Aplique à matriz os três algoritmos hierárquicos descritos nesta seção e descreva, de maneira detalhada, os cálculos realizados por cada um deles em cada passo.

4.4. Algoritmo *k-means*

O algoritmo *k-means* busca por uma partição que minimize a soma dos erros quadráticos (SSE, do inglês, *sum of squared errors*) entre os objetos de um conjunto de dados e o centróide dos seus respectivos grupos [Jain 2010]. A medida SSE é definida como:

$$\text{SSE} = \sum_{i=1}^{k} \sum_{x_j \in C_i} d(x_j, \bar{x}_{C_i})^2, \tag{9}$$

sendo $d(\cdot,\cdot)$ a distância euclidiana e $\bar{x}_{C_i}$ o centróide de um grupo C_i, calculado como:

$$\bar{x}_{C_i} = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j. \tag{10}$$

Em [Drineas et al. 2004] é provado que o problema de minimização da SSE é NP-difícil, inclusive quando $k = 2$. Portanto, o algoritmo *k-means* é um procedimento de

otimização com inicialização aleatória que garante a convergência para um mínimo local da Equação (9). O mesmo é apresentado no Algoritmo 2.

Algoritmo 2: *k-means*.

Entrada: conjunto de dados $X \in \mathbb{R}^{n \times m}$ e o número de grupos k
Saída : agrupamento particional de X em k grupos

1 gerar k centróides aleatoriamente;
2 **repita**
3 calcular a distância entre cada objeto x_j e cada centróide $\bar{x}_{C_i}$;
4 atribuir cada objeto x_j ao grupo C_i com centróide mais próximo;
5 recalcular o centróide de cada grupo conforme a Equação (10);
6 **até** *que um critério pré-definido seja atingido ou que os objetos não mudem de grupo*;

Um exemplo da execução do *k-means* para o conjunto `blobs.csv` é apresentado abaixo.

```
from sklearn.cluster import k_means
import pandas

# carregando blobs.csv sem a coluna 'label'
dados = pandas.read_csv('blobs.csv', usecols=[0, 1])

# O metodo 'k_means' recebe como entrada um numpy.array.
# O atributo 'values' de um pandas.DataFrame retorna
# seus valores em tal formato.
dados = dados.values

# Executa o algoritmo k-means.
# 'n_clusters' indica o numero de grupos buscados.
# 'init' indica o tipo de inicializacao.
# 'n_init' indica a quantidade de vezes que o algoritmo
# sera executado. Dentre todas as 'n_init' execucoes,
# eh retornada aquela com o menor valor de sse.
# O metodo retorna tres valores: o primeiro, um
# um numpy.array com k linhas e m colunas,
# contendo os centroides finais; o segundo, um
# numpy.array contendo os rotulos de cada objeto; e,
# por fim, o valor de sse da solucao retornada.
centroides, rotulos, sse = k_means(dados,
                                   n_clusters=3,
                                   init='random',
                                   n_init=100)
```

Exercício 4.4. Considerando a função objetivo do *k-means*, apresentada na Equação (9):

a Discuta quais serão as características dos grupos encontrados por esse algoritmo.

b Descreva em quais cenários o *k-means* não é capaz de produzir agrupamentos de boa qualidade.

c Um dos maiores problemas do algoritmo *k-means* decorre da influência de *outliers* em sua performance. Em tais casos, o valor de SSE sofrerá um grande incremento e os centróides finais não serão tão representativos quanto seriam com a ausência dos *outliers* [Tan et al. 2006]. Uma alternativa para isso seria utilizar o algoritmo *k-medians*, o qual substitui a média pela mediana no cálculo de um centróide[3]. Embora resultados mais robustos podem ser encontrados, o *k-medians* possui uma grande desvantagem em relação ao *k-means*. Enuncie e discuta tal desvantagem.

4.5. Algoritmo DBSCAN

O algoritmo DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) busca por grupos definidos como regiões com alta densidade de objetos, separados por regiões de baixa densidade [Tan et al. 2006]. Uma das principais vantagens desse algoritmo advém do fato de não ser necessário informar previamente o número desejado de grupos. Para isso, ele se baseia na classificação de cada objeto do conjunto de dados em uma dentre 3 categorias [Ester et al. 1996]:

- **Objeto central:** todo objeto x_i contendo uma quantidade de objetos vizinhos, contando ele próprio, maior ou igual a um parâmetro *MinPts*. Um vizinho é determinado como todo objeto separado por, no máximo, uma distância ε de x_i.

- **Objeto de borda:** todo objeto que não satisfaz as condições para objeto central, mas que pertence à vizinhança de um objeto central.

- **Ruído:** todo objeto que não pertence a nenhuma das duas categorias anteriores.

A Figura 4.2 ilustra um exemplo da classificação de um conjunto de objetos considerando $MinPts = 3$. Nela, os objetos vermelhos são centrais, os amarelos de borda e o azul é ruído. Os círculos em torno de cada objeto denotam o raio ε que define as vizinhanças. O Algoritmo 3 apresenta o pseudocódigo do algoritmo DBSCAN [Tan et al. 2006].

[3]É provado que este novo algoritmo minimiza a distância L_1, também conhecida como distância de Manhattan.

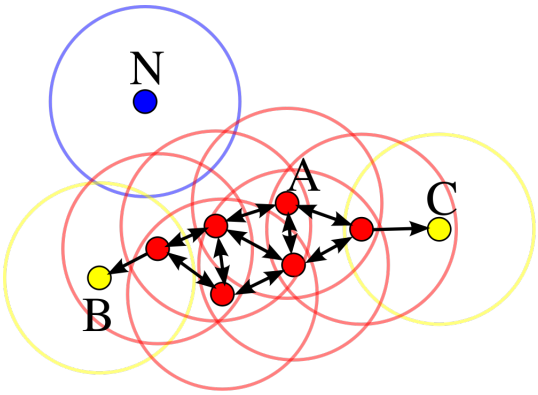

Figura 4.2: Exemplo de classificação de objetos feita pelo DBSCAN com $MinPts = 3$ (Fonte: [Chire 2011][4]).

Algoritmo 3: DBSCAN.

Entrada: conjunto de dados $X \in \mathbb{R}^{n \times m}$, um raio ε e a quantidade mínima de vizinhos para um objeto ser considerado central *MinPts*

Saída : agrupamento particional de X

1 rotular cada objeto como central, borda ou ruído;
2 remover objetos rotulados como ruído;
3 ligar todos os objetos rotulados como centrais que estejam dentro de um raio ε uns dos outros;
4 definir cada componente de objetos centrais conectados como um grupo;
5 inserir cada objeto rotulado como borda ao grupo de algum dos seus objetos centrais associados, resolvendo empates que venham a ocorrer;

Por fim, um exemplo em Python da aplicação do algoritmo DBSCAN a um conjunto de dados é apresentado no código a seguir.

```
from sklearn.cluster import dbscan
import pandas

# carregando blobs.csv sem a coluna 'label'
dados = pandas.read_csv('blobs.csv', usecols=[0, 1])

# O metodo 'dbscan' recebe como entrada um numpy.array.
# O atributo 'values' de um pandas.DataFrame retorna
# seus valores em tal formato.
dados = dados.values

# Executa o algoritmo DBSCAN.
# 'eps' eh o parametro que define o raio de cada objeto.
# 'min_samples' indica a quantidade minima de objetos
# para considerar um objeto como central.
```

[4]A imagem original está disponível sob a licença CC-BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/).

```
# 'metric' define a medida de distancia a ser utilizada
# e, seu valor padrao consiste na distancia euclidiana.
# O metodo retorna dois valores:
# o primeiro eh um numpy.array contendo os indices
# dos objetos classificados como centrais;
# o segundo eh um numpy.array contendo o rotulo de
# grupo de cada objeto
objetos_centrais, rotulos = dbscan(dados,
                                   eps=1.5,
                                   min_samples=3)
```

Exercício 4.5. Uma das principais desvantagens do DBSCAN decorre de o mesmo não ser capaz de identificar corretamente um agrupamento quando as densidades variam amplamente de grupo para grupo [Tan et al. 2006]. Por que isso acontece? Por que não é possível contornar tal problema ao aumentar o valor de ε ou alterar o valor de *MinPts*?

4.6. Validação de agrupamentos

Para avaliar quão bons são os agrupamentos encontrados, uma medida de avaliação ou validação deve ser utilizada. As medidas ou critérios de validação podem ser internos, externos ou relativos. A seguir são apresentadas os principais critérios de validação utilizados para avaliar agrupamentos gerados por algoritmos de agrupamento de dados.

4.6.1. Critérios de validação relativa

"Qual dentre um conjunto de soluções de agrupamento melhor representa os dados?". Esta é a pergunta que deve ser respondida por um critério de validação relativa de uma maneira quantitativa [Jain e Dubes 1988]. Entretanto, é importante observar, conforme será apresentado a seguir, que, assim como algoritmos de agrupamento, diferentes índices de validação relativa possuem diferentes suposições e viéses. Desse modo, cabe aos envolvidos no processo de mineração de dados optarem por aqueles mais adequados na etapa de validação.

4.6.1.1. Índice de Dunn (ID)

O ID é um critério de validação relativa baseado na ideia de compactação intra-grupo e separação inter-grupos [Vendramin et al. 2010]. Em outras palavras, ele é apropriado para identificar agrupamentos que contém grupos cujos objetos estão próximos entre si e distantes de objetos contidos em outros grupos. O ID pode ser formalmente definido como:

$$\text{Dunn}(C) = \min_{\substack{1 \leq i,j \leq k \\ i \neq j}} \left\{ \frac{d(C_i, C_j)}{\max\limits_{1 \leq l \leq k} \{D(C_l)\}} \right\}, \tag{11}$$

sendo $d(C_i, C_j)$ e $D(C_l)$ definidos, respectivamente:

$$d(C_i, C_j) = \min_{\substack{x_a \in C_i \\ x_b \in C_j}} d(x_a, x_b), \tag{12}$$

$$D(C_l) = \max_{x_a, x_b \in C_l} d(x_a, x_b). \tag{13}$$

Portanto, valores altos dessa medida indicam soluções que obedecem seu viés. Ademais, ao generalizar os cálculos de $d(C_i, C_j)$ e $D(C_l)$, variantes dessa medida podem ser geradas [Vendramin et al. 2010]. Dentre elas, 17 são descritas em [Bezdek e Pal 1998].

A implementação do ID não está disponível nas bibliotecas utilizadas neste material. Ela será exigida como exercício na Seção 4.7.

4.6.1.2. Largura de silhueta (LS)

Assim como o ID, a LS baseia-se nos conceitos de compactação intra-grupo e separação inter-grupos. A silhueta de um objeto individual x_i pertencente a um grupo C_l ($S(x_i)$) é definida por:

$$S(x_i) = \frac{b(x_i) - a(x_i)}{\max\{a(x_i), b(x_i)\}} \tag{14}$$

sendo $a(x_i)$ e $b(x_i)$ calculados como:

$$a(x_i) = \frac{1}{|C_l| - 1} \sum_{x_j \in C_l} d(x_i, x_j), \tag{15}$$

$$b(x_i) = \min_{\substack{1 \leq h \leq k \\ h \neq l}} \left\{ \frac{1}{|C_h|} \sum_{x_j \in C_h} d(x_i, x_j) \right\}. \tag{16}$$

Com isso, a LS é definida como a silhueta média entre todos os objetos do conjunto de dados. Ou seja:

$$\mathrm{LS}(C) = \frac{1}{n} \sum_{i=1}^{n} S(x_i). \tag{17}$$

A LS assume valores no intervalo $[-1, 1]$. A melhor partição possível segundo esse critério é aquela que atinge $LS(C) = 1$.

Um exemplo da LS em Python é apresentado a seguir.

```
from sklearn.cluster import k_means
from sklearn.metrics import silhouette_score
import pandas

# carregando blobs.csv sem a coluna 'label'
dados = pandas.read_csv('blobs.csv', usecols=[0, 1])
dados = dados.values

# Executando o algoritmo k-means.
centroides, rotulos, sse = k_means(dados,
```

```
                                        n_clusters=3,
                                        init='random',
                                        n_init=100)

# Calculando a largura de silhueta.
# O primeiro parametro eh o conjunto de dados estudado.
# O segundo engloba os rotulos encontrados por um algoritmo.
# O parametro 'metric' indica a medida de distancia
# utilizada. No caso do algoritmo k-means, sera informado
# 'sqeuclidean' que eh a distancia euclidiana ao
# quadrado.
s = silhouette_score(dados, rotulos, metric='sqeuclidean')
```

Exercício 4.6.1. Os dois índices relativos apresentados nesta parte não são adequados para validar o algoritmo DBSCAN. Por que?

Para auxiliar na resposta, aplique o algoritmo DBSCAN no conjunto de dados `moons.csv`, fornecido como material suplementar (observe que pode ser necessário ajustar o valor do parâmetro ε para se obter um bom resultado). Gere um *scatter plot* para a base de dados com cores para seus rótulos originais e um *scatter plot* com cores para os rótulos encontrados pelo DBSCAN. Calcule o valor o ID e o LS para a solução gerada pelo DBSCAN. Compare os *scatter plots* com os valores dos índices.

4.6.2. Critérios de validação interna

Índices de validação interna medem o grau em que uma solução de agrupamento é justificada com base apenas no conjunto de dados original ou em uma matriz de similaridades ou dissimilaridades calculadas a partir do mesmo [Jain e Dubes 1988]. Assim, um índice de validação interna pode ser visto como o grau de concordância entre um agrupamento encontrado por um algoritmo de agrupamento e o próprio conjunto de dados.

Muitas vezes, índices de validação interna são utilizados como função objetivo a ser otimizada por algoritmos de agrupamento. Como exemplo, tem-se a relação entre o SSE (Equação 9) e o *k-means*.

Exercício 4.6.2. Em muitas aplicações, o SSE é utilizado como critério de qualidade na seleção de uma dentre várias soluções de agrupamento disponíveis. Desse modo, aquela com valor mínimo de SSE é normalmente escolhida. Com base nisso, responda:

a Para quais formatos geométricos de grupos existentes em um conjunto de dados esse critério é adequado?

b O SSE normalmente não é adequado para escolher uma dentre várias soluções com diferentes números de grupos. Por que? (Dica: aplique o *k-means* em diversos conjuntos de dados, por exemplo `blobs.csv` e Iris, considerando vários valores para o números de grupos e observe o comportamento do SSE).

4.6.3. Critérios de validação externa

Critérios de validação externa avaliam o grau de concordância entre duas soluções de agrupamento de dados [Jain e Dubes 1988]. Em várias aplicações práticas, uma das partições comparadas consistirá em uma solução obtida por algum algoritmo, denotada por $C^{ob} = \{C_1^{ob}, \cdots, C_k^{ob}\}$, enquanto que a partição restante representará uma solução de referência para o conjunto de dados estudado, denotada por $C^{ref} = \{C_1^{ref}, \cdots, C_q^{ref}\}$.

Vários dos critérios de similaridade para a comparação de partições baseiam-se na contagem de pares de objetos. Assim, nesse tipo de abordagem, as seguintes variáveis são definidas:

- a_{11}: quantidade de pares de objetos em um mesmo grupo em C^{ob} e C^{ref};
- a_{10}: quantidade de pares de objetos em um mesmo grupo em C^{ob} mas em grupos diferentes em C^{ref};
- a_{01}: quantidade de pares de objetos em grupos diferentes em C^{ob} mas em um mesmo grupo em C^{ref};
- a_{00}: quantidade de pares de objetos pertencentes a grupos diferentes tanto em C^{ob} quanto em C^{ref}.

A seguir, cinco dos índices de validação externa mais conhecidos para avaliação de agrupamentos particionais exclusivos serão apresentados. O respectivo código será apresentado ao final desta seção.

4.6.3.1. Índice Rand (IR)

O critério IR calcula a proporção de acordos no agrupamento de pares de objetos entre duas partições (a_{11} e a_{00}) em relação ao total de pares possíveis de objetos. Ou seja:

$$\mathrm{IR}(C^{ob}, C^{ref}) = \frac{a_{11} + a_{00}}{a_{11} + a_{10} + a_{01} + a_{11}} = \frac{a_{11} + a_{00}}{\binom{n}{2}}. \tag{18}$$

Esta medida retorna valores no intervalo $[0, 1]$, com valores mais altos indicando uma maior similaridade entre duas partições.

4.6.3.2. Índice Jaccard (IJ)

O critério IJ calcula a proporção de pares de objetos agrupados conjuntamente em C^{ob} e C^{ref} em relação à quantidade de pares de objetos em um mesmo grupo em C^{ob} ou em C^{ref}. Ou seja:

$$\mathrm{IJ}(C^{ob}, C^{ref}) = \frac{a_{11}}{a_{11} + a_{01} + a_{10}}. \tag{19}$$

Assim como o IR, o IJ está contido no intervalo $[0,1]$ com valores mais altos apontando uma maior concordância entre C^{ob} e C^{ref}.

4.6.3.3. Índice Rand Ajustado (IRA)

Um dos problemas do IR tradicional advém da dificuldade em determinar "quão bom"ou "quão alto"é um valor obtido na comparação de duas partições. Essa dificuldade existe pois, ao comparar duas partições geradas aleatoriamente, valores inesperadamente altos podem ocorrer. Em tais situações, torna-se necessário *ajustar* o índice para aleatoriadade. Desse modo, a versão ajustada do IR obedece a definição a seguir:

$$\text{IRA}(C^{ob},C^{ref}) = \frac{\text{IR}(C^{ob},C^{ref}) - E[\text{IR}(C^{ob},C^{ref})]}{\max\{IR\} - E[\text{IR}(C^{ob},C^{ref})]}, \tag{20}$$

onde $E[\text{IR}(C^{ob},C^{ref})]$ indica o valor esperado do IR ao comparar as partições C^{ob} e C^{ref} e $\max\{IR\}$ indica o valor máximo atingido por essa medida (ou seja, $\max\{IR\} = 1$).

Critérios de validação externa, quando ajustados para aleatoriedade, assumem valores no intervalo $(-\infty, 1]$. Assim, valores positivos indicam que a similaridade entre C^{ob} e C^{ref} é maior do que o valor esperado ao comparar agrupamentos gerados aleatoriamente [Horta e Campello 2015]. Em [Hubert e Arabie 1985], o IRA é proposto assumindo uma distribuição hipergeométrica como modelo nulo. O cálculo do IRA é dado por:

$$\text{IRA}(C^{ob},C^{ref}) = \frac{a - \frac{(a+c)(a+b)}{a+b+c+d}}{\frac{(a+c)+(a+b)}{2} - \frac{(a+c)(a+b)}{a+b+c+d}}. \tag{21}$$

4.6.3.4. Informação Mútua (IM)

A medida IM mede a informação compartilhada entre C^{ob} e C^{ref}. Em outras palavras, essa medida quantifica a quantidade de informação sobre C^{ref} obtida através de C^{ob} e vice-versa. A IM é calculada como:

$$\text{IM}(C^{ob},C^{ref}) = \sum_{i=1}^{|C^{ob}|}\sum_{j=1}^{|C^{ref}|} P(i,j)\ \log\left(\frac{P(i,j)}{P_{ob}(i)\ P_{ref}(j)}\right) \tag{22}$$

sendo $P(i,j)$, $P_{ob}(i)$ e $P_{ref}(j)$ definidos, respectivamente, por:

$$P(i,j) = \frac{|C_i^{ob} \cap C_j^{ref}|}{n}, \tag{23}$$

$$P_{ob}(i) = \frac{|C_i^{ob}|}{n}, \tag{24}$$

$$P_{ref}(j) = \frac{|C_j^{ref}|}{n}. \tag{25}$$

O valor mínimo para a IM é igual a zero, sendo que valores mais altos indicam uma melhor concordância entre as partições comparadas. Uma das principais desvatangens dessa medida é a ausência de um limitante superior. Para fins comparativos, uma versão normalizada da mesma para o intervalo $[0,1]$ é mais adequada [Strehl e Ghosh 2002]. Uma das variações mais conhecidas (IMN) é apresentada na próxima subseção.

4.6.3.5. Informação Mútua Normalizada (IMN)

Uma das normalizações mais conhecidas da IM pode ser definida como:

$$\text{IMN}(C^{ob}, C^{ref}) = \frac{\text{IM}(C^{ob}, C^{ref})}{\sqrt{H(C^{ob})\, H(C^{ref})}}, \tag{26}$$

onde $H(\cdot)$ representa a entropia de um agrupamento, calculada por meio da fórmula a seguir:

$$H(C) = -\sum_{i=1}^{|C|} P(i)\, \log(P(i)). \tag{27}$$

Desse modo, a IMN está contida no intervalo $[0,1]$, com valores mais altos apontando uma maior similaridade entre as partições comparadas.

4.6.3.6. Exemplos

A seguir é apresentado um exemplo de código em Python para o cálculo do IRA, IM e IMN. Os critérios IR e o IJ não estão disponíveis nas bibliotecas utilizadas neste material. Portanto, as respectivas implementações serão exigidas como exercício na Seção 4.7.

```
from sklearn.cluster import k_means
from sklearn.metrics import adjusted_rand_score
from sklearn.metrics import mutual_info_score
from sklearn.metrics import normalized_mutual_info_score
import pandas

# carregando blobs.csv sem a coluna 'label'
dados = pandas.read_csv('blobs.csv', usecols=[0, 1])
dados = dados.values

# Executando o algoritmo k-means.
centroides, rotulos_kmeans, sse = k_means(dados,
                                          n_clusters=3,
                                          init='random',
                                          n_init=100)

# Calculando o IRA.
ira = adjusted_rand_score(rotulos_dados, rotulos_kmeans)
```

```
# Calculando o IM.
im = mutual_info_score(rotulos_dados, rotulos_kmeans)

# Calculando o IMN.
imn = normalized_mutual_info_score(rotulos_dados,
                                   rotulos_kmeans)
```

Exercício 4.6.3. O IR possui um forte viés para a comparação de partições contendo maiores números de grupos. Para tal caso, um dos termos irá dominar o valor calculado para a medida. Responda:

a Qual é o termo que domina o valor do IR? Por que isso acontece?

b Em qual cenário o valor de IR será igual a zero?

4.7. Exercícios

1. Uma extensão direta do algoritmo *k-means*, capaz de gerar, divisivamente, uma hierarquia de grupos, é conhecida por *bisecting k-means*. Tal extensão é descrita na Seção 8.2.3 do Capítulo 8 do livro de [Tan et al. 2006][5]. Implemente o *bisecting k-means*, de modo que retorne a hierarquia de grupos produzida, e aplique-o ao conjunto de dados Iris.

2. Implemente o ID, o IR e o IJ. Aplique o *k-means* para os conjuntos `blobs.csv`, `moons.csv` e Iris. Calcule os valores dos índices para os resultados do k-means.

Referências

[Bezdek e Pal 1998] Bezdek, J. C. e Pal, N. R. (1998). Some new indexes of cluster validity. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(3):301–315.

[Chire 2011] Chire (2011). DBSCAN Illustration. `https://commons.wikimedia.org/wiki/File:DBSCAN-Illustration.svg`. Acesso em: 05 set. 2017.

[Drineas et al. 2004] Drineas, P., Frieze, A., Kannan, R., Vempala, S., e Vinay, V. (2004). Clustering large graphs via the singular value decomposition. *Machine learning*, 56(1):9–33.

[Ester et al. 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[5]Este capítulo está disponível gratuitamente em `http://www-users.cs.umn.edu/~kumar/dmbook/ch8.pdf`.

[Everitt et al. 2001] Everitt, B., Landau, S., Leese, M., e Stahl, D. (2001). Cluster analysis. 2001. *Arnold, London*.

[Faceli et al. 2011] Faceli, K., Lorena, A. C., Gama, J., e Carvalho, A. C. P. L. F. (2011). Inteligência artificial: Uma abordagem de aprendizado de máquina. *Rio de Janeiro: LTC*, 2:192.

[Horta e Campello 2015] Horta, D. e Campello, R. J. G. B. (2015). Comparing hard and overlapping clusterings. *Journal of Machine Learning Research*, 16:2949–2997.

[Hubert e Arabie 1985] Hubert, L. e Arabie, P. (1985). Comparing partitions. *Journal of classification*, 2(1):193–218.

[Jain 2010] Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666.

[Jain e Dubes 1988] Jain, A. K. e Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc.

[Strehl e Ghosh 2002] Strehl, A. e Ghosh, J. (2002). Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617.

[Tan et al. 2006] Tan, P.-N. et al. (2006). *Introduction to data mining*. Pearson Education India.

[Vendramin et al. 2010] Vendramin, L., Campello, R. J., e Hruschka, E. R. (2010). Relative clustering validity criteria: A comparative overview. *Statistical analysis and data mining: the ASA data science journal*, 3(4):209–235.

[Xu e Wunsch 2005] Xu, R. e Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678.

[Zhu e Goldberg 2009] Zhu, X. e Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130.