

Agrupamento de dados particional

*Aprendizado não
supervisionado*

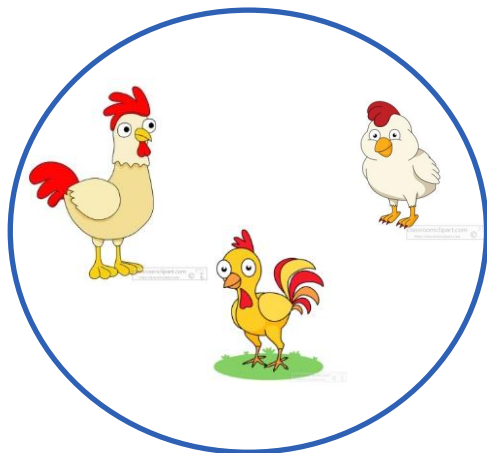
Heloisa de Arruda Camargo



INFORMAÇÃO,
TECNOLOGIA
& INOVAÇÃO

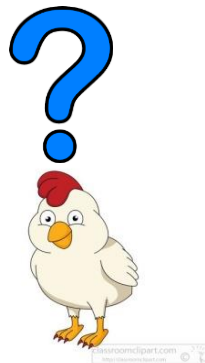
Agrupamento de dados

Objetivo: agrupar objetos em *clusters* (grupos) de modo que objetos pertencentes a um mesmo *cluster* são *mais similares* entre si de acordo com alguma *medida de similaridade* pré-definida, enquanto que objetos pertencentes a clusters diferentes têm uma *similaridade menor*.



Agrupamento de dados – Discussão geral

O que é similaridade?



Medidas de proximidade

- São medidas de dissimilaridade ou similaridade
- Devem ser escolhidas de acordo com o tipo e escala dos dados

- Entre objetos com atributos contínuos:

- Medidas de distância (dissimilaridade)
- Medidas de correlação (similaridade)
 - Correlação de Pearson
 - Medida de cosseno

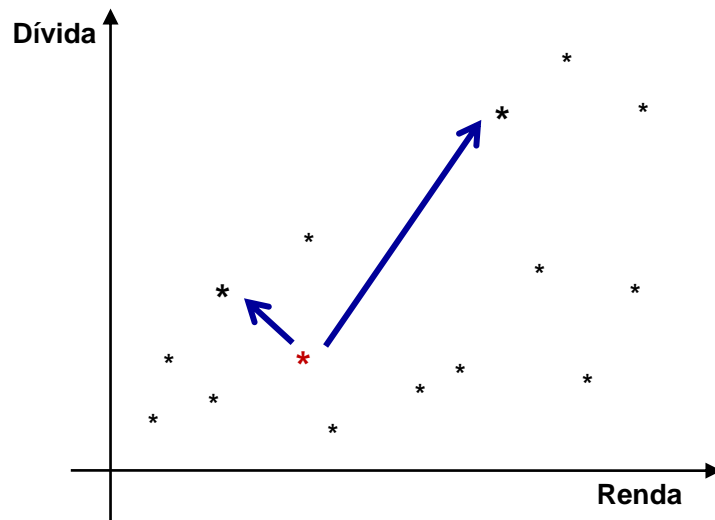
- Entre objetos com atributos discretos:

- Coeficiente de casamento simples (CCS) (similaridade)
- Coeficiente de Jaccard (similaridade)

(São muitas as medidas definidas na literatura)

Medidas de distância

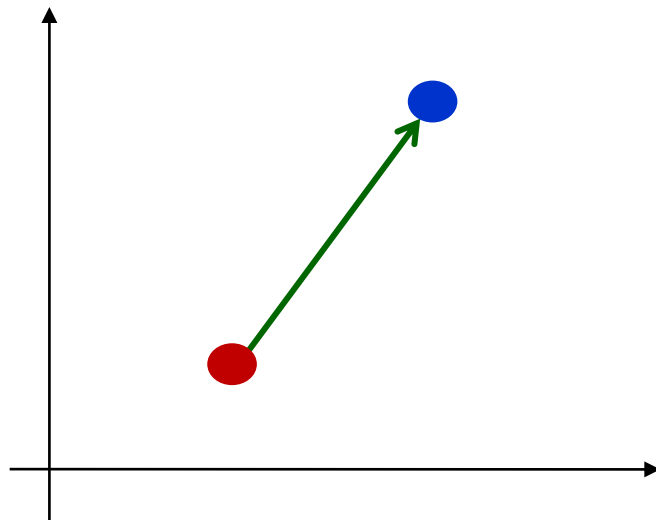
- Medida de **dissimilaridade** para atributos contínuos
- atributos dos exemplos são considerados como **dimensões** de um espaço multidimensional
- cada exemplo corresponde a um **ponto** no espaço
- **similaridade** entre dois pontos é inversamente proporcional a distância entre eles



Medidas de distância

- Dissimilaridade para variáveis contínuas
- **Distância Euclidiana**

$$d(x, y) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$



Medidas de distância

- Dissimilaridade para variáveis contínuas

- **Distância Euclidiana**

$$d(x, y) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$

	potência	Peso	aceleração
E ₁	130	3504	12
E ₂	165	3693	11,5
E ₃	150	3436	11

$$d(E_1, E_2) = \sqrt{\sum_{i=1}^m (E_1^i - E_2^i)^2} = \sqrt{(130 - 165)^2 + (3504 - 3693)^2 + (12 - 11,5)^2} = 192,2141$$

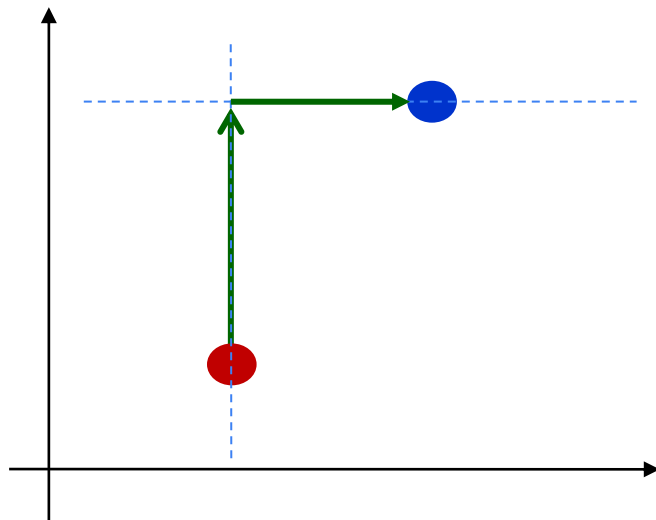
Medidas de distância

- Dissimilaridade para variáveis contínuas
- **Distância Euclidiana**
 - Atributos com maior escala e maior variabilidade têm maior peso no cálculo da distância
 - Implementações eficientes usam distância quadrática: $d(x, y)^2$
 - Tendem a induzir clusters hiper esféricos

Medidas de distância

- Dissimilaridade para variáveis contínuas
- **Distância de Manhattan (City block, Taxicab)**

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m |x_i - y_i|$$



Medidas de distância

- Dissimilaridade para variáveis contínuas

- **Distância de Manhattan (City block, Taxicab)**

$$d(x, y) = \sum_{i=1}^m |x_i - y_i|^2$$

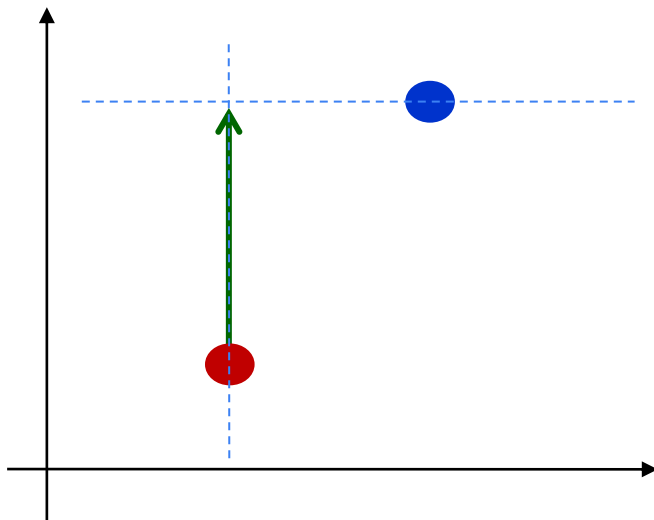
	potência	Peso	aceleração
E ₁	130	3504	12
E ₂	165	3693	11,5
E ₃	150	3436	11

$$d(E_1, E_2) = \sum_{i=1}^3 |130 - 165|^2 + |3504 - 3693|^2 + |12 - 11,5|^2 = 36946,25$$

Medidas de distância

- Dissimilaridade para variáveis contínuas
- **Distância de Chebshev (tabuleiro de xadrez)**
- Usada em situações muito específicas

$$d(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|$$



Medidas de distância

- Dissimilaridade para variáveis contínuas
- **Distância de Chebshev (tabuleiro de xadrez)** $d(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|$

	potência	Peso	aceleração
E ₁	130	3504	12
E ₂	165	3693	11,5
E ₃	150	3436	11

$$d(E_1, E_2) = \max(|130 - 165|, |3504 - 3693|, |12 - 11,5|) = 189$$

Medidas de distância

- Dissimilaridade para variáveis contínuas

- **Distância de Minkowsky**
$$d(x, y)_p = \left(\sum_{i=1}^m (x_i - y_i)^p \right)^{1/p}$$

- $p = 2$: Distância Euclidiana

- $p = 1$: Distância de Manhattan (city block, taxicab)
 - Cai na distância de Hamming para atributos binários

- $p \rightarrow \infty$: Distância suprema (Chebyshev)

Medidas de proximidade

- Medidas de correlação
- **Correlação de Pearson**

$$\text{pearson}(\mathbf{x}, \mathbf{y}) = \frac{\text{convariância}(\mathbf{x}, \mathbf{y})}{\text{variância}(\mathbf{x})\text{variância}(\mathbf{y})}$$

- Insensível à diferenças na magnitude dos atributos
- Muito usada em bioinformática (apenas o padrão de variação dos objetos é importante)
- Interpretação intuitiva é difícil

Medidas de proximidade

- Medidas de correlação

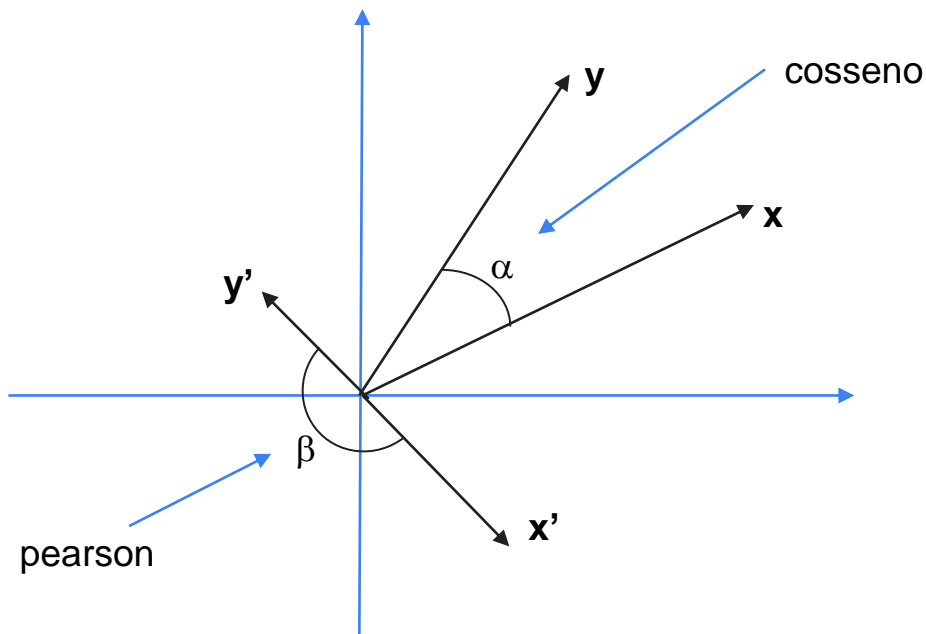
- **Cosseno**

$$\text{cosseno}(x, y) = \frac{\sum_{i=1}^m x_i y_i}{\sqrt{\sum_{i=1}^m x_i^2 \sum_{i=1}^m y_i^2}}$$

- Cosseno do ângulo entre vetores originais
- Muito usada em mineração de textos

Correlação de Pearson e Cosseno

Interpretação geométrica



Medidas de proximidade

- Entre objetos com atributos discretos - Atributos binários
 - Coeficiente de casamento simples (CCS)
 - Tabela de contingências

Coeficiente de casamento simples

$$S_{CCS}(x, y) = \frac{n_{11} + n_{00}}{n_{11} + n_{00} + n_{10} + n_{01}} = \frac{n_{11} + n_{00}}{n}$$

$$1 - S(x, y) = \frac{n_{10} + n_{01}}{n}$$

		y		
x		1	0	Total
	1	n_{11}	n_{10}	$n_{11}+n_{10}$
	0	n_{01}	n_{00}	$n_{01}+n_{00}$
	Total	$n_{11}+n_{01}$	$n_{10}+n_{00}$	n

- Indicada para atributos “simétricos”: todos têm a mesma importância
- Considera casamentos do tipo 1-1 e 0-0

Medidas de proximidade

- Entre objetos com atributos discretos
 - Coeficiente de Jaccard
 - Tabela de contingências

Coeficiente de Jaccard

$$S_{Jaccard}(x, y) = \frac{n_{11}}{n_{11} + n_{10} + n_{01}}$$

$$1 - S(x, y) = \frac{n_{10} + n_{01}}{n}$$

		y		
x		1	0	Total
	1	n_{11}	n_{10}	$n_{11}+n_{10}$
	0	n_{01}	n_{00}	$n_{01}+n_{00}$
	Total	$n_{11}+n_{01}$	$n_{10}+n_{00}$	n

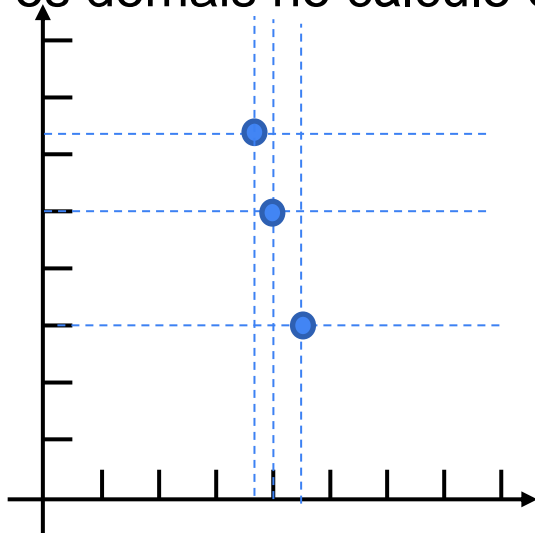
- Indicada para atributos “assimétricos”: presença de um valor é mais importante que a sua ausência
- Considera casamentos do tipo 1-1 e despreza casamentos do tipo 0-0

Transformação de dados

- O cálculo das similaridades e dissimilaridades exige atenção quanto ao tipo e forma dos valores de atributos:
 - Escala
 - Tipo dos valores (contínuo, discreto binário)
 - Valores ausentes
- Valores de atributos devem ser transformados quando necessário

Transformação de dados - Normalizações

- Objetos com atributos com valores e variâncias muito distintos tendem a dominar os demais no cálculo das distâncias



	A1	A2
x_1	40	50
x_2	45	30
x_3	38	63

- Problemas podem ser evitados pela normalização

Transformação de dados - Normalizações

- Re-escala linear [0,1]

- $$x_{ij}' = \frac{x_{ij} - \min(a_j)}{\max(a_j) - \min(a_j)}$$

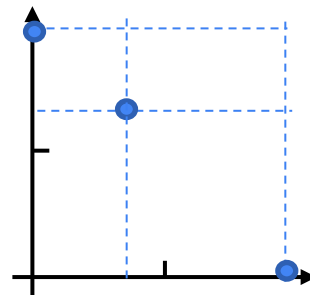
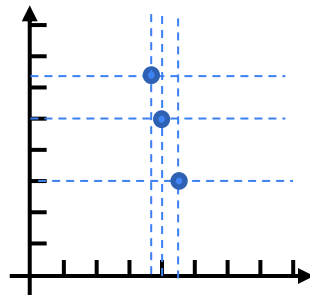
	a_1	a_2
x_1	40	50
x_2	43	30
x_3	38	60

	a_1	a_2
x_1	0,4	0,67
x_2	1	0
x_3	0	1

$$x_{11}' = \frac{40-38}{43-38} = \frac{2}{5} = 0,4 \quad x_{12}' = \frac{50-30}{60-30} = \frac{20}{30} = 0,67$$

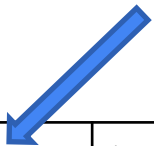
$$x_{21}' = \frac{43-38}{43-38} = \frac{5}{5} = 1 \quad x_{22}' = \frac{30-30}{60-30} = \frac{0}{30} = 0$$

$$x_{31}' = \frac{38-38}{43-38} = \frac{0}{5} = 0 \quad x_{32}' = \frac{60-30}{60-30} = \frac{30}{30} = 1$$

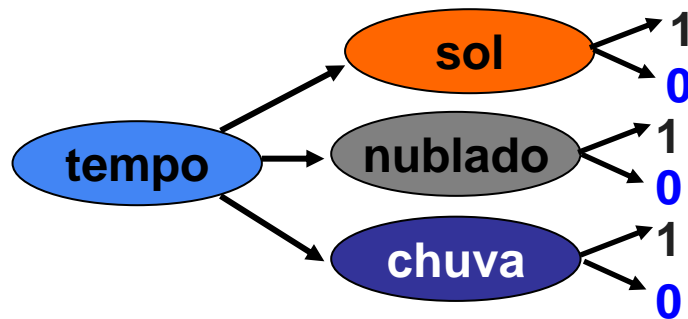


Transformação de dados

- Transformar atributos nominais (não binários) para binários
- Codificação 1-de-n
- Transformar cada valor do atributo nominal em uma variável binária fictícia.



	tempo	temperatura	umidade	vento
E1	sol	amena	alta	forte
E2	nublado	frio	média	forte
E3	sol	frio	alta	fraco



-  Pode levar a um número muito elevado de atributos

Transformação de dados

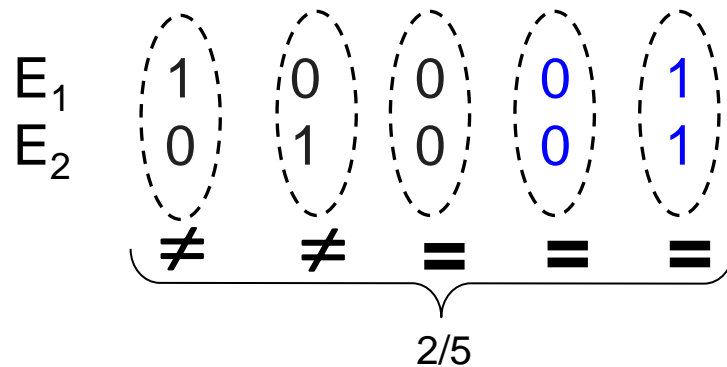
	tempo	temperatura	umidade	vento
E ₁	sol	amena	alta	forte
E ₂	nublado	frio	média	forte
E ₃	sol	frio	alta	fraco

	sol	nublado	chuva	temperatura	umidade	vento
E ₁	1	0	0	amena	alta	forte
E ₂	0	1	0	frio	média	forte
E ₃	1	0	0	frio	alta	fraco

Similaridade para variáveis nominais binárias

- Depois de transformar todos os atributos para binários, podemos utilizar a medida de (dis)similaridade coeficiente de casamento simples

	tempo			vento	
	sol	nublado	chuva	fraco	forte
E ₁	1	0	0	0	1
E ₂	0	1	0	0	1
E ₃	1	0	0	1	0



$$1 - S(x, y) = \frac{n_{10} + n_{01}}{n}$$

$$1 - S(E_1, E_2) = \frac{n_{10} + n_{01}}{n} = \frac{1 + 1}{5} = \frac{2}{5}$$

Agrupamento de dados particional



Algoritmos particionais

- Dado um conjunto de n objetos $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$
- Algoritmos particionais (partição rígida sem sobreposição) dividem o conjunto de objetos em k subconjuntos **disjuntos não vazios**
- $X = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$
- $C = \{\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_4\}, \{\mathbf{x}_6\}, \{\mathbf{x}_3, \mathbf{x}_5\}\}$
 - Gera uma única partição nos dados.

Algoritmos particionais

- Muitos algoritmos particionais tem um elemento representante (protótipo) para cada cluster, que resume as informações do cluster

- Protótipos mais comuns:

- Centróide -
$$\bar{x}^k = \frac{1}{n_k} \sum_{x_i \in C_k} x_i$$

- Medoide – elemento mais próximo dos outros elementos

- Mediana – elemento médio

Algoritmos particionais de erro quadrático

- Encontram a **partição** dos dados (agrupamento) de forma iterativa, repetindo passos que visam otimizar (minimizar) uma função de erro quadrático
- A **função de erro quadrático** é o critério do processo de agrupamento que garante a propriedade de compactação dos clusters
- O erro quadrático de um agrupamento C com k clusters de um conjunto de exemplos E é:

$$err2(E, C) = \sum_{j=1}^k \sum_{i=1}^{n_j} \|x_i^{(j)} - c_j\|^2$$

- Onde $x_i^{(j)}$ é o i -ésimo exemplo pertencente ao j -ésimo cluster
- e c_j é o centróide do j -ésimo cluster

Algoritmo K-Means (K-Médias)

- Mais simples e mais conhecido algoritmo de erro quadrático.
- É fácil de implementar e sua complexidade é $O(n)$ com n sendo o número de exemplos.
- O protótipo do grupo é o **centroide** do grupo
- Na versão original e na maioria das aplicações usa **distância euclidiana quadrática** para calcular dissimilaridade entre os elementos
- **Problemas:**
 - O usuário define previamente o número de grupos k
 - É sensível à partição inicial e pode convergir para um mínimo local do valor da função critério se a partição inicial não for escolhida apropriadamente

Algoritmo K-Means (K-Médias)

- 1) **Escolha** k centros de clusters que podem ou não coincidir com os exemplos.
- 2) **Atribua** cada exemplo ao centro de cluster mais próximo.
- 3) **Recalcule** os centros de clusters usando os grupos definidos.
- 4) Se o **critério de convergência** não for satisfeito, vá para o passo 2.

Algoritmo K-Means (K-Médias)

- **Parâmetros de entrada:**

- Conjunto de N exemplos não rotulados $x_i, i=1, \dots, N$
- k – número de grupos
- *dist* - medida de distância

- **Parâmetros de saída:**

- k vetores que representam **centroides** de grupos

Algoritmo K-Means (K-Médias)

Escolha aleatoriamente um conjunto de vetores distintos para representar os centroides $c_j, j = 1, \dots, k$

Repeat

For $i=1$ to N

 Calcule a distância $dist(x_i, c_j)$ de x_i a cada centroide $c_j, i = 1, \dots, k$

 Associe x_i ao centroide c_j que minimiza $dist(x_i, c_j)$

End {For}

For $j=1$ to m

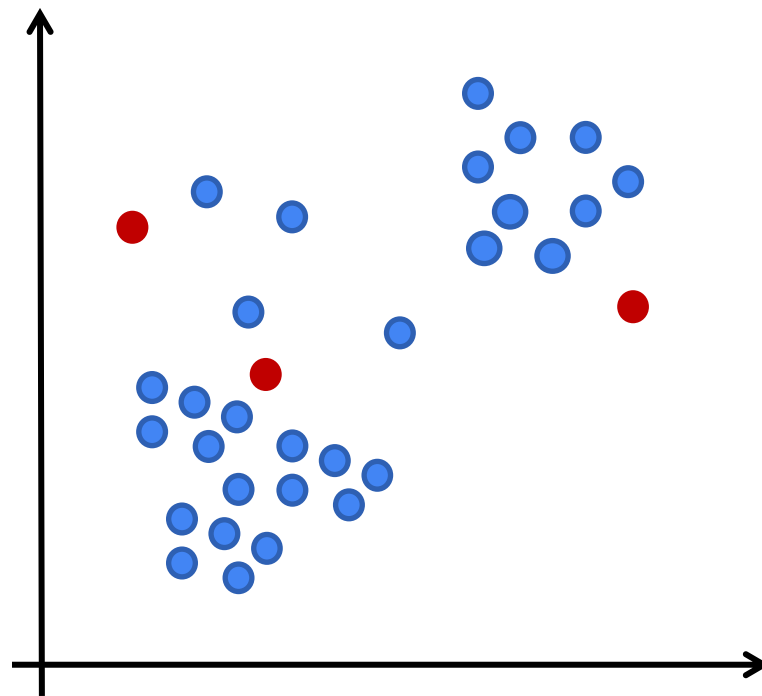
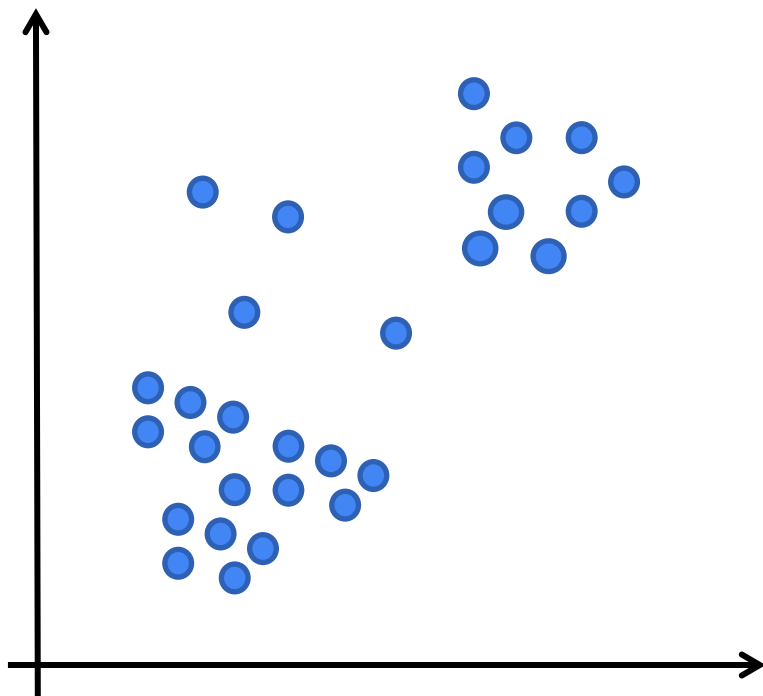
 Atualize os centroides $c_j, j = 1, \dots, k$ calculando a média dos exemplos que pertencem ao grupo com centro c_j

End {For}

Until nenhuma mudança nos c_j ocorra entre duas iterações sucessivas

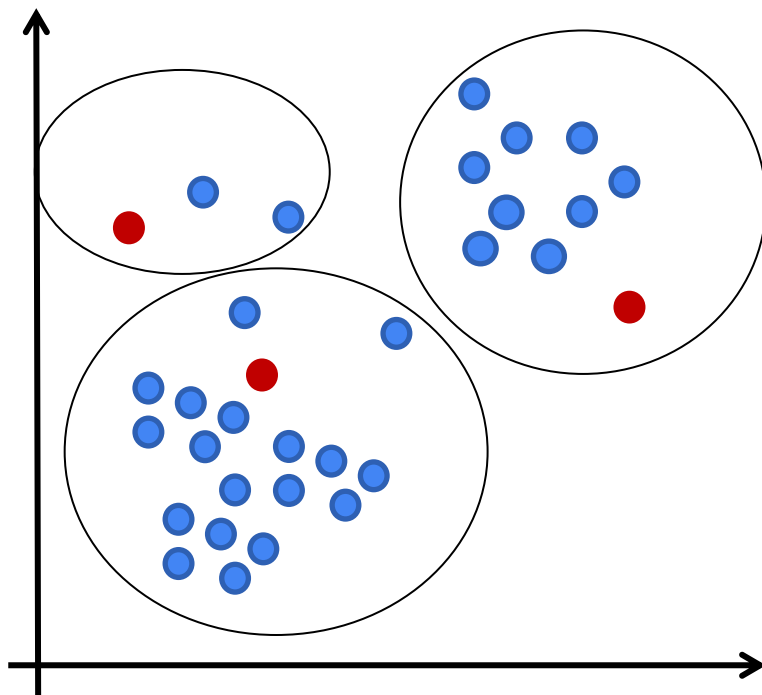
Algoritmo K-Means (K-Médias)

Valores iniciais dos centros de cluster são definidos aleatoriamente



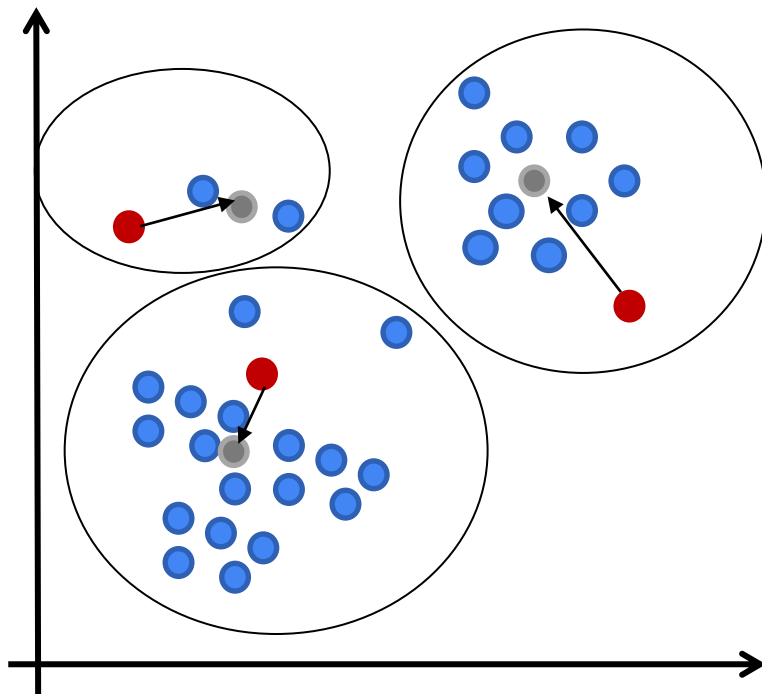
Algoritmo K-Means (K-Médias)

Dados são atribuídos a grupos, pela menor distância ao centro de cada grupo



Algoritmo K-Means (K-Médias)

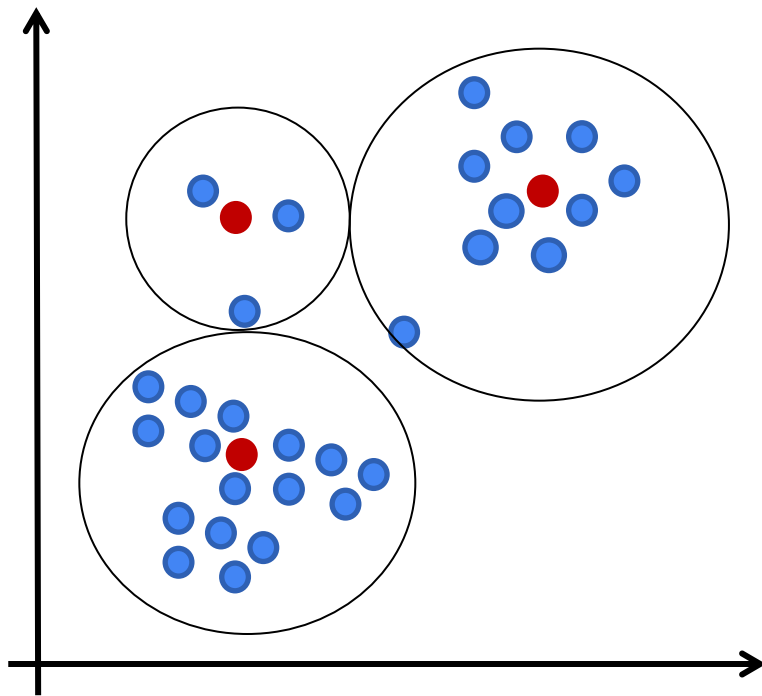
Novos centros de cluster são calculados (centróide)



Algoritmo K-Means (K-Médias)

Dados são reatribuídos, podendo mudar de grupo

O processo se repete
até que as mudanças
sejam muito
pequenas ou
inexistentes.



Algoritmo K-Means (K-Médias)

□ Considere os exemplos:

$$\mathbf{x}_1 = [2, 5]$$

$$\mathbf{x}_2 = [6, 4]$$

$$\mathbf{x}_3 = [5, 3]$$

$$\mathbf{x}_4 = [2, 2]$$

$$\mathbf{x}_5 = [1, 4]$$

$$\mathbf{x}_6 = [5, 2]$$

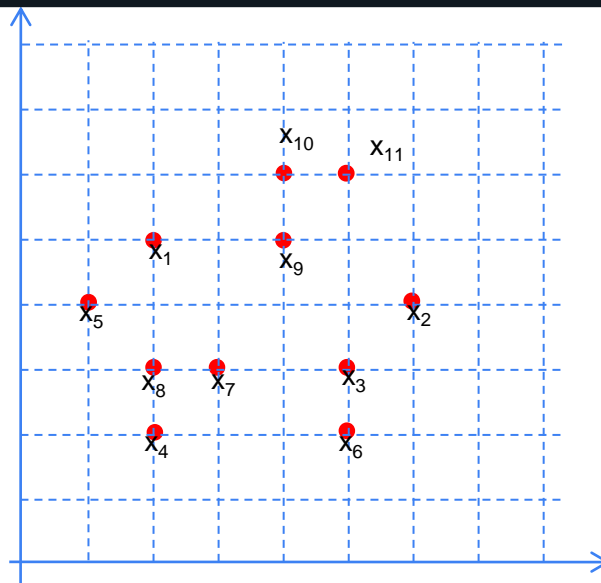
$$\mathbf{x}_7 = [3, 3]$$

$$\mathbf{x}_8 = [2, 3]$$

$$\mathbf{x}_9 = [4, 5]$$

$$\mathbf{x}_{10} = [4, 6]$$

$$\mathbf{x}_{11} = [5, 6]$$



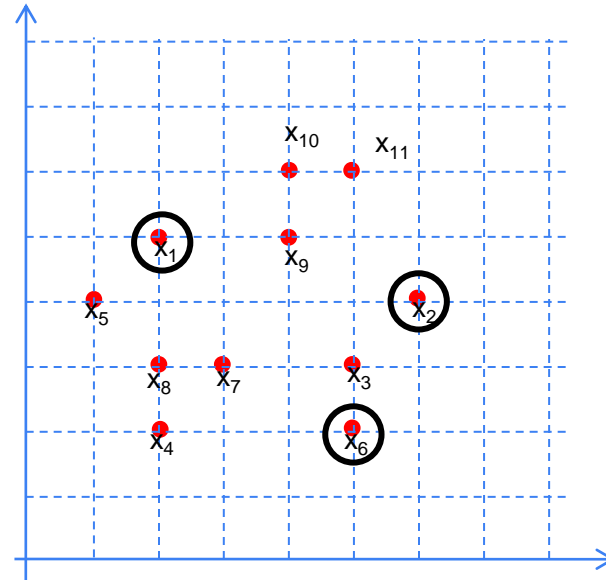
Considere que a distância entre exemplos é calculada como a distância euclidiana

Aplicar o algoritmo K-means (para k=3)

- Gerar aleatoriamente o vetor de centróides inicial:

$$\Theta = \{x_1, x_2, x_6\}$$

c1	2	5
c2	6	4
c3	5	2



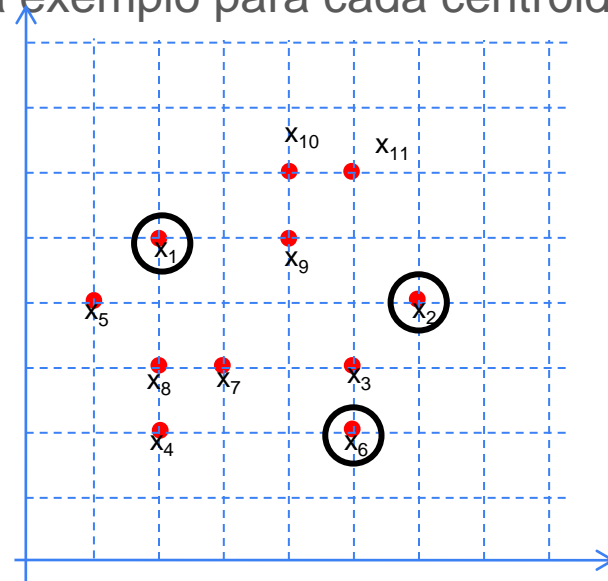
Aplicar o algoritmo K-means (para k=3)

- Gerar aleatoriamente o vetor de centróides inicial:

$$\Theta = \{x_1, x_2, x_6\}$$

- Calcular a distância de cada exemplo para cada centróide:

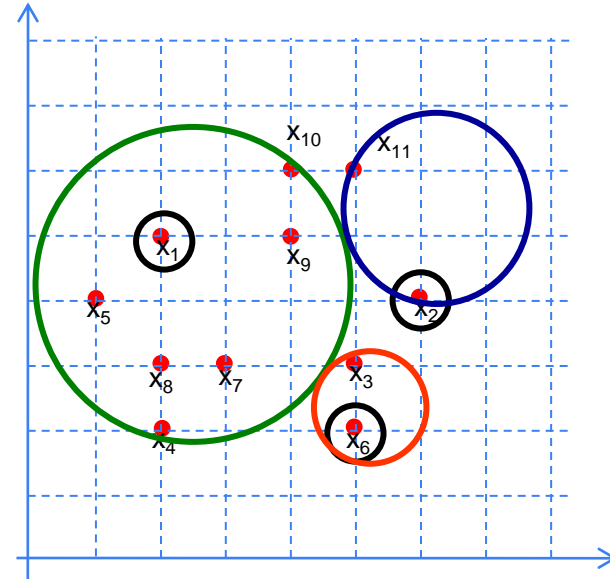
	c1	c2	c3
x1	0	4,1	4,2
x2	4,1	0,0	2,2
x3	3,6	1,4	1,0
x4	3,0	4,5	3,0
x5	1,4	5,0	4,5
x6	4,2	2,2	0,0
x7	2,2	3,2	2,2
x8	2,0	4,1	3,2
x9	2,0	2,2	3,2
x10	2,2	2,8	4,1
x11	3,2	2,2	4,0



Aplicar o algoritmo K-means (para k=3)

- Atribuir cada exemplo a um cluster, pela menor distância ao centróide:

	c1	c2	c3	grupo
x1	0	4,1	4,2	c1
x2	4,1	0,0	2,2	c2
x3	3,6	1,4	1,0	c3
x4	3,0	4,5	3,0	c1
x5	1,4	5,0	4,5	c1
x6	4,2	2,2	0,0	c3
x7	2,2	3,2	2,2	c1
x8	2,0	4,1	3,2	c1
x9	2,0	2,2	3,2	c1
x10	2,2	2,8	4,1	c1
x11	3,2	2,2	4,0	c2



Aplicar o algoritmo K-means (para $k=3$)

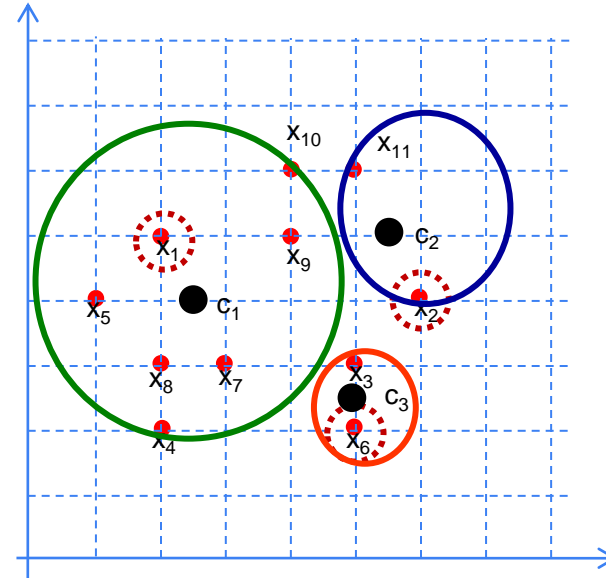
- Recalcular os centróides:

Atual:

c1	2	5
c2	6	4
c3	5	2

Novo:

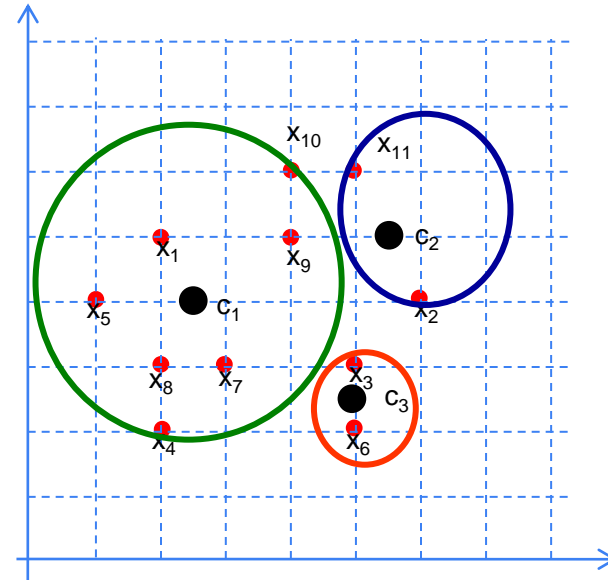
c1	2,6	4
c2	5,5	5
c3	5	2,5



Aplicar o algoritmo K-means (para k=3)

- Calcular a distância de cada exemplo para cada centróide:
- Atribuir cada exemplo a um cluster, pela menor distância ao centróide:

	c1	c2	c3	
x1	1,2	3,5	3,9	c1
x2	3,4	1,1	1,8	c2
x3	2,6	2,1	0,5	c3
x4	2,1	4,6	3,0	c1
x5	1,6	4,6	4,3	c1
x6	3,1	3,0	0,5	c3
x7	1,1	3,2	2,1	c1
x8	1,2	4,0	3,0	c1
x9	1,7	1,5	2,7	c2
x10	2,4	1,8	3,6	c2
x11	3,1	1,1	3,5	c2

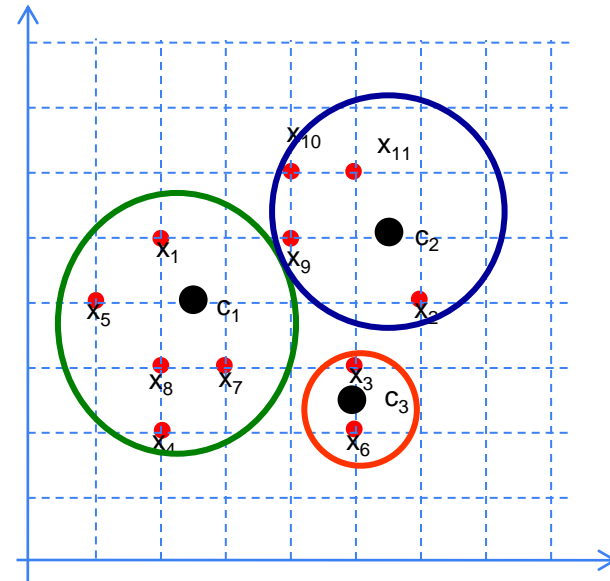


Aplicar o algoritmo K-means (para k=3)

- Atribuir cada exemplo a um cluster, pela menor distância ao centróide:

	c1	c2	c3	
x1	1,2	3,5	3,9	c1
x2	3,4	1,1	1,8	c2
x3	2,6	2,1	0,5	c3
x4	2,1	4,6	3,0	c1
x5	1,6	4,6	4,3	c1
x6	3,1	3,0	0,5	c3
x7	1,1	3,2	2,1	c1
x8	1,2	4,0	3,0	c1
x9	1,7	1,5	2,7	c2
x10	2,4	1,8	3,6	c2
x11	3,1	1,1	3,5	c2

x_9 e x_{10} mudaram de c_1 para c_2



Aplicar o algoritmo K-means (para k=3)

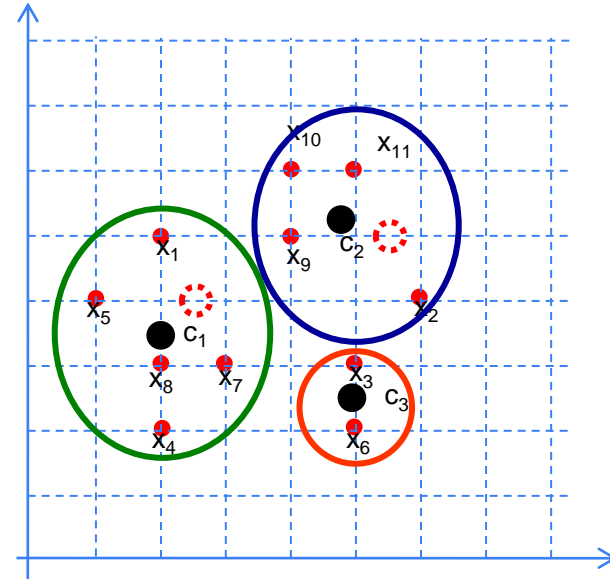
- Recalcular os centróides:

Atual:

c1	2,6	4
c2	5,5	5
c3	5	2,5

Novo:

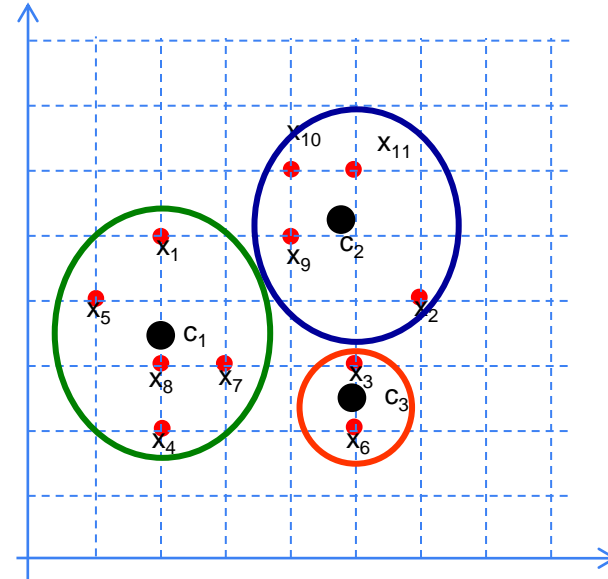
c1	2,0	3,4
c2	4,8	5,3
c3	5	2,5



Aplicar o algoritmo K-means (para k=3)

- Calcular a distância de cada exemplo para cada centróide:
- Atribuir cada exemplo a um cluster, pela menor distância ao centróide:

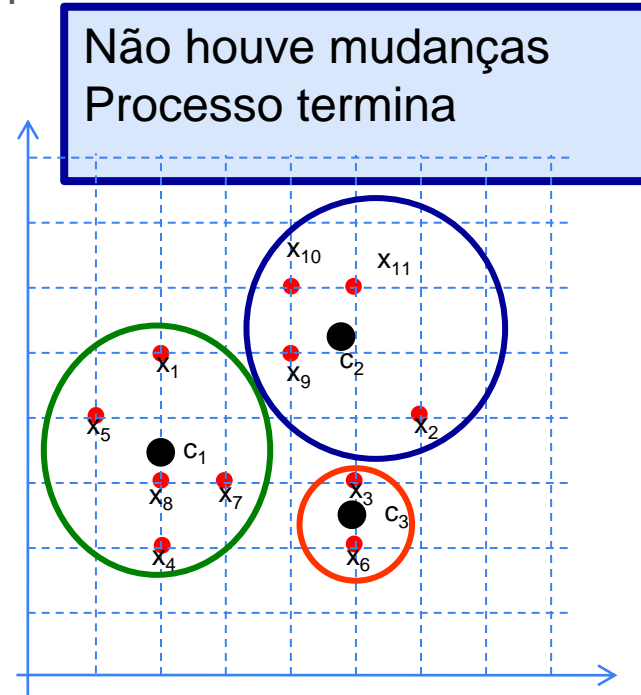
	c1	c2	c3	
x1	1,6	2,8	3,9	c1
x2	4,0	1,77	1,80	c2
x3	3,0	2,3	0,5	c3
x4	1,4	4,3	3,0	c1
x5	1,2	4,0	4,3	c1
x6	3,3	3,3	0,5	c3
x7	1,1	2,9	2,1	c1
x8	0,4	3,6	3,0	c1
x9	2,6	0,9	2,7	c2
x10	3,3	1,1	3,6	c2
x11	4,0	0,7	3,5	c2



Aplicar o algoritmo K-means (para k=3)

- Atribuir cada exemplo a um cluster, pela menor distância ao centróide:

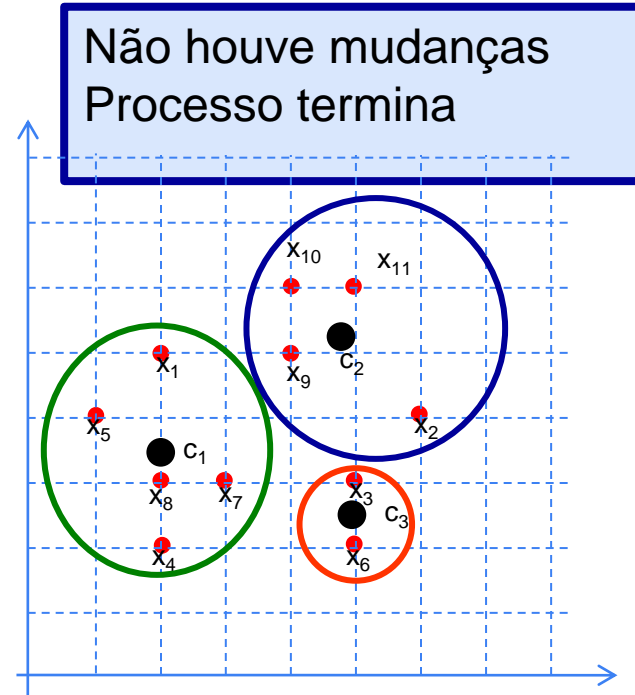
	c1	c2	c3	
x1	1,6	2,8	3,9	c1
x2	4,0	1,77	1,80	c2
x3	3,0	2,3	0,5	c3
x4	1,4	4,3	3,0	c1
x5	1,2	4,0	4,3	c1
x6	3,3	3,3	0,5	c3
x7	1,1	2,9	2,1	c1
x8	0,4	3,6	3,0	c1
x9	2,6	0,9	2,7	c2
x10	3,3	1,1	3,6	c2
x11	4,0	0,7	3,5	c2



Aplicar o algoritmo K-means (para k=3)

- Atribuir cada exemplo a um cluster, pela menor distância ao centróide:

	c1	c2	c3	
x1	1,6	2,8	3,9	c1
x2	4,0	1,77	1,80	c2
x3	3,0	2,3	0,5	c3
x4	1,4	4,3	3,0	c1
x5	1,2	4,0	4,3	c1
x6	3,3	3,3	0,5	c3
x7	1,1	2,9	2,1	c1
x8	0,4	3,6	3,0	c1
x9	2,6	0,9	2,7	c2
x10	3,3	1,1	3,6	c2
x11	4,0	0,7	3,5	c2



Algoritmo K-Means (K-Médias)

- Veremos exemplos de geração de dados sintéticos e execução do algoritmo Kmeans em Python com esses dados
- Para isso vamos recordar como gerar e visualizar (plotar) dados sintéticos de grupos
- Bibliotecas Python:
 - Scikit-learn: <http://scikit-learn.org>
 - Numpy: <https://numpy.org/>
 - Matplotlib: <https://matplotlib.org/>
- Google Colaboratory: <https://colab.research.google.com>
- Tutorial: <https://colab.research.google.com/notebooks/intro.ipynb>

Geração de dados com make_blobs

- Generate isotropic Gaussian blobs for clustering.
- Modulo: sklearn.datasets
- **Função:**
make_blobs(*n_samples=100, n_features=2, centers=None, cluster_std=1.0, center_box=(-10.0, 10.0), shuffle=True, random_state=None, return_centers=False*)
- **Parâmetro centers :**
 - ***int ou array da forma [n_centers, n_features] (opcional)***
 - (default=None)
 - Define o número de centros a gerar ou as localizações dos centros de grupos
 - Se n_samples é int e centers é None, são gerados 3 centros.
 - Se n_samples é um array, centers deve ser ou None ou um array de tamanho [centers, n_features].

Geração de dados com make_blobs

- **Parâmetro cluster_std :**
 - *float ou sequencia de floats, (opcional)*
 - *default=1.0*
 - Desvio padrão dos clusters.
- **Retorna:**
 - ***X : ndarray da forma [n_samples, n_features]***
 - As amostras geradas.
 - ***y : ndarray da forma [n_samples]***
 - Inteiros que representam o cluster de cada amostra.
 - ***centers : ndarray da forma [n_centers, n_features]***
 - Centros de cada cluster, somente se return_centers = True

Geração de dados com make_blobs

#gerando dados 2D

```
n_grupos = 3
```

```
X, y = make_blobs(centers = n_grupos)
```

#gerando grupos com tamanhos diferentes

```
n_grupos = 2
```

```
X, y = make_blobs(n_samples=[100,10], centers = ([[1,1],[5,5]]))
```

Visualizando dados com scatter

- Matplotlib: biblioteca para criar visualizações em Python
- Pacote: matplotlib.pyplot
 - Função:
`scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, *, plotnonfinite=False, data=None, **kwargs)`
- Parâmetros x e y:
 - Array da forma (n,)
 - x: Valores do primeiro atributo dos pontos a serem plotados
 - y: Valores do segundo atributo dos pontos a serem plotados

Exemplos de geração de dados

#Para gerar e visualizar dados vamos carregar os seguintes pacotes e funções:

```
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.datasets import make_blobs
```

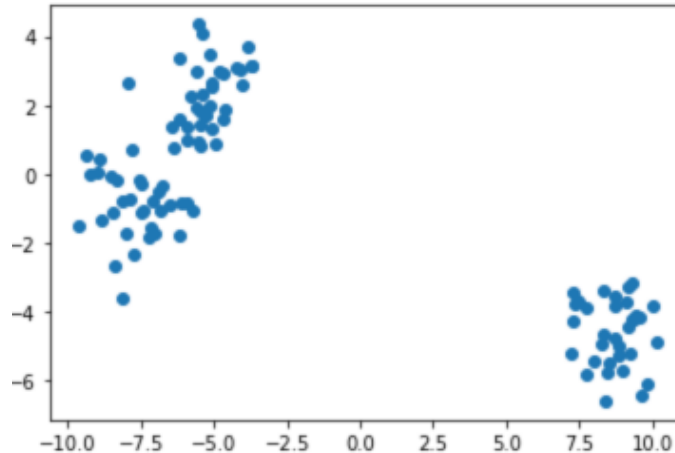
Exemplos de geração de dados

#gerando dados 2D com 3 grupos e 100 instâncias no total

n_grupos = 3

X, y = make_blobs(centers = n_grupos)

plt.scatter(X[:,0], X[:,1])



#gerando dados 2D com 3 grupos e 100 instâncias no total

#X: instâncias geradas (2 atributos)

#y: grupos de cada instância

X

```
array([[ -6.88837536,  1.79066592],
       [ -8.61279112,  6.89739496],
       [ -9.02420088,  2.87707642],
       [ -5.56362095, 10.0475503 ],
       [  2.73613076,  6.35793177],
       [  3.25990611,  4.15780354],
       [ -7.75125968,  7.00458238],
       [  1.57086943,  3.85329018],
       [ -9.16612953,  0.66575654],
       [ -9.34508894,  1.71651655],
       [ -9.653855 ,  1.78554725],
       [  0.72210776,  5.83982055],
       [ -8.19291948,  8.10795287],
       [ -8.48785964,  0.18276928],
       [ -9.38218364,  0.41973531],
       [ -9.89646045,  4.16543154],
       [  1.63224782,  4.4470093 ],
       [-10.47519836,  1.39705357],
       [ -8.0911452 ,  5.4985043 ],
       [ -8.55746931,  3.61760131],
       [ -8.42234322,  8.67185696],
       [ -8.67481147,  3.64569272],
       [  1.48871533,  5.13508547],
```

Y

```
array([1, 0, 0, 2, 2, 1, 0, 2, 2, 0, 1, 0, 1, 2, 0, 1, 2, 0, 2, 2, 1, 0,
       2, 2, 1, 2, 1, 2, 2, 1, 2, 0, 1, 0, 0, 1, 0, 0, 0, 1, 2, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 2, 2, 1, 2, 0, 1, 0, 2, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 1, 2, 2, 2, 0, 1, 2, 1, 0, 0, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0,
       1, 1, 2, 1, 1, 0, 2, 1, 1, 1, 1, 1])
```

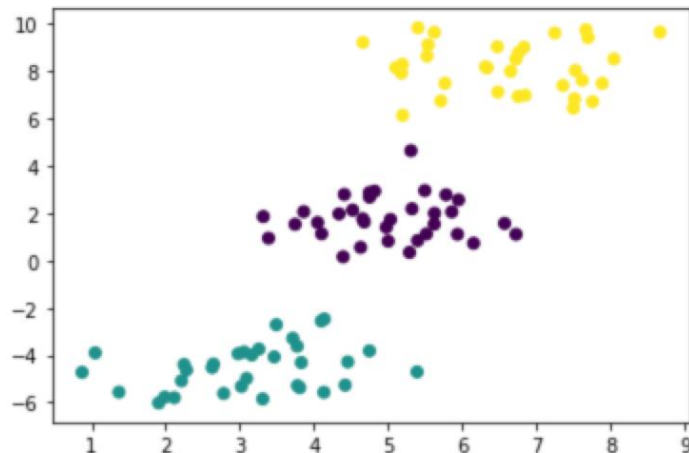
Visualizando grupos com cores diferentes

#gerando dados 2D com 3 grupos e 100 instâncias no total

n_grupos = 3

X, y = make_blobs(centers = n_grupos)

plt.scatter(X[:,0], X[:,1], c=y)

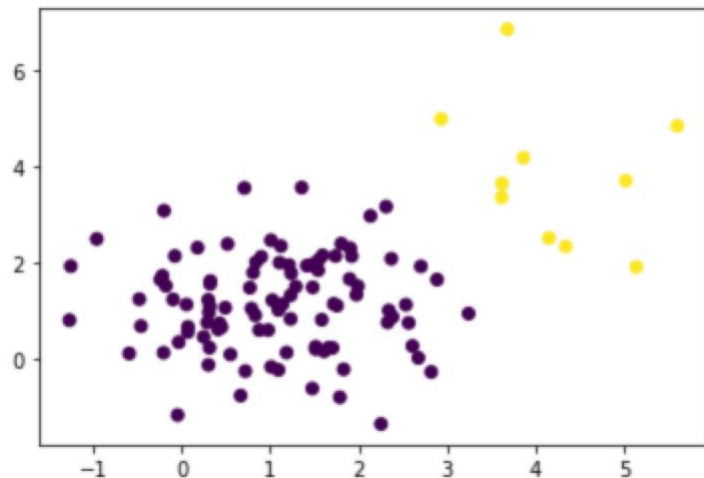


Gerando grupos com tamanhos diferentes

#gerando grupos com tamanhos diferentes e definindo os centros

```
X, y = make_blobs(n_samples=[100,10], centers = ([[1,1],[4,4]]))
```

```
plt.scatter(X[:,0], X[:,1], c=y)
```



Gerando grupos com números de instâncias diferentes

#gerando grupos com tamanhos diferentes e definindo o desvio padrão

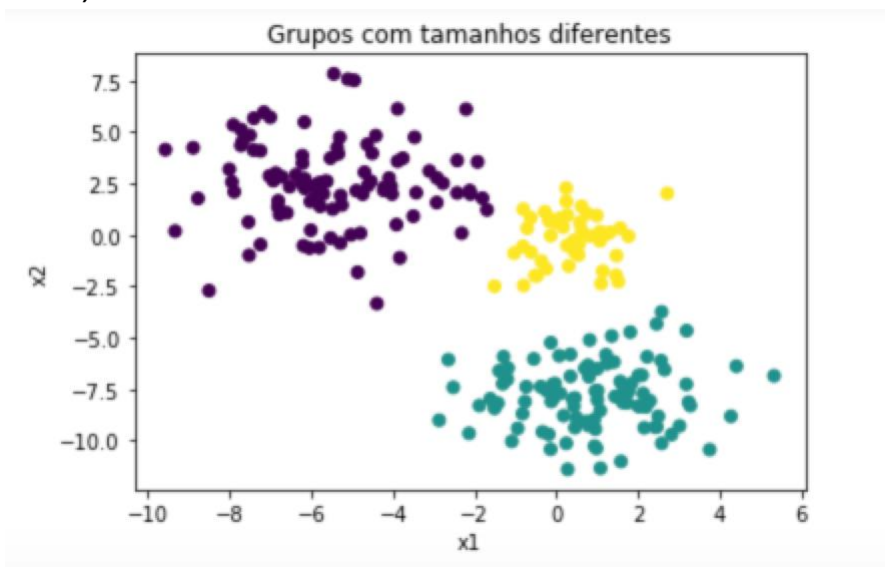
```
X, y = make_blobs(n_samples=[100,100,50], cluster_std=[2,1.5,1])
```

```
plt.scatter(X[:,0], X[:,1], c=y)
```

```
plt.title("Grupos com tamanhos diferentes")
```

```
plt.xlabel("x1")
```

```
plt.ylabel("x2")
```



K-Means em Python

- Importar a biblioteca e o objeto Kmeans

```
from sklearn.cluster import Kmeans
```

```
class sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd')
```

- Parâmetros

- **n_clusters** número de clusters que o algoritmo vai gerar (padrão: 8)
- **init** modo como o algoritmo será inicializado:
 - **k-means++**: método padrão, favorece a convergência.
 - **random**: Inicializa os centroides de forma aleatória
 - **ndarray**: Especifica um array de valores indicando qual seriam os centroides a serem usados para a inicialização.
- **max_iter**: número máximo de iterações (padrão: 300)
- **algorithm**: especifica a versão do algoritmo K-Means a ser utilizada. A versão clássica é executada através do valor **lloyd**.

K-Means em Python

- **Atributos:**
- **cluster_centers_** *ndarray of shape (n_clusters, n_features)*
Coordenadas dos centros dos cluster
- **labels_** *ndarray of shape (n_samples,)*
Indicador do cluster de cada ponto
- **inertia_** *float*
- Soma dos quadrados das distâncias das amostras ao centro mais próximo
- **n_iter** Número de iterações executadas
- **n_features** Número de atributos olhados durante o processo
- **feature_names_in_** *ndarray of shape (n_features_in_,)*
- Nomes dos atributos vistos durante a execução. Definido somente quando X tem nomes de atributos que são todos strings

K-Means em Python

- Inicializar o K-means utilizando 3 clusters e método de inicialização random.

```
kmeans = KMeans(n_clusters = 3, init = 'random')
```

- Executar o método **fit()** para executar o algoritmo e agrupar os dados. O método fit() recebe como parâmetro os dados a serem agrupados

```
kmeans.fit(X)
```

- Exibir os centroides gerados pelo atributo **cluster_centers_**.

```
kmeans.cluster_centers_
```

K-Means em Python

- Tabela de distâncias **fit_transform()**:

- executa o K-means para agrupar os dados e retorna uma tabela de distâncias.

```
distance = kmeans.fit_transform(X)
distance
```

- Atributo **labels_** : retorna os labels para cada instância, ou seja, o número do cluster a que a instância de dados foi atribuída.

```
labels = kmeans.labels_ labels
```

Índices de validação interno em Python

- A própria função de erro quadrático, otimizada no processo do Kmeans, pode ser usada como uma métrica de validação do agrupamento
- O algoritmo Kmeans em Python tem um atributo:
 - **inertia_** : float
 - Soma do quadrado das distâncias das instâncias para o centro de cluster mais próximo

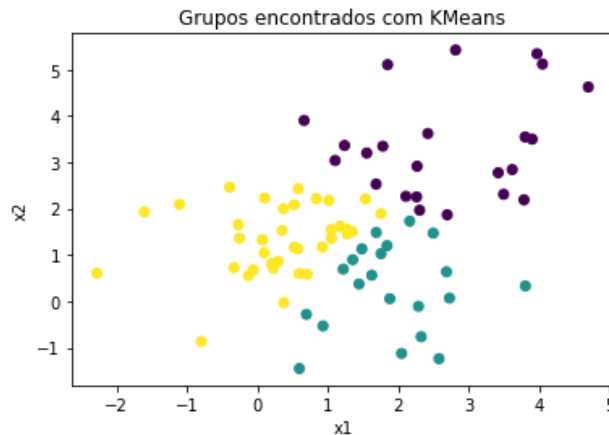
```
# Executando KMeans
```

```
y_pred = Kmeans(n_clusters=3,init='random')
```

```
y_pred.fit(X)
```

```
y_pred.inertia_
```

```
127.4038470604296
```



Encontrar o melhor número de grupos para o algoritmo KMeans

- Método do “cotovelo”
 - Definir um intervalo de valores para o número de grupos $[k_{\min}, k_{\max}]$
 - Escolher um índice de validação (soma do quadrado das distâncias, silhueta, davis-bouldin,...)
 - Executar o algoritmo Kmeans para cada um dos valores de k (número de grupos)
 - Criar um gráfico com as coordenadas número de grupos “versus” índice
 - Encontrar o ponto do gráfico em que a melhora no valor do índice não é significativa (cotovelo)

Encontrar o melhor número de grupos para o algoritmo KMeans

#Importações

```
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.metrics as sm
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
```

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Usar a função read_csv da biblioteca pandas para ler o conjunto de dados  
#Mostrar as primeiras cinco linhas com a função head()
```

```
data = pd.read_csv('Wholesale customers data.csv')  
data.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Definir a lista de atributos categóricos e atributos contínuos
```

```
#Criar as estatísticas com a função describe
```

```
categorical_features = ['Channel', 'Region']
```

```
continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']
```

```
data[continuous_features].describe()
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Transformar atributos categóricos em binários
```

```
for col in categorical_features:  
    dummies = pd.get_dummies(data[col], prefix=col)  
    data = pd.concat([data, dummies], axis=1)  
    data.drop(col, axis=1, inplace=True)  
data.head()
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	Channel_1	Channel_2	Region_1	Region_2	Region_3
0	12669	9656	7561	214	2674	1338	0	1	0	0	1
1	7057	9810	9568	1762	3293	1776	0	1	0	0	1
2	6353	8808	7684	2405	3516	7844	0	1	0	0	1
3	13265	1196	4221	6404	507	1788	1	0	0	0	1
4	22615	5410	7198	3915	1777	5185	0	1	0	0	1

Encontrar o melhor número de grupos para o algoritmo KMeans

```
#Normalizar os atributos contínuos
```

```
#escalar os atributos contínuos  
mms = MinMaxScaler()  
mms.fit(data)  
data_transformed = mms.transform(data)
```

```
# Fazer o agrupamento para cada quantidade de grupo no intervalo definido  
Sum_of_squared_distances = [ ]  
K = range(1,15)  
for k in K:  
    km = KMeans(n_clusters=k)  
    km = km.fit(data_transformed)  
    Sum_of_squared_distances.append(km.inertia_)
```

Encontrar o melhor número de grupos para o algoritmo KMeans

#Construir o gráfico número de clusters X índice

```
plt.plot(K, Sum_of_squared_distances, 'bx-')  
plt.xlabel('k')  
plt.ylabel('Soma dos quadrados das distâncias')  
plt.title('Método do Cotovelo para encontrar melhor valor de k')  
plt.show()
```

