KMeans

Algumas observações

Algoritmo K-Means – Algumas observações

- **É possível utilizar K-means com outras medidas de distância?**

- Em geral, K-means não tem bom desempenho com outras medidas de distância, quanto á convergência
  - Implementação do K-means no Scikit-Learn não permite alterar a medida de distância.

- Entretanto, é comum, na área de “text analytics”, agrupar documentos de texto com K-means usando dissimilaridade de cosseno
  - Bilbioteca NLTK permite selecionar a medida de distância na implementação do K-means
  - https://www.nltk.org/api/nltk.cluster.html#nltk.cluster.api.ClusterI.cluster

Algoritmo K-Means – Algumas observações

#Exemplo Kmeans – implementação do NLTK

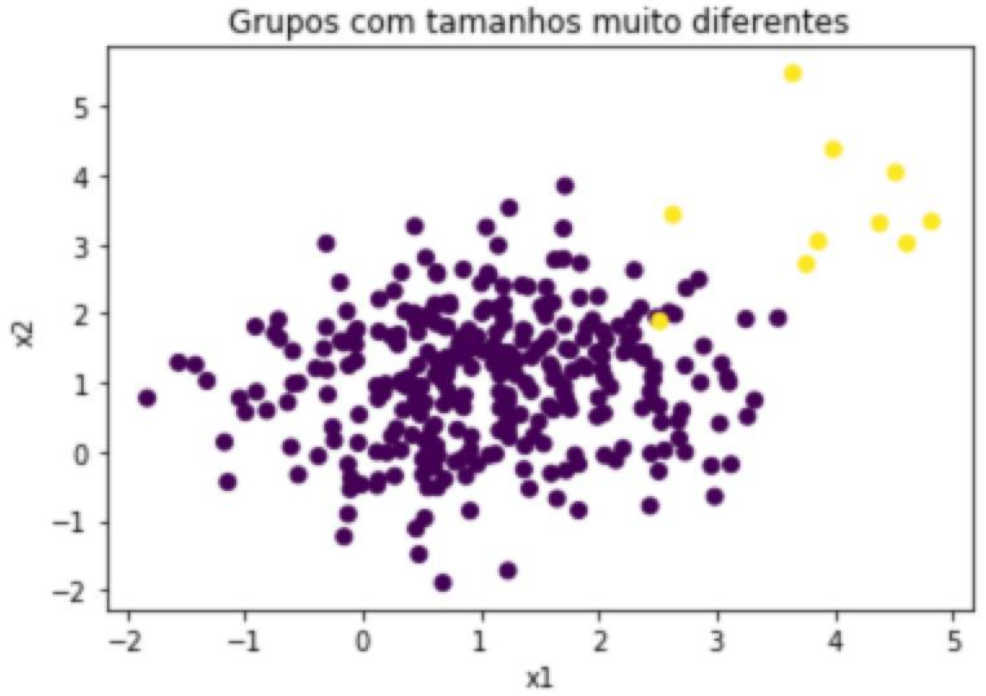
```
from nltk.cluster import KMeansClusterer,euclidean_distance, cosine_distance

NUM_CLUSTERS=3

clusterer = KMeansClusterer(NUM_CLUSTERS, distance=cosine_distance)
#clusterer = KMeansClusterer(NUM_CLUSTERS, euclidean_distance)

clusters = clusterer.cluster(X, True)
print('Labels:', clusters)
print('Means:', clusterer.means())
```

Algoritmo K-Means - Desvantagens

- Algoritmo K-means não funciona bem com:
  - Grupos com **tamanhos** (número de objetos) muito diferentes
  - Grupos com **densidades** muito diferentes
  - Conjuntos de dados com **outliers**
- É sensível (pode chegar a partições muito diferentes) à definição inicial de centroides
- O número de grupos deve ser definido *a priori*
- Não é adequado para atributos nominais

Algoritmo K-means - desvantagens

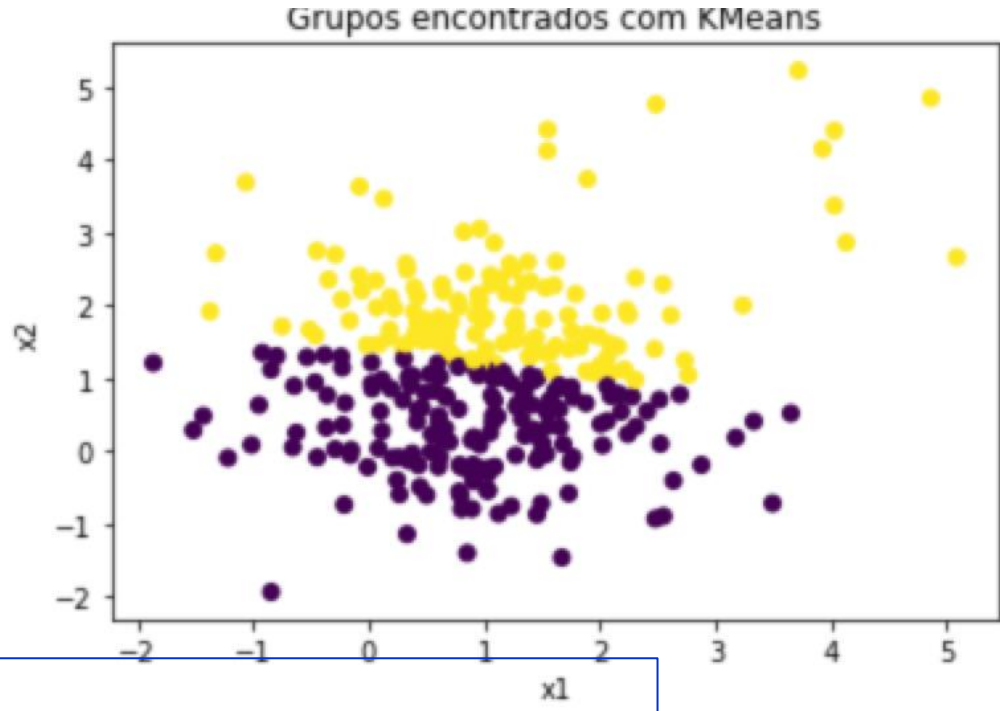
- *Tamanhos de clusters significativamente diferentes*
  - Foram gerados dois grupos de dados, sendo:
  - Um com 300 dados, outro com 10




```
#gerando dados 2D com dois grupos de tamanhos diferentes

X, y = datasets.make_blobs(n_samples=[300,10])
plt.scatter(X[:,0], X[:,1], c=y)
```

Algoritmo K-means - desvantagens

- *Tamanhos de clusters significativamente diferentes*
  - Agrupando os dados com K-means
  - 2 grupos
  - Centros inicializados aleatoriamente

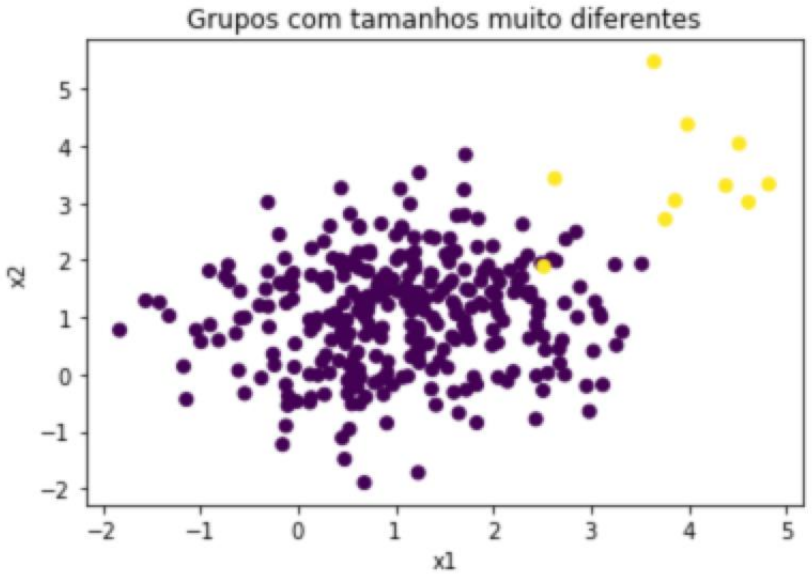



```
#agrupando com KMeans
kmeans = KMeans(n_clusters = 2, init = 'random')
kmeans.fit(X)
kmeans.cluster_centers_
y_pred = kmeans.labels_
plt.scatter(X[:,0], X[:,1], c=y_pred)
```

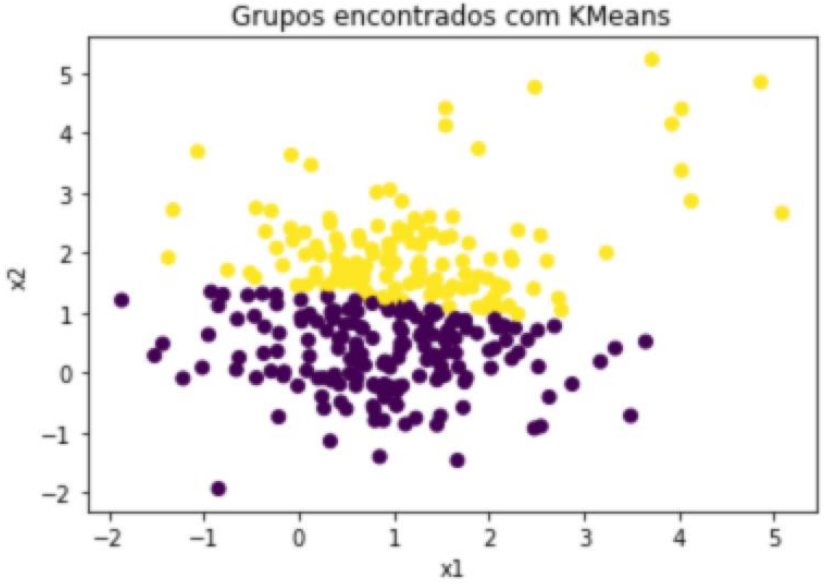
Algoritmo K-means - desvantagens

Tamanhos de clusters significativamente diferentes

Grupos originais



Grupos encontrados



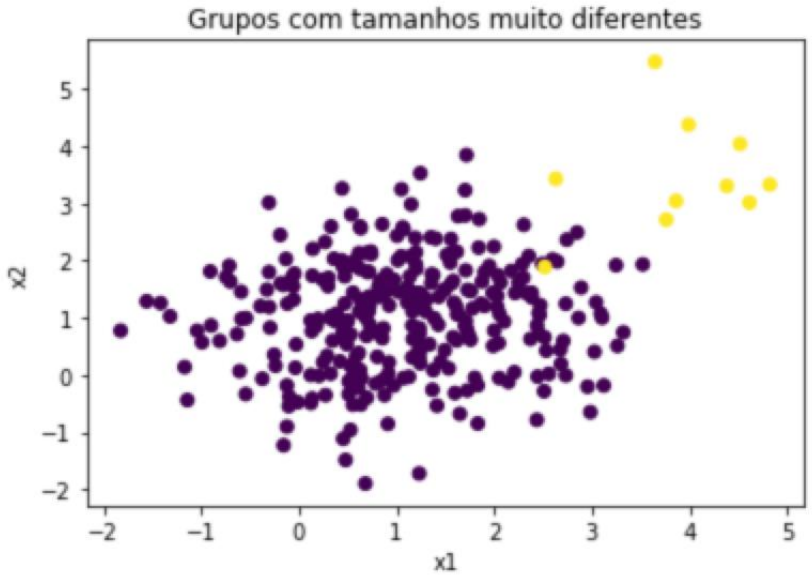
Depois da convergência o grupo grande foi dividido em dois.

Isso indica que K-means não consegue lidar bem com clusters que tenham tamanhos significativamente diferentes.

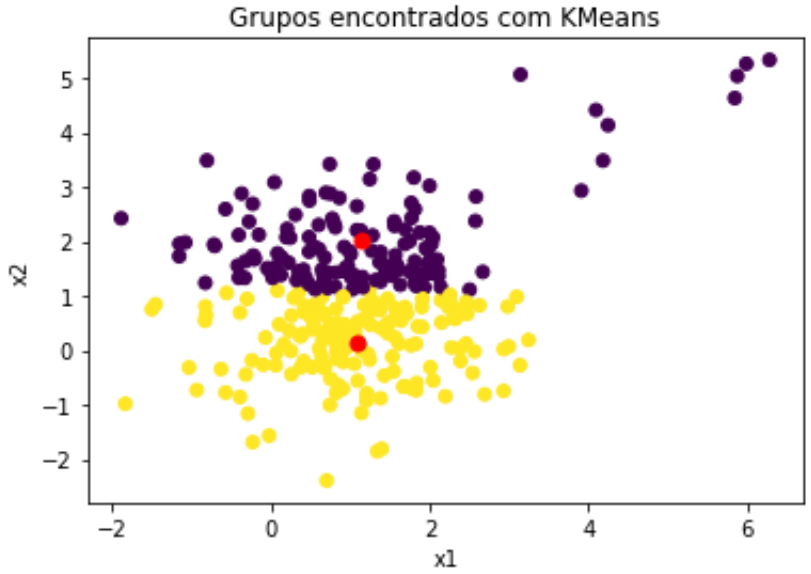
Algoritmo K-means - desvantagens

Tamanhos de clusters significativamente diferentes

Grupos originais



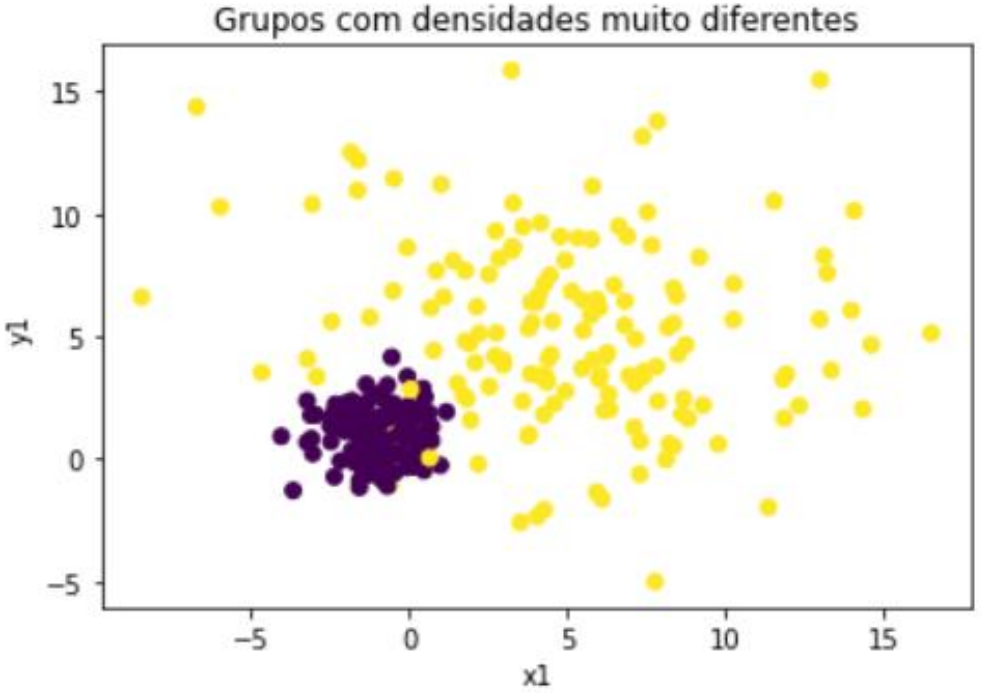
Grupos encontrados



- Depois da convergência o grupo grande foi dividido em dois.
- Isso indica que K-means não consegue lidar bem com clusters que tenham tamanhos significativamente diferentes.

Algoritmo K-means - desvantagens

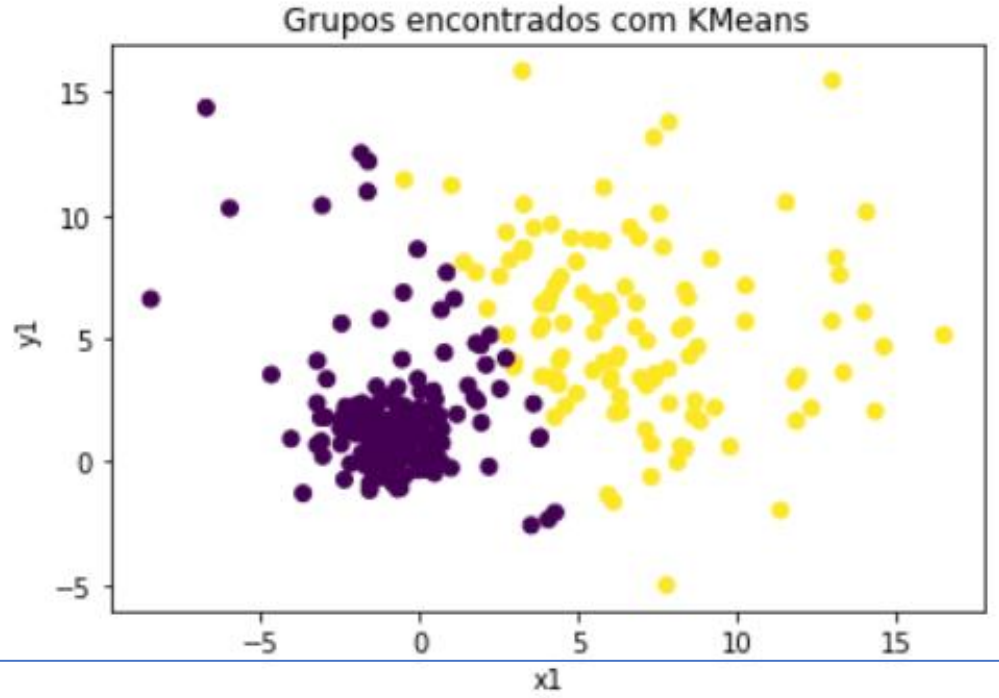
- *Densidade de clusters significativamente diferentes*
  - Foram gerados dois grupos de dados, sendo:
  - Cada um com 150 dados
  - Distribuições Gaussianas
  - Centros inicializados aleatoriamente




```
#gerando dados 2D com dois grupos com densidades diferentes
X,y = datasets.make_blobs(n_samples=[150,150],centers=[[1,1],[5,5]],
cluster_std=[1,4])
plt.scatter(X[:,0], X[:,1], c=y)
```

Algoritmo K-means - desvantagens

- *Densidades de clusters significativamente diferentes*
  - Agrupando os dados com K-means
  - 2 grupos
  - Centros inicializados aleatoriamente

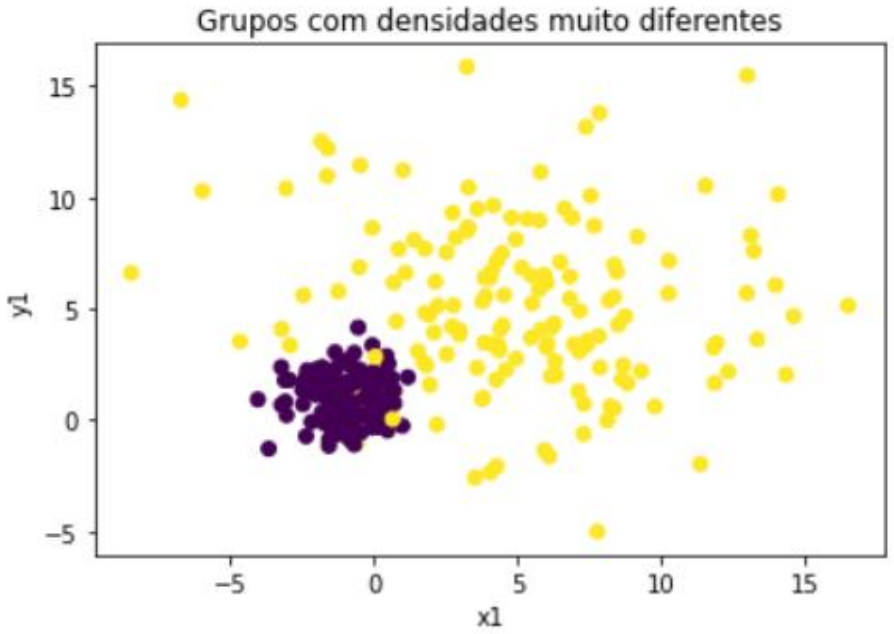



```
#agrupando com KMeans
kmeans = KMeans(n_clusters = 2, init = 'random')
kmeans.fit(X)
kmeans.cluster_centers_
y_pred = kmeans.labels_
plt.scatter(X[:,0], X[:,1], c=y_pred)
```

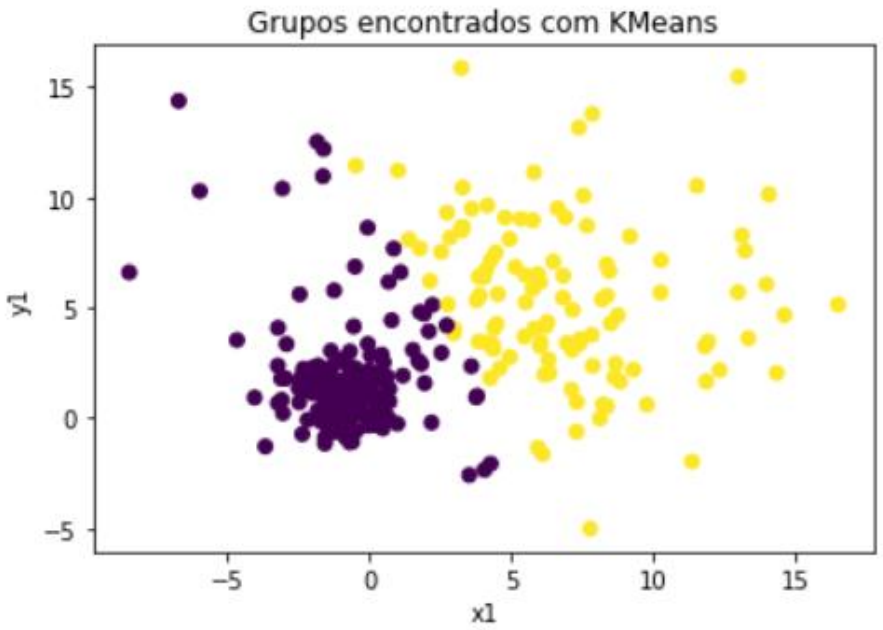
Algoritmo K-means - desvantagens

- *Densidades de clusters significativamente diferentes*

Grupos originais



Grupos encontrados

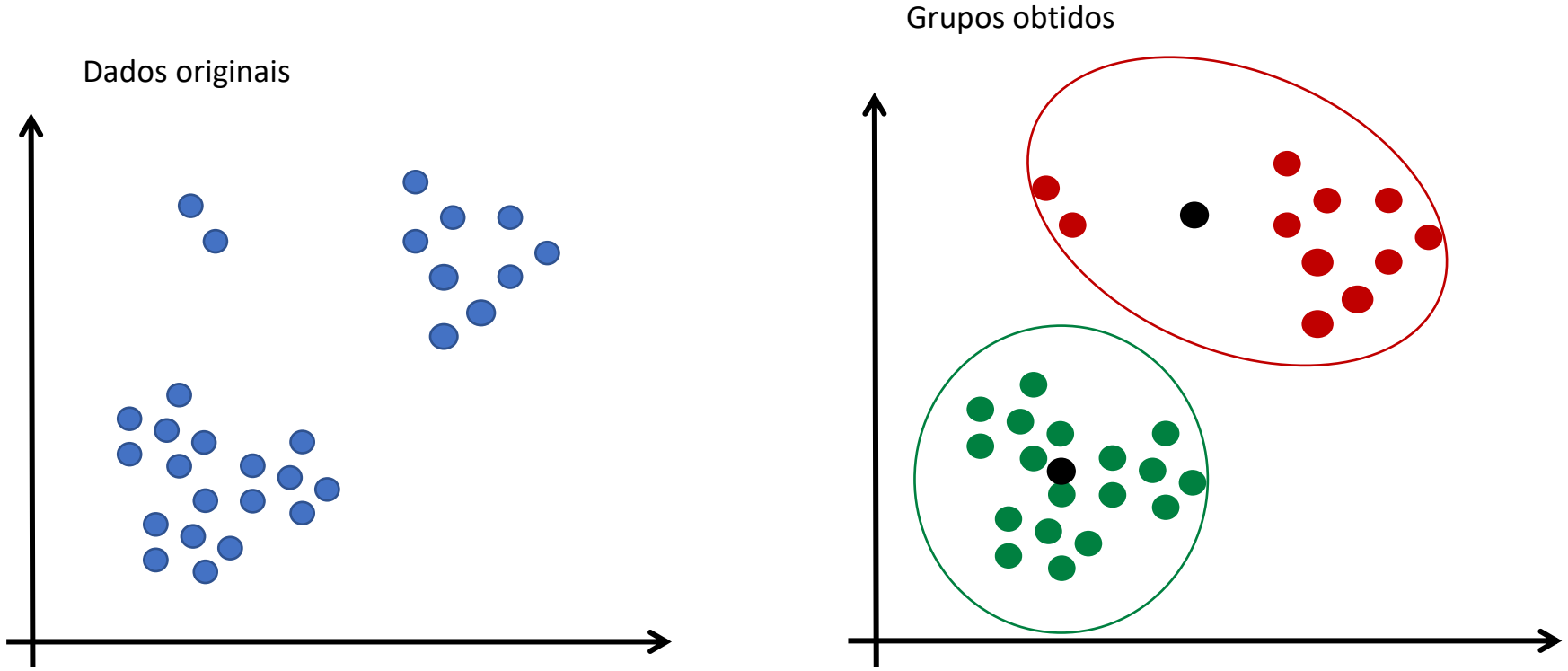


- Depois da convergência o grupo grande foi dividido em dois.
- Isso indica que K-means não consegue lidar bem com clusters que tenham tamanhos significativamente diferentes.

Algoritmo K-means - desvantagens

- *Sensível a outliers e ruído.*

- Exemplo: Agrupar os dados em dois grupos



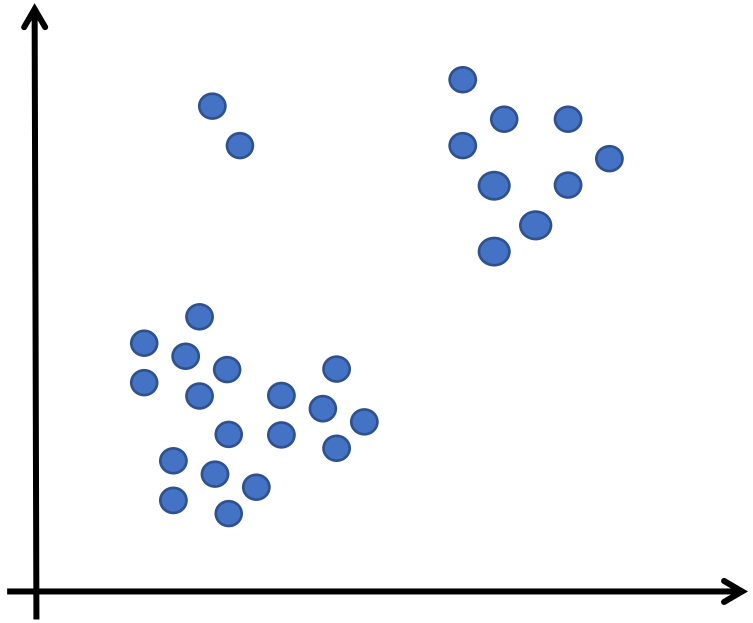
Algoritmo K-means - desvantagens

- *Sensível a outliers e ruído.*

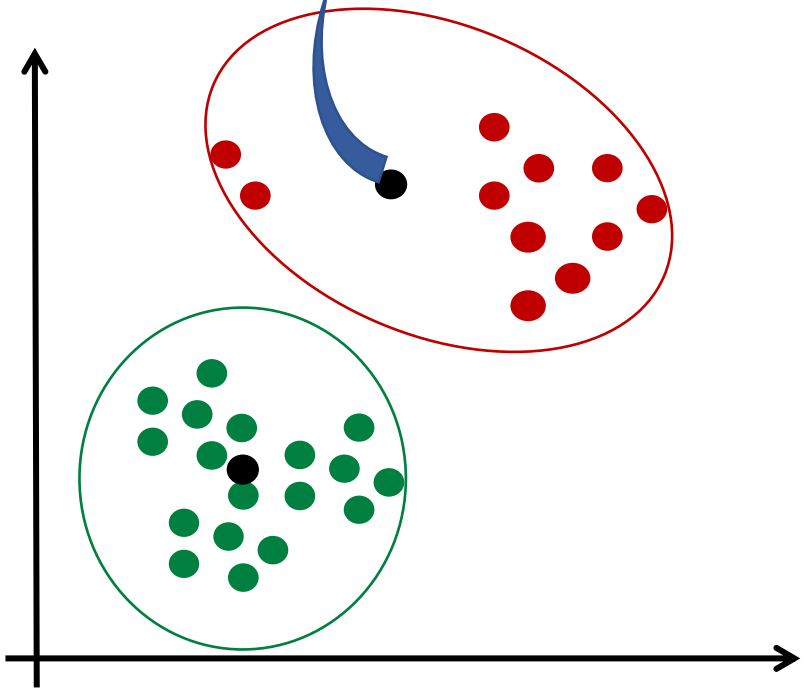
- Exemplo: Agrupar os dados em dois grupos

Centroide do grupo pode
não ser representativo
Os dados isolados “puxam” a
média para eles

Dados originais



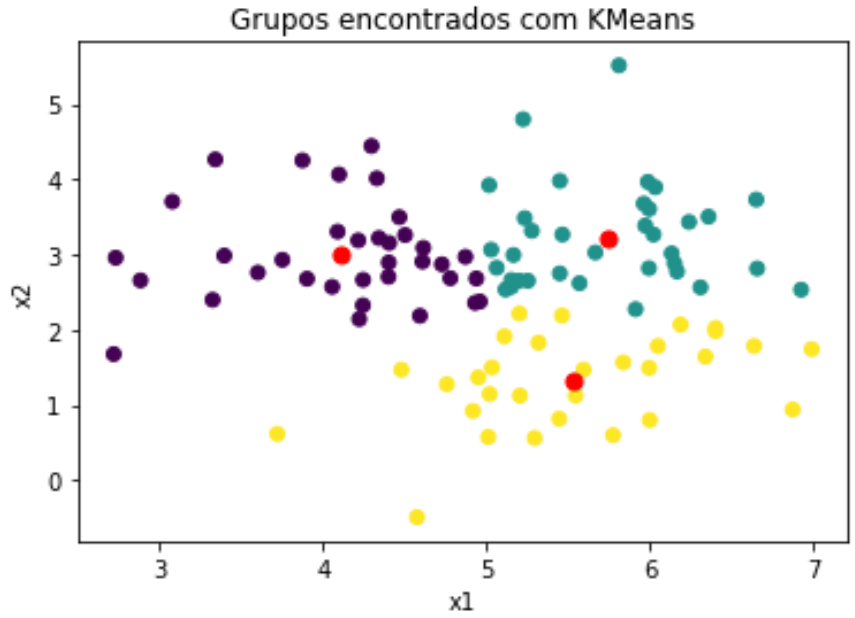
Grupos obtidos



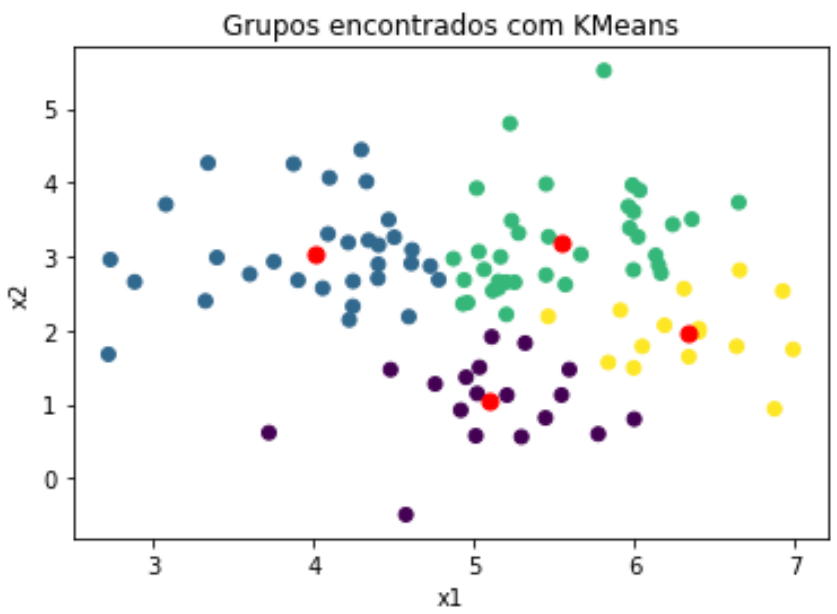
Algoritmo K-means - desvantagens

- *Sensível a outliers e ruído.*

3 grupos



4 grupos



No agrupamento com 4 grupos o grupo amarelo se dividiu gerando dois grupos menores

Ao aumentar o número de grupos a tendência é que os grupos fiquem menores contendo dados mais separados dos demais

Algoritmo K-means - desvantagens

- *Sensível a outliers e ruído.*

- Estratégias para tratar essa desvantagem:

  - Descarte todos os clusters “pequenos” (esses provavelmente formarão outliers).

  - Use um algoritmo k-medoide, em que um cluster é representado por um de seus pontos.

Algoritmo K-means - desvantagens

- *Sensível à definição da partição inicial*
- Partições iniciais diferentes podem levar o K-means a produzir agrupamentos finais diferentes, cada um correspondendo a um mínimo local diferente.

Estratégias para contornar essa desvantagem:

- Métodos de uma única rodada
  - Use um algoritmo sequencial para produzir estimativas iniciais para os centros.
  - Particione randomicamente o conjunto de dados em m subconjuntos e use as médias desses subconjuntos como as estimativas iniciais para os centros.

Algoritmo K-means - desvantagens

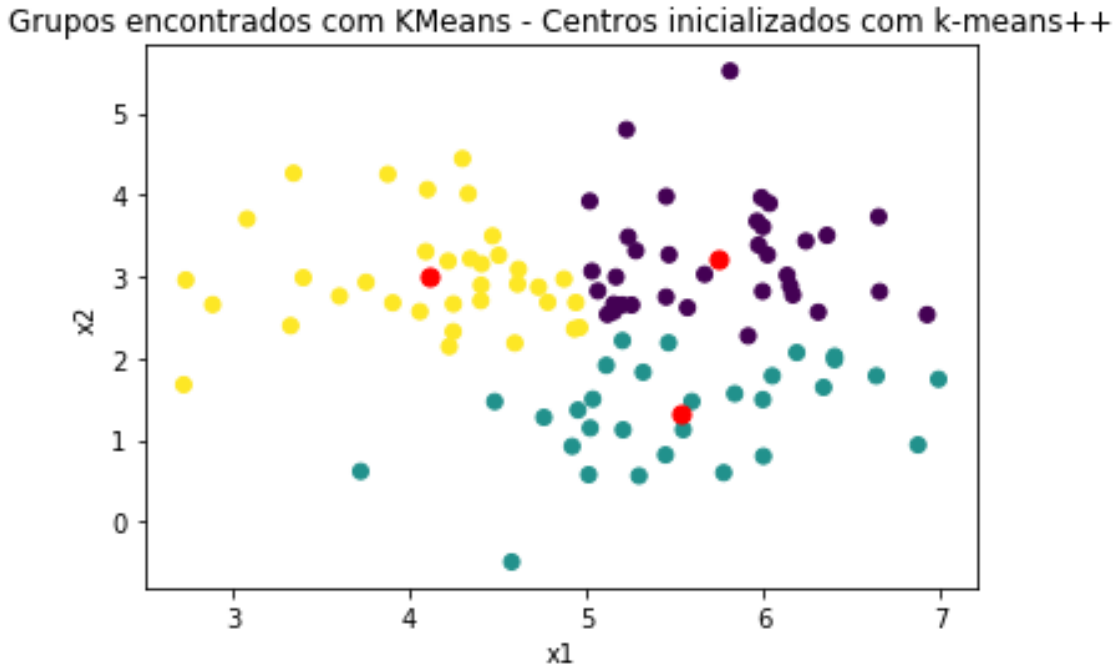
- *Sensível à definição da partição inicial*
- Inicialização do KMeans com k-means++
  - Princípio: Os centros iniciais devem estar o mais espalhados possível

- Algoritmo k-means++:
  1. Escolha um centro de cluster aleatoriamente entre todos os pontos.
  2. Para cada ponto x, calcule D(x), a distância entre x e o centro mais próximo que já foi escolhido
  3. Escolha um novo ponto aleatoriamente como novo centro, usando uma distribuição de probabilidade ponderada onde um ponto x é escolhido com probabilidade proporcional a $D(x)^2$.
  4. Repita os passos 2 e 3 até que k centros tenham sido escolhidos
  5. Aplique o algoritmo K-Means padrão usando os centros definidos

Algoritmo K-means - desvantagens

- *Sensível à definição da partição inicial*
- Inicialização do Kmeans com k-means++

```
# Executando KMeans e inicializando com k-means++
y_pred3 = KMeans(n_clusters=3,init='k-means++')
y_pred3.fit(X2)
y_pred3.cluster_centers_
```



Algoritmo K-means - desvantagens

- *O número de clusters m deve ser definido a priori.*

Estratégias para tratar essa desvantagem:

- Utilize operações de divisão, *merging* e descarte de clusters resultantes do k-means.
- Estime m por:
  - Rode um algoritmo sequencial várias vezes para limites de dissimilaridade Θ diferentes.
  - Crie o gráfico de Θ versus o número de clusters e identifique o maior *plateau* no gráfico e defina m com o valor correspondente ao *plateau*.

Algoritmo K-means - desvantagens

- *K-means não é adequado para dados com atributos nominais (categóricos) .*

- Estratégias para tratar essa desvantagem:

  - Use um algoritmo k-medoide.

Algoritmo K-Medoide

- Algoritmo semelhante ao K-means que minimiza a soma das dissimilaridades entre cada ponto e o **medoide** do cluster

- O medoide é um dado do conjunto que tem a menor dissimilaridade total a todos os outros membros do seu cluster

☑ Permite o uso de qualquer métrica de dissimilaridade no agrupamento

☑ O representante do cluster é mais "interpretável".

☑ É mais robusto a outliers

⊗ Complexidade $O(N^2KT)$

Algoritmo K-Medoide - versões

- Método Alternado

  - Inicialização: selecionar k dados do conjunto de dados como medoides, de forma randômica, usando heurística, ou k-medoide++
  - Atribuição: atribua cada elemento do conjunto de dados ao medoide mais próximo
  - Atualização: Identifique o novo medoide de cada cluster
  - Repita os passos de atribuição e atualização enquanto os medoides estiverem mudando ou até o número máximo de iterações ser atingido.

Algoritmo K-Medoide - versões

- Método PAM (Partitioning Around Medoids)

  - (BUILD) Inicialize (Inicialização gulosa):
    - Selecione o ponto (medoide) que minimiza a soma das distâncias dos outros ponto a ele.
    - Adicione, sucessivamente, um ponto por vez, que minimize o custo, até atingir o número desejado de medoides

  - (SWAP): Para todos os medoides já selecionados, calcule o custo de trocar esse medoide por qualquer outro ponto não medoide. Faça a troca que leva à maior diminuição no custo. Repita este passo até que não ocorram mais trocas.

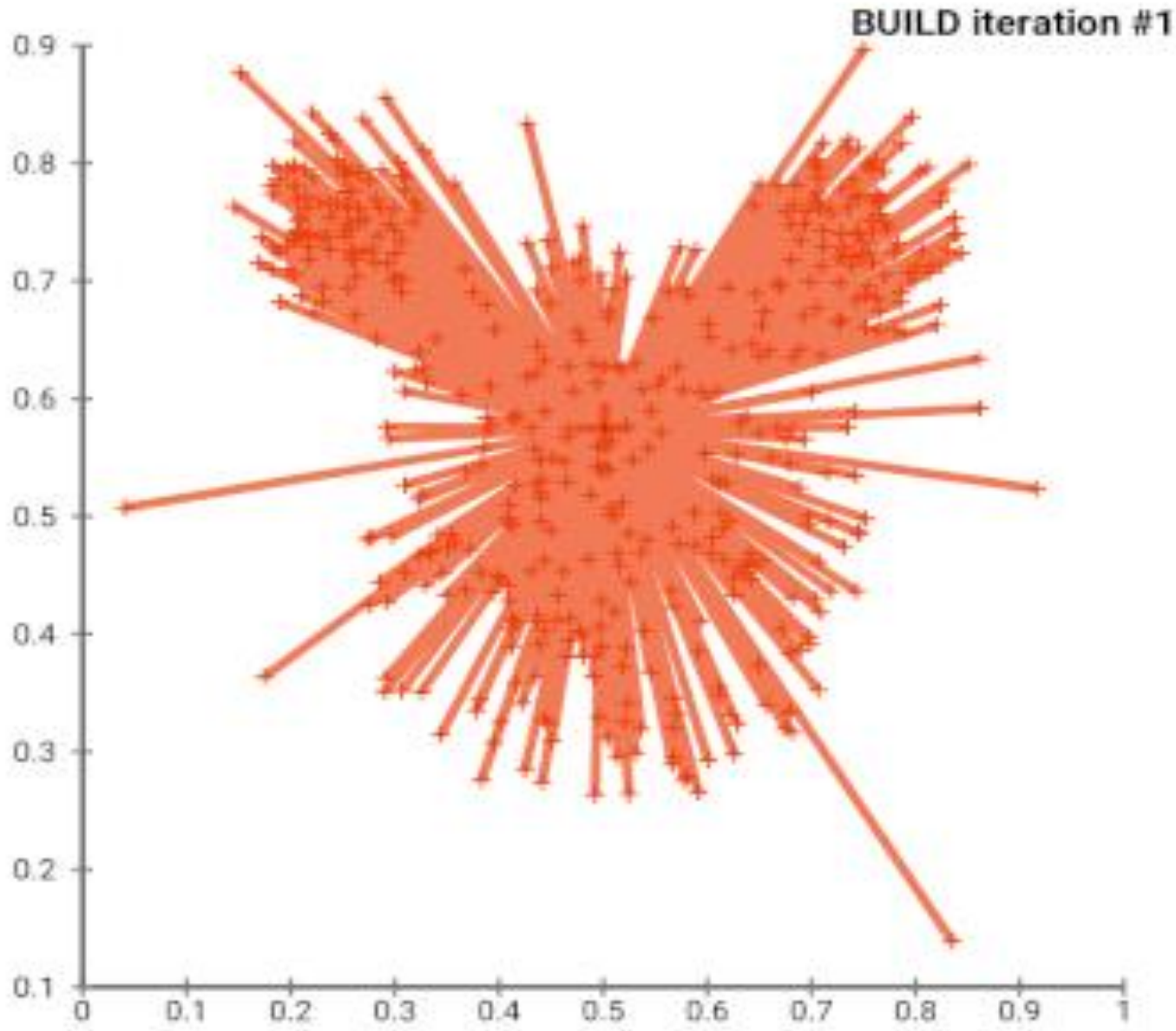
Algoritmo K-Medoide - implementação

- Implementação:

https://scikit-learn-extra.readthedocs.io/en/stable/

sklearn_extra.cluster.Kmedoids

class **sklearn_extra.cluster.KMedoids**(n_clusters=*8*, metric=*'euclidean'*, method=*'alternate'*, init=*'heuristic'*, max_iter=*300*, random_state=*None*)


BUILD iteration #1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1