



**FEELT - Aprendizagem de Máquina**

## **Trabalho 10 - K-means**

**Matheus Pelegrini Bucater**

**17 de Abril de 2025**

# 1 Introdução

Este relatório apresenta uma implementação do algoritmo K-Means para clusterização de dados. O objetivo é agrupar pontos de dados em clusters.

## 2 Metodologia

O algoritmo K-Means foi implementado em Python seguindo estas etapas:

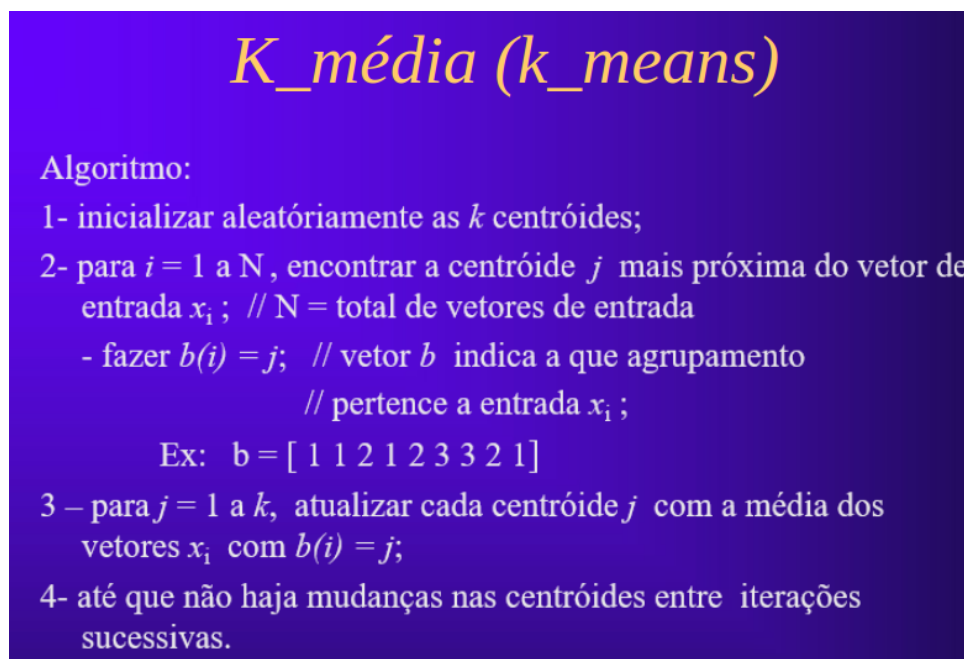
1. Inicialização aleatória dos centroides
2. Atribuição de pontos ao cluster mais próximo
3. Atualização dos centroides
4. Repetição até convergência

A métrica de erro utilizada foi o erro quadrático:

$$E = \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2$$

onde  $C_i$  é o cluster  $i$  e  $\mu_i$  seu centróide.

O algoritmo foi implementado usando o pseudo-código apresentado em sala de aula.



*K\_média (k\_means)*

Algoritmo:

- 1- inicializar aleatoriamente as  $k$  centróides;
- 2- para  $i = 1$  a  $N$ , encontrar a centróide  $j$  mais próxima do vetor de entrada  $x_i$ ; //  $N$  = total de vetores de entrada
  - fazer  $b(i) = j$ ; // vetor  $b$  indica a que agrupamento // pertence a entrada  $x_i$ ;

Ex:  $b = [1\ 1\ 2\ 1\ 2\ 3\ 3\ 2\ 1]$

- 3 – para  $j = 1$  a  $k$ , atualizar cada centróide  $j$  com a média dos vetores  $x_i$  com  $b(i) = j$ ;
- 4- até que não haja mudanças nas centróides entre iterações sucessivas.

Figure 1: Pseudo-código

## 2.1 Observação dos dados de entrada

Primeiramente os dados iniciais foram visualizados em um *scatterplot*.

Dessa forma, fica evidente 4 grupos de pontos bem comportados e separados, assim é possível notar o agrupamento do conjunto de entradas em  $K = 4$  clusters.

É possível, também, pensar em uma inicialização dos centróides mais eficiente, dividindo a imagem em 4 quadrantes. Assim, pode-se inicializar aleatoriamente os centróides dentro dos quadrantes definidos nos intervalos:

- Primeiro Quadrante => x:  $[0, 4.5]$ , y:  $[0, 4.5]$ ;
- Segundo Quadrante => x:  $[0, 4.5]$ , y:  $[5.5, 12]$ ;
- Terceiro Quadrante => x:  $[5.5, 12]$ , y:  $[0, 4.5]$ ;
- Quarto Quadrante => x:  $[5.5, 12]$ , y:  $[5.5, 12]$ ;

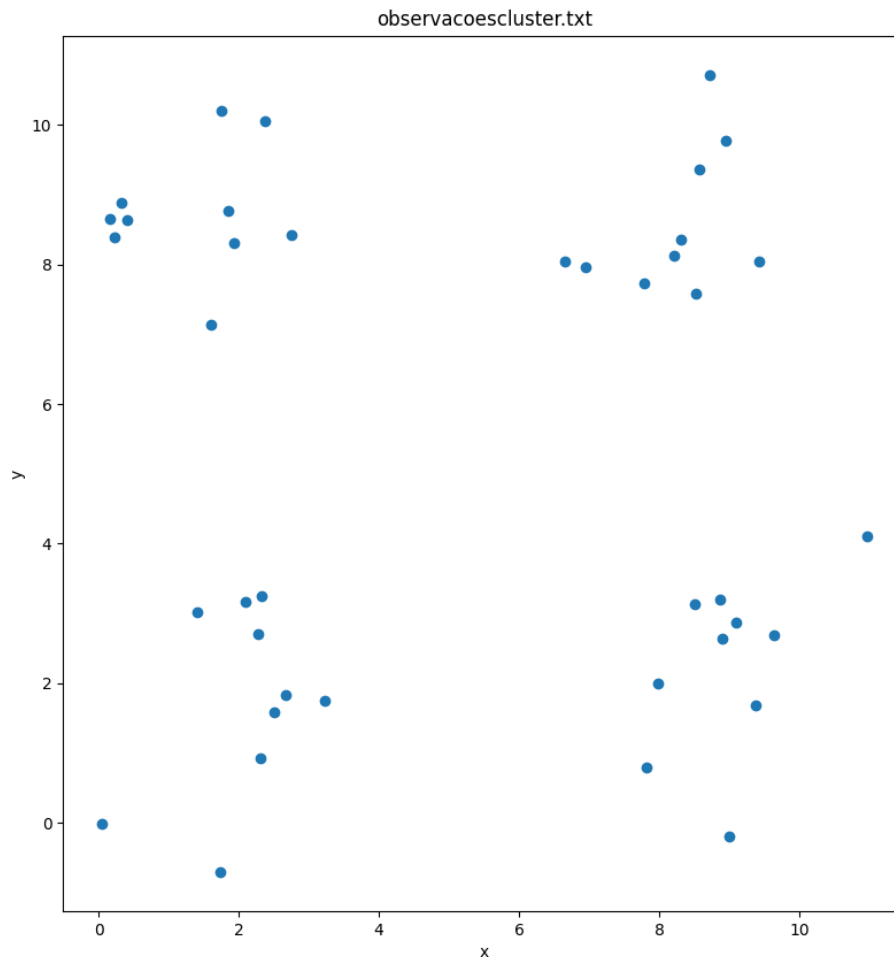


Figure 2: Scatterplot dos dados iniciais

## 3 Resultados

Vamos então observar o comportamento do algoritmo utilizando duas abordagens de inicialização dos centróides.

### 3.1 Inicialização completamente aleatória

Nessa abordagem os centróides foram inicializados de maneira inteiramente aleatória. A única restrição imposta foi a de que os centróides deveriam ser inicializados dentro do intervalo  $[0, 12]$  (tanto em  $x$  quanto em  $y$ ).

De maneira geral, a regra foi que o algoritmo funcionou bem inicializando aleatoriamente. Na maior parte dos casos os centróides convergiram para os quadrantes descritos. Entretanto houveram alguns casos em que o resultado divergiu do esperado e um desses casos será o analisado aqui.

#### 3.1.1 Curva do erro

Pode-se observar que em 5 iterações o algoritmo convergiu de acordo com a condição de parada descrita.

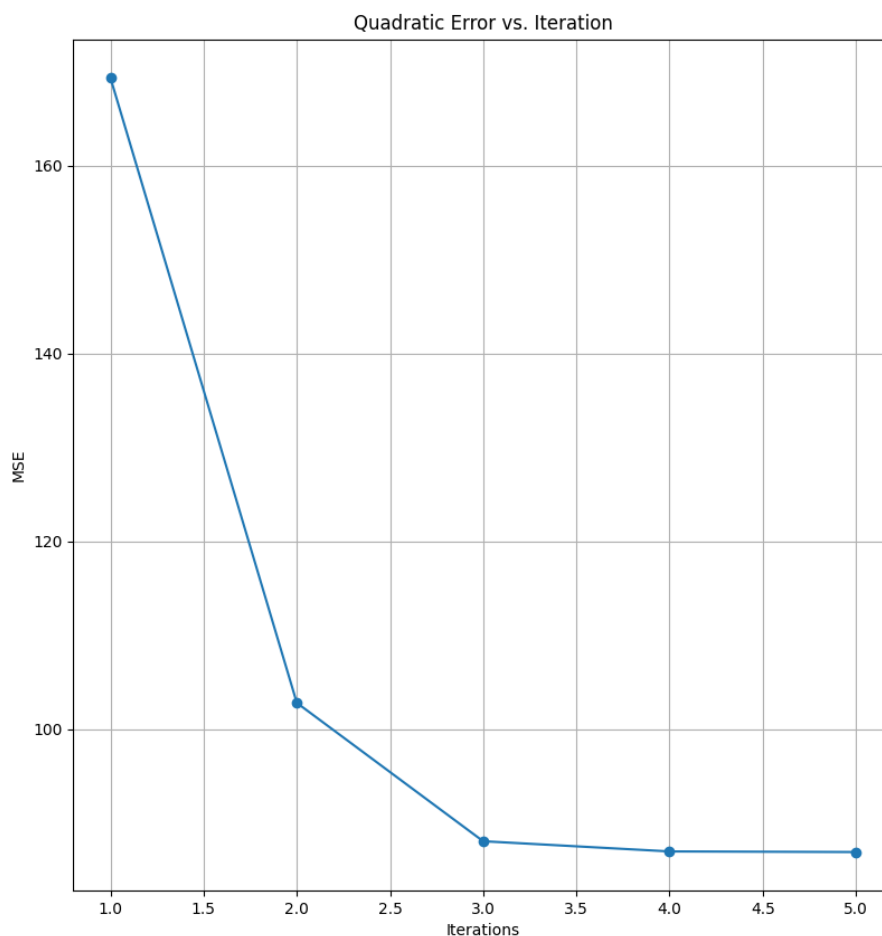


Figure 3: Curva do erro

#### 3.1.2 Convergência dos clusters

Pode-se observar que apesar da convergência do algoritmo, o agrupamento dos clusters diferiu do resultado esperado.

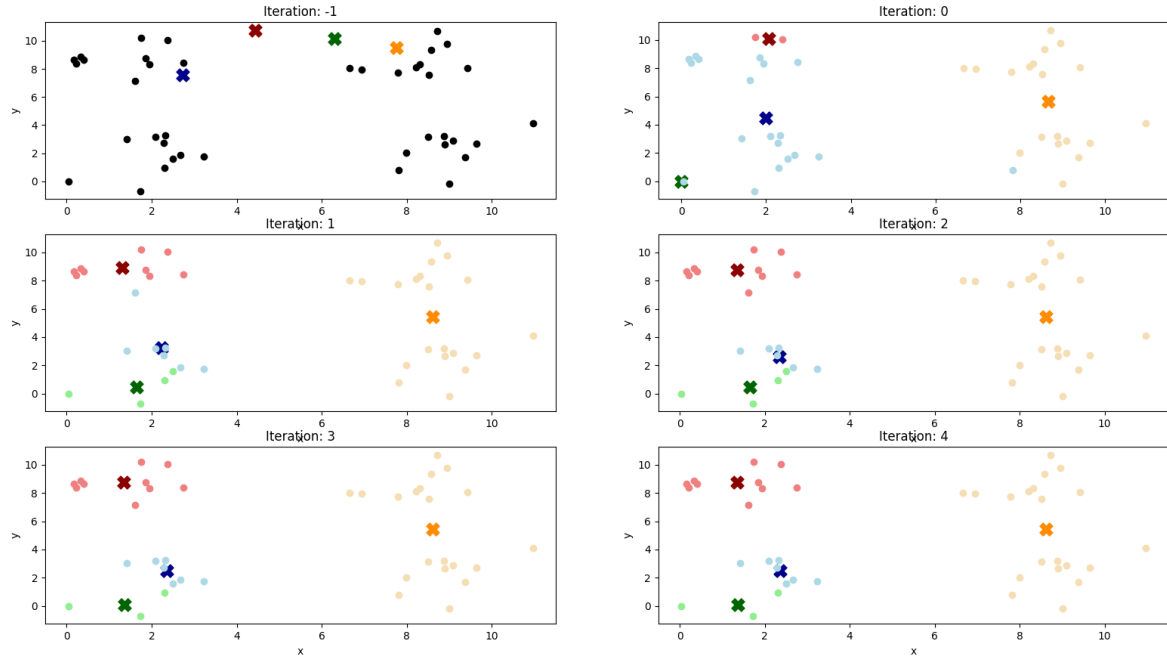


Figure 4: Evolução dos clusters e centróides

## 3.2 Inicialização aleatória controlada

Nessa abordagem os centróides foram inicializados de maneira aleatória dentro do intervalo dos quadrantes descritos.

### 3.2.1 Curva do erro

Pode-se observar que em 2 iterações o algoritmo convergiu de acordo com a condição de parada descrita.

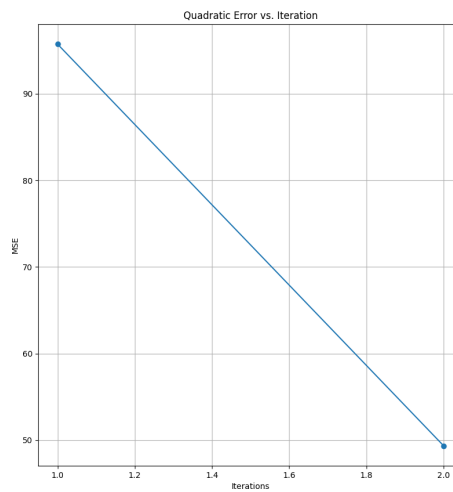


Figure 5: Curva do erro

### 3.2.2 Convergência dos clusters

Pode-se observar que o agrupamento dos clusters convergiu ao resultado esperado em apenas 2 iterações.

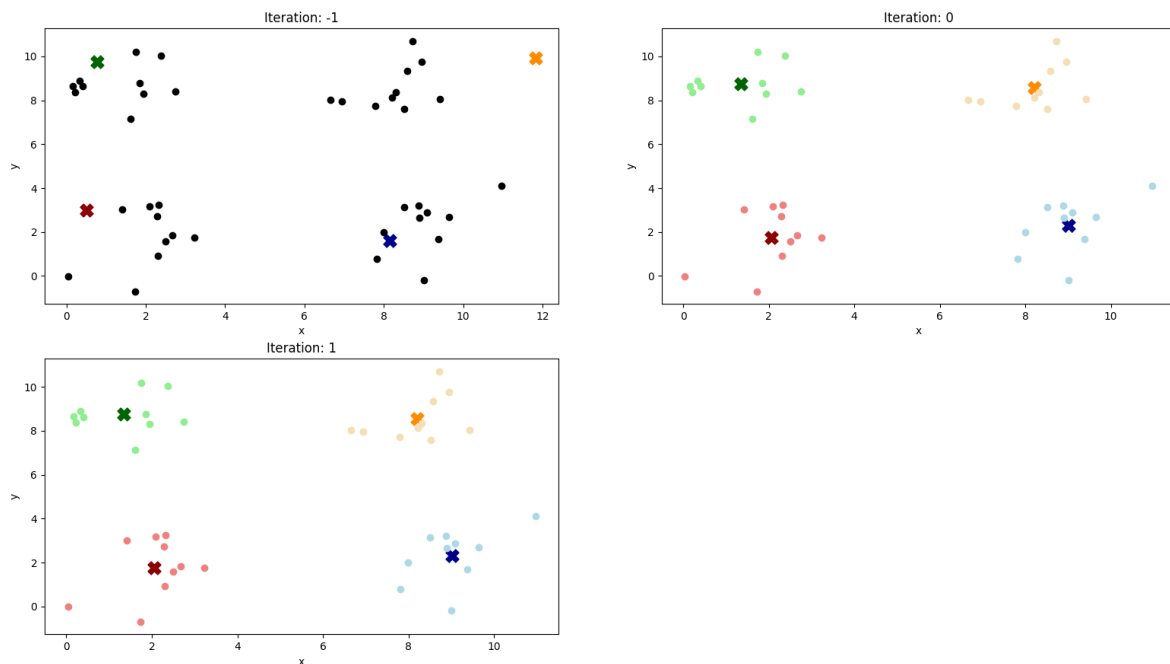


Figure 6: Evolução dos clusters e centróides

## 4 Conclusão

O uso do algoritmo K-means para o agrupamento dos dados de entrada em clusters convergiu, na maior parte das vezes, para um resultado ótimo, perto do esperado.

Ademais, a inicialização dos centróides de maneira mais controlado mostrou um ganho qualitativo e quantitativo da eficiência e resposta do algoritmo.

## 5 Apêndices

### 5.1 Código fonte

```
1 # main.py
2
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 from copy import copy
7 from math import sqrt, pow
8 from random import uniform
9
10 ch = []
11 cxsh = []
```

```

12 cysh = []
13
14 def read_data(input_file: str) -> (list[float], list[float]):
15     data: list[tuple[float, float]] = []
16     xs: list[float] = []
17     ys: list[float] = []
18     with open(input_file, "r") as file:
19         for line in file.readlines():
20             x, y = line.strip().split()
21             xs.append(float(x))
22             ys.append(float(y))
23     return xs, ys
24
25 def plot_clusters(
26     k: int,
27     clusters: list[int],
28     xs: list[float], ys: list[float],
29     cxs: list[float], cys: list[float],
30     title: str
31 ):
32
33     centroid_colors = ["darkred", "darkgreen", "darkblue", "darkorange"
34 ]
35     for i in range(k):
36         plt.scatter(cxs[i], cys[i], color=centroid_colors[i], label=f"
37             centroid-{i}", marker="X", s=150)
38
39     if clusters is None:
40         plt.scatter(xs, ys, color="black", label="points")
41
42     else:
43         xxs = []
44         yys = []
45         cluster_colors = ["lightcoral", "lightgreen", "lightblue", "
46             wheat"]
47
48         for i in range(K):
49             xx = [xs[j] for j in range(N) if clusters[j] == i]
50             xxs.append(xx)
51             yy = [ys[j] for j in range(N) if clusters[j] == i]
52             yys.append(yy)
53
54         for i, (xx, yy) in enumerate(zip(xxs, yys)):
55             plt.scatter(xx, yy, color=cluster_colors[i], label=f"
56                 cluster-{i}")
57
58     plt.xlabel("x")
59     plt.ylabel("y")
60     plt.title(title)
61     # plt.legend()
62     # plt.show()
63
64 def plot_clusters_history(k: int, xs: list[float], ys: list[float]):
65     M = len(ch)
66     subplot = f"{(M//2)+1}x{(M//2)+1}"
67
68     for i in range(M):
69         plt.subplot(int(f"{subplot}{i+1}"))

```

```

66         c = ch[i]
67         cxs = cxsh[i]
68         cys = cysh[i]
69
70         plot_clusters(k, c, xs, ys, cxs, cys, f"Iteration:_{i}_{i}")
71     plt.show()
72
73 def init_centroids(
74     k: int,
75     min_x: float,
76     max_x: float,
77     min_y: float,
78     max_y: float
79 ) -> (list[float], list[float]):
80
81     return [uniform(min_x, max_x) for _ in range(k)], [uniform(min_y,
82         max_y) for _ in range(k)]
83
84 def distance(xb: float, xa: float, yb: float, ya: float) -> float:
85     return sqrt(pow(xb - xa, 2) + pow(yb - ya, 2))
86
87 def k_means(
88     k: int,
89     xs: list[float], ys: list[float],
90     min_x: float, max_x: float, min_y: float, max_y: float,
91     init_centroids_method: str = "rand",
92     ranges_x: list[tuple[float, float]] = None, ranges_y: list[tuple[
93         float, float]] = None,
94     exact_x: list[float] = None, exact_y: list[float] = None
95 ) -> list[int]:
96
97     if init_centroids_method == "rand":
98         cxs, cys = init_centroids(k, min_x, max_x, min_y, max_y)
99     if init_centroids_method == "nrand":
100         cxs = []
101         cys = []
102         for i in range(k):
103             min_x, max_x = ranges_x[i]
104             min_y, max_y = ranges_y[i]
105             cx, cy = init_centroids(1, min_x, max_x, min_y, max_y)
106             cxs.append(*cx)
107             cys.append(*cy)
108     if init_centroids_method == "exact":
109         cxs = exact_x
110         cys = exact_y
111
112     itc = -1
113     n = len(xs)
114     cf = []
115     e = []
116
117     ch.append(None)
118     cxsh.append(copy(cxs))
119     cysh.append(copy(cys))
120
121     while True:
122         itc += 1
123         clusters = []

```



```

122
123     for i in range(n):
124         x = xs[i]
125         y = ys[i]
126
127         ds = []
128         for j in range(k):
129             cx = cxs[j]
130             cy = cys[j]
131             d = distance(x, cx, y, cy)
132             ds.append(d)
133             clusters.append(ds.index(min(ds)))
134
135     ce = 0
136     for i in range(n):
137         x = xs[i]
138         y = ys[i]
139         cluster = clusters[i]
140         cx = cxs[cluster]
141         cy = cys[cluster]
142         ce += distance(x, cx, y, cy)
143     e.append(ce)
144
145     c_changes = 0
146     for i in range(k):
147         x = [xs[j] for j in range(len(clusters)) if clusters[j] ==
148             i]
149         y = [ys[j] for j in range(len(clusters)) if clusters[j] ==
150             i]
151
152         med_x = 0
153         med_y = 0
154         if len(x) != 0:
155             med_x = sum(x) / len(x)
156         if len(y) != 0:
157             med_y = sum(y) / len(y)
158
159         if med_x != cxs[i] or med_y != cys[i]:
160             cxs[i] = med_x
161             cys[i] = med_y
162             c_changes += 1
163
164     ch.append(copy(clusters))
165     cxsh.append(copy(cxs))
166     cysh.append(copy(cys))
167
168     if c_changes == 0:
169         cf = clusters
170         break
171
172     plt.plot(range(1, len(e) + 1), e, marker='o')
173     plt.xlabel("Iterations")
174     plt.ylabel("MSE")
175     plt.title("Quadratic Error vs. Iteration")
176     plt.grid(True)
177     plt.show()
178
179     return cf

```

```

178
179 if __name__ == "__main__":
180     xs, ys = read_data("observacoescluster.txt")
181     plt.scatter(xs, ys)
182     plt.xlabel("x")
183     plt.ylabel("y")
184     plt.title("observacoescluster.txt")
185     plt.show()
186
187     # A partir da observacao do scatter, temos K = 4
188     K = 4
189     N = len(xs)
190
191     # Random
192     # clusters = k_means(K, xs, ys, 0, 12, 0, 12)
193
194     # Random dentro dos 4 quadrantes definidos
195     clusters = k_means(K, xs, ys, 0, 12, 0, 12, "nrand",
196         [(0, 4.5), (0, 4.5), (5.5, 12), (5.5, 12)],
197         [(0, 4.5), (5.5, 12), (0, 4.5), (5.5, 12)]
198     )
199
200     plot_clusters_history(K, xs, ys)

```