Professor Rodrigo Bossini
https://sites.google.com/site/professorrodrigobossini/

Data Science com Python 3
O Algoritmo K-Means

Muitas vezes encontrar subgrupos no conjunto de dados sob análise de modo que cada subgrupo (ou cluster, em inglês) possua itens similares entre si. Alguns exemplos de aplicação são

- agrupamento de clientes de uma loja online, a fim de direcionar ofertas mais apropriadas

- análise de imagens, com o objetivo de identificar regiões similares

- análise de documentos a fim de agrupar aqueles cujo conteúdo é semelhante

Há muitos algoritmos para agrupamento (do inglês, **clustering**). Neste material iremos estudar o algoritmo conhecido como K-Means.

Passo 1 (Entendendo o algoritmo K-Means) O algoritmo K-Means opera da seguinte forma.

1. Um número inteiro **k** é escolhido arbitrariamente. Esse é o número de subgrupos a serem encontrados. Essa escolha é fundamental para o bom funcionamento do algoritmo, o que requer conhecimento prévio sobre o conjunto de dados sob análise.

2. k instâncias do conjunto de dados são escolhidas aleatoriamente (há formas diferentes para se fazer essa escolha). Cada instância será a representante de cada um dos k grupos.

3. As demais instâncias são associadas ao grupo cuja instância representar for a mais similar (mais próxima) a ela.

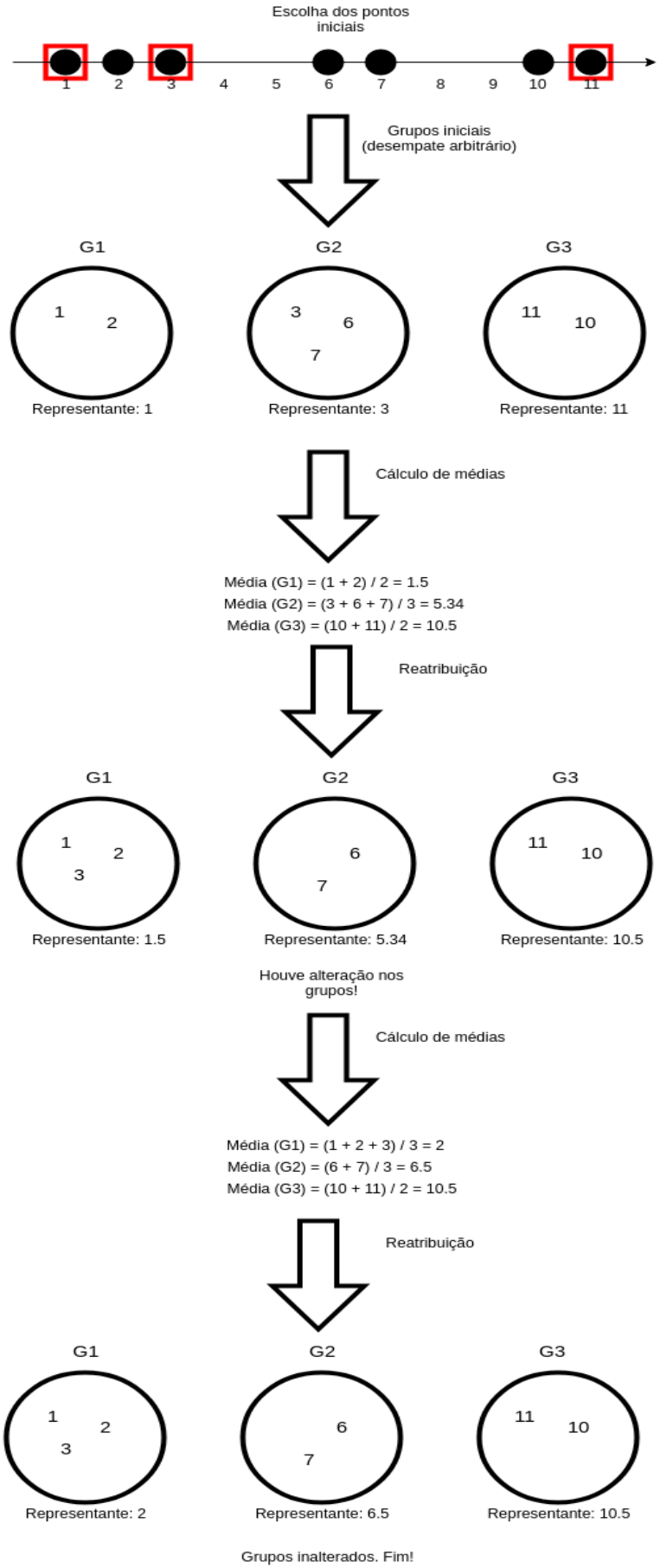
4. A média em cada grupo é calculada e ela passa a ser sua representante.

5. Cada instância é atribuída ao grupo cuja representante for a mais próxima a ela.

6. Se os grupos permanecerem iguais, o algoritmo termina. Caso contrário, ele é repetido a partir do passo 4.

A Figura 1.1 ilustra o funcionamento do algoritmo K-Means.

Figura 1.1


Escolha dos pontos iniciais
1 2 3 4 5 6 7 8 9 10 11
Grupos iniciais (desempate arbitrário)
G1
1 2
Representante: 1
G2
3 6 7
Representante: 3
G3
11 10
Representante: 11
Cálculo de médias
Média (G1) = (1 + 2) / 2 = 1.5
Média (G2) = (3 + 6 + 7) / 3 = 5.34
Média (G3) = (10 + 11) / 2 = 10.5
Reatribuição
G1
1 2 3
Representante: 1.5
G2
6 7
Representante: 5.34
G3
11 10
Representante: 10.5
Houve alteração nos grupos!
Cálculo de médias
Média (G1) = (1 + 2 + 3) / 3 = 2
Média (G2) = (6 + 7) / 3 = 6.5
Média (G3) = (10 + 11) / 2 = 10.5
Reatribuição
G1
1 2 3
Representante: 2
G2
6 7
Representante: 6.5
G3
11 10
Representante: 10.5
Grupos inalterados. Fim!

Passo 2 (Funções que já utilizamos e que serão necessárias) Os dados com os quais trabalharemos terão, eventualmente, muitas dimensões. Cada instância pode ser, em geral, representada como um vetor. Por isso, faremos isso de funções vistas na aula sobre álgebra linear.

2.1 Dados n vetores, podemos calcular sua média como mostra a Listagem 2.1.1.

Listagem 2.1.1

```
#multiplica um vetor por um escalar
def scalar_multiply (escalar, vetor):
    return [escalar * i for i in vetor]

#soma n vetores
def vector_sum (vetores):
    resultado = vetores[0]
    for vetor in vetores[1:]:
        resultado = [resultado[i] + vetor[i] for i in range(len(vetor))]
    return resultado

# calcula a média de n vetores
def vector_mean(vetores):
    return scalar_multiply(1/len(vetores), vector_sum(vetores))
```

2.2 Com as funções da Listagem 2.2.1 podemos calcular a distância entre dois vetores.

Listagem 2.2.1

```
#produto escalar
def dot (v, w):
    return sum(v_i * w_i for v_i, w_i in zip(v, w))

# subtração de vetores
def vector_subtract (v, w):
    return [v_i - w_i for v_i, w_i in zip (v, w)]

#soma dos quadrados
def sum_of_squares (v):
    return dot (v, v)

#distância ao quadrado
def squared_distance (v, w):
    return sum_of_squares (vector_subtract(v, w))
```

Passo 3 (Implementação do algoritmo KMeans) A Listagem 3.1 mostra uma possível implementação para o algoritmo KMeans.

Listagem 3.1

```
class KMeans:
    def __init__ (self, k, means = None):
        self.k = k
        self.means = means

    def classify (self, ponto):
        return min (range (self.k), key = lambda i: squared_distance(ponto, self.means[i]))

    def train (self, pontos):
        #escolha de k elementos
        #self.means = random.sample (pontos, self.k)
        #nenhuma atribuição, para começar
        assignments = None
        while True:
            #associa cada instância a um inteiro 0 <= i < k
            new_assignments = list(map (self.classify, pontos))

            #se não houver mudança, termina
            if new_assignments == assignments:
                return
            #atribuição atual se torna a nova
            assignments = new_assignments
            #cálculo das novas médias
            for i in range (self.k):
                #pontos associados ao agrupamento i
                #note que pontos e assignments estão na ordem
                #por exemplo pontos = [1, 2, 3]  e assignments = [1, 2, 2]
                #indicam que a primeira instância está no grupo 1 e as demais
                # no grupo 2
                i_points = [p for p, a in zip (pontos, assignments) if a == i]
                # tem alguém nesse grupo?
                if i_points:
                    self.means[i] = vector_mean (i_points)
```

Passo 4 (Testando o algoritmo) A função da Listagem 4.1 mostra um teste inicial para o KMeans, usando os dados do exemplo da Figura 1.1.

Listagem 4.1

```
def test_k_means ():
    dados = [[1], [3], [6], [7], [10], [11]]
    kmeans = KMeans(3, [[1], [3], [11]])
    kmeans.train(dados)
    print (kmeans.means)
```

Exercício

Adapte o algoritmo para que seja possível verificar cada um dos grupos.

Bibliografia

GRUS, J. **Data Science from Scratch: First Principles with Python.** 1st ed. O'Reilly Media, 2015.