Lista 4 Parte 1

Análise de dados de microarray
e técnicas de clusterização

Pedro Foletto Pimenta - 00229778
INF05018 - Biologia Computacional - Turma: U (2019/2)
Prof. Márcio Dorn

K-Means

- Método de clustering
  - Particiona um conjunto de vetores em K grupos/classes
- **Entradas:**
  - Vetores a serem classificados
  - K (número de classes)
- **Saídas:**
  - Posição final dos K centróides
  - Classe atribuída a cada um dos vetores de entrada

K-Means: Pseudo-código

1. Gerar centróides iniciais
2. Gerar classificação inicial dos vetores
3. Enquanto os centróides não convergirem:
   a. atualizar posição dos centróides
   b. classificar os vetores de acordo com as distâncias para cada centróide
4. Retornar os centróides finais e a classe de cada vetor

Entrada

Base de dados formada por perfis de **expressão gênica de pacientes com leucemia aguda**

- Total de **72 amostras**
- Cada amostra consiste da expressão de **7129 genes (features)**
- **Rótulos** das amostras (dois tipos distintos de leucemia):
  - **`ALL`** (acute lymphoidleukemia)
  - **`AML`** (acute myeloid leukemia)
- Expressão gênica com valores no intervalo [-3.5, 3.5]
  - Então **foi aplicada uma normalização** para [0, 1]

K-Means: Inicialização dos centróides

- Muito importante para o bom funcionamento do algoritmo
- Implementação:
  - **centróide = (média dos pontos) + 0.1*(vetor aleatório)**

K-Means: Implementação

- **Python 3**
- Bibliotecas utilizadas:
  - **random** (para gerar os centróides iniciais)
  - **pandas** (para carregar os dados do arquivo csv)
  - **numpy** (para manipular os vetores e matrizes)

Implementação

```python
# retorna as classes de cada ponto e os centroides de cada classe
def k_means(k, points):
    # points[num_points, num_dim] : pontos a classificar em k grupos
    # k : numero de grupos para classificar os pontos
    # classes[num_points] : classificacao dada a cada ponto de points
    # centroids[k, num_dim] : centroides de cada classe
    # old_centroids[k, num_dim] : centroides de cada classe na ultima iteracao (para criterio de parada)

    num_points, num_dim = np.shape(points)

    # gerar centroides iniciais aleatoriamente mas perto da media dos pontos
    centroids = [np.mean(points, axis=0) + 0.1 * np.random.rand(num_dim) for i in range(k)]
    old_centroids = np.zeros((k, num_dim))

    # classificacao inicial dos pontos
    classes = classify_points(points, centroids, num_points, k)

    # iterar ate convergencia
    while(not np.array_equal(centroids,old_centroids)): # enquanto os centroides nao convergirem
        old_centroids = centroids
        # atualiza centroides
        centroids = update_centroids(points, classes, k)
        # atualiza classificacao de cada ponto
        classes = classify_points(points, centroids, num_points, k)

    return classes, centroids
```

Implementação

```
# retorna um numpy array onde cada posiçao i contem a classe atribuida ao ponto i
def classify_points(points, centroids, num_points, k):
    classes = np.zeros(num_points)
    for i in range(num_points):
        p = points[i]
        min_dist = 99999
        for j in range(k):
            c = centroids[j]
            dist = np.linalg.norm(p-c) # distancia entre ponto p e centroide c
            if(dist < min_dist): # atualiza classe do centroide mais perto do ponto p
                min_dist = dist
                classes[i] = j
    return classes
```

Implementação

```
# retorna um numpy array contendo a posicao dos k centroides
def update_centroids(points, classes, k):

    num_points, num_dim = np.shape(points)
    centroids = np.zeros((k, num_dim))

    for i in range(k):
        if(points[classes==i].size == 0):
            # classe vazia -> reinicia cetroide aleatoriamente perto da media dos pontos
            centroids[i] = np.mean(points, axis=0) + 0.1 * np.random.rand(k, num_dim)
        else:
            # ajusta centroide para a media dos pontos contidos na sua classe
            centroids[i] = np.mean(points[classes==i], axis=0)
    return centroids
```

Implementação

```
# retorna o score de um agrupamento de pontos/vetores
# OBS: so funciona com clustering em dois grupos (k=2 : ALL e AML)
def get_clustering_score(classes, labels):

    assert(np.shape(classes )== np.shape(labels))
    vecloko = np.ones(np.shape(classes)) # vetor de 1s para calculo do score

    # caso A -> ALL: 0, AML: 1
    matches_A = np.sum(vecloko[np.logical_and(labels=='ALL',classes==0)])
    matches_A = matches_A + np.sum(vecloko[np.logical_and(labels=='AML',classes==1)])

    # caso B -> ALL: 1, AML: 0
    matches_B = np.sum(vecloko[np.logical_and(labels=='ALL',classes==1)])
    matches_B = matches_B + np.sum(vecloko[np.logical_and(labels=='AML',classes==0)])

    # score = matches / num_points
    score = float(max(matches_A, matches_B)) / np.size(classes)

    return score
```

Implementação: "main"

```
# carregar dados do arquivo csv
df = pandas.read_csv('leukemia_big.csv', header=None)

# get data from dataframe
labels = np.array(df.iloc[0].values) # get labels (first row)
df = df.drop(0) # remove labels from dataframe
df = df.T # transpose data
points = (df.values).astype(np.float) # convert strings to floats and put it in a numpy array
points = normalize_points(points) # normalize to [0,1] interval

k = 2
classes, centroids = k_means(k, points)
score = get_clustering_score(classes, labels)
print("\nResultado para K=2")
print("...classes: "+str(classes))
print("...centroides: "+str(centroids))
print("...score: " + str(score))

k = 3
classes, centroids = k_means(k, points)
print("\nResultado para K=3")
print("...classes: "+str(classes))
print("...centroides: "+str(centroids))
```

Resultado

```
pfpimenta@s-72-206-08:~/biocomp2019/lista4parte1$ python e4-1.py

Resultado para K=2
...classes: [ 0.  0.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  0.  1.  0.  0.  0.  0.  1.  0.  0.  1.  0.  0.  1.  0.  0.  0.  1.  1.
  1.  1.  1.  1.  0.  0.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  0.  1.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
...centroides: [[ 0.43677646  0.48381527  0.42909435 ...,  0.62599264  0.53793563
   0.47403535]
 [ 0.40292162  0.43025285  0.37926059 ...,  0.60595592  0.46441134
   0.39788242]]
...score: 0.819444444444

Resultado para K=3
...classes: [ 2.  2.  1.  1.  1.  2.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  2.
  2.  1.  0.  0.  0.  0.  0.  0.  2.  2.  2.  0.  1.  0.  2.  2.  2.  1.
  2.  2.  1.  2.  2.  2.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  2.  2.  1.  0.  1.  2.  0.  2.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
...centroides: [[ 0.4695667   0.53189634  0.44329266 ...,  0.59728988  0.57485418
   0.47372337]
 [ 0.39063346  0.41413059  0.35606303 ...,  0.60474188  0.44405319
   0.37597481]
 [ 0.40671986  0.4347486   0.42709527 ...,  0.6466195   0.49682799
   0.47132305]]
```

Resultado (em uma das execuções)

K = 2

	ALL	AML
classe 0	31	2
classe 1	16	23

score: 75%

K = 3

	ALL	AML
classe 0	23	3
classe 1	2	15
classe 2	22	7

Resultado

- K-means é um método estocástico
  - resultado depende da inicialização dos centróides
  - cada execução produz um resultado diferente
  - score médio de 100 execuções: **0.7225 (72.5%)**
    - separação razoável

Lista 4 Parte 1

Análise de dados de microarray
e técnicas de clusterização

Pedro Foletto Pimenta - 00229778
INF05018 - Biologia Computacional - Turma: U (2019/2)
Prof. Márcio Dorn