K-Means em Julia

February 12, 2016

1 Trabalho de Implementação

1.1 INF2912 - Otimização Combinatória

1.1.1 Prof. Marcus Vinicius Soledade Poggi de Aragão

1.1.2 2015-2

1.1.3 Ciro Cavani

BigData / Globo.com Algoritmos de clusterização.

1.2 Conteúdo

Esse notebook tem o desenvolvimento e avaliação do algoritmo iterativo do K-Means (algoritmo de Lloyd).

A avaliação do algoritmo é baseada em um mapeamento entre a maioria dos itens que foram atribuídos a um determinado cluster e o correspondente os valores verdadeiros gerados nesse cluster.

O K-Means teve resultados muito bons.

1.3 Dataset

```
In [1]: include("../src/clustering.jl")
        import Inf2912Clustering
        const Clustering = Inf2912Clustering
```

```
Out[1]: Inf2912Clustering
```

```
In [2]: dataset = Clustering.dataset_tiny()
        Clustering.summary(dataset)
        sleep(0.2)
```

```
Number of Groups: 3
Number of Features: 16
Number of Features (group): 3
Probability of Activation: 0.8
Number of Objects (total): 100
Number of Objects per Group (min): 20
Number of Objects per Group (max): 40
Number of Objects in 1: 40
Number of Objects in 2: 33
Number of Objects in 3: 27
```

1.4 K-Means

Consiste em executar o algoritmo K-means determinar os pontos centrais de cada grupo e classificar cada objeto como sendo do grupo com ponto central mais próximo

https://en.wikipedia.org/wiki/K-means_clustering

1.4.1 Algoritmo Iterativo

1. Choose k cluster centers randomly generated in a domain containing all the points,
2. Assign each point to the closest cluster center,
3. Recompute the cluster centers using the current cluster memberships,
4. If a convergence criterion is met, stop; Otherwise go to step 2.

In [3]:
```
"Algoritmo de clusterização K-Means (algoritmo de Lloyd)."
function kmeans(dataset, k; maxiters=20)
    inputs = map(v -> float(v[1]), dataset.data)

    # inicialização com amostragem sem reposição de k objetos como centros iniciais
    means = map(i -> inputs[i], randperm(length(inputs))[1:k])

    # função que calcula o índice do centro de menor distância de v
    classify(v) = indmin(map(c -> norm(c - v), means))

    assignments::Array{Int,1} = []
    iters = 0

    while iters < maxiters
        iters += 1

        # calcula o centro associado a cada objeto
        new_assignments = map(classify, inputs)

        # encerra o processamento se não tiver mudança com a última iteração
        assignments == new_assignments && break

        # recalcula os centros como a média dos pontos do último agrupamento
        assignments = new_assignments

        #println("Centros ", iters, ": ", means)
        #println("Agrupamentos ", iters, ": ", new_assignments)

        for i=1:k
            # lista todos os objetos do i-ésimo agrupamento
            i_points = map(ii -> inputs[ii], findin(assignments, i))

            isempty(i_points) && continue
            means[i] = mean(i_points)
        end
    end

    assignments
end

kmeans(dataset, 3)
```

Out[3]:
```
100-element Array{Int64,1}:
 3
 2
 1
 1
 3
```

```
 2
 3
 2
 3
 3
 2
 3
 1
 ⋮
 2
 1
 2
 3
 2
 3
 3
 2
 3
 3
 1
 3
```

```
In [4]: import Clustering.mapping

        "Algoritmo de clusterização K-Means (algoritmo de Lloyd) \
        aproximado para os grupos pré-definidos do dataset."
        function kmeans_approx(dataset, k)
            assignments = kmeans(dataset, k)
            centermap = mapping(dataset, assignments, k)
            map(c -> centermap[c], assignments)
        end

        let
            k = dataset.groups
            prediction = kmeans_approx(dataset, k)
            Clustering.evaluation_summary(dataset, prediction; verbose=true)
            sleep(0.2)
        end
```

```
Matriz de Confusão:

[36 2 2
 2 31 0
 0 2 25]

Tamanho: 100
Acertos: 92
Erros: 8
Accuracy: 92.0%

Cluster 1

Tamanho: 40
Accuracy: 94.0%
```

```
Precision: 94.74%
Recall: 90.0%
F-score: 0.92

Acerto positivo: 36 (90.0%)
Acerto negativo: 58 (96.67%)
Falso negativo: 4 (50.0%)
Falso positivo: 2 (25.0%)

Cluster 2

Tamanho: 33
Accuracy: 94.0%
Precision: 88.57%
Recall: 93.94%
F-score: 0.91

Acerto positivo: 31 (93.94%)
Acerto negativo: 63 (94.03%)
Falso negativo: 2 (25.0%)
Falso positivo: 4 (50.0%)

Cluster 3

Tamanho: 27
Accuracy: 96.0%
Precision: 92.59%
Recall: 92.59%
F-score: 0.93

Acerto positivo: 25 (92.59%)
Acerto negativo: 71 (97.26%)
Falso negativo: 2 (25.0%)
Falso positivo: 2 (25.0%)
```

```
In [5]: Clustering.test_dataset("small", kmeans_approx)
        sleep(0.2)
```

```
0.017226 seconds (43.69 k allocations: 21.664 MB, 38.59% gc time)
Matriz de Confusão:

[408 0 0
 0 304 0
 0 0 288]

Tamanho: 1000
Acertos: 1000
Erros: 0
Accuracy: 100.0%

Cluster 1

Tamanho: 408
Accuracy: 100.0%
Precision: 100.0%
```

```
Recall: 100.0%
F-score: 1.0

Acerto positivo: 408 (100.0%)
Acerto negativo: 592 (100.0%)
Falso negativo: 0 (NaN%)
Falso positivo: 0 (NaN%)

Cluster 2

Tamanho: 304
Accuracy: 100.0%
Precision: 100.0%
Recall: 100.0%
F-score: 1.0

Acerto positivo: 304 (100.0%)
Acerto negativo: 696 (100.0%)
Falso negativo: 0 (NaN%)
Falso positivo: 0 (NaN%)

Cluster 3

Tamanho: 288
Accuracy: 100.0%
Precision: 100.0%
Recall: 100.0%
F-score: 1.0

Acerto positivo: 288 (100.0%)
Acerto negativo: 712 (100.0%)
Falso negativo: 0 (NaN%)
Falso positivo: 0 (NaN%)
```

```
In [6]: Clustering.test_dataset("large", kmeans_approx)
        sleep(0.2)
```

```
0.190526 seconds (448.75 k allocations: 216.802 MB, 45.58% gc time)
Matriz de Confusão:

[2299 0 0
 0 3311 0
 0 0 4390]

Tamanho: 10000
Acertos: 10000
Erros: 0
Accuracy: 100.0%

Cluster 1

Tamanho: 2299
Accuracy: 100.0%
Precision: 100.0%
Recall: 100.0%
```

```
F-score: 1.0

Acerto positivo: 2299 (100.0%)
Acerto negativo: 7701 (100.0%)
Falso negativo: 0 (NaN%)
Falso positivo: 0 (NaN%)

Cluster 2

Tamanho: 3311
Accuracy: 100.0%
Precision: 100.0%
Recall: 100.0%
F-score: 1.0

Acerto positivo: 3311 (100.0%)
Acerto negativo: 6689 (100.0%)
Falso negativo: 0 (NaN%)
Falso positivo: 0 (NaN%)

Cluster 3

Tamanho: 4390
Accuracy: 100.0%
Precision: 100.0%
Recall: 100.0%
F-score: 1.0

Acerto positivo: 4390 (100.0%)
Acerto negativo: 5610 (100.0%)
Falso negativo: 0 (NaN%)
Falso positivo: 0 (NaN%)
```