P-Median 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 Programan Inteiro do P-Median (Facility Location Problem).

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 P-Median 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: 37
Number of Objects in 2: 41
Number of Objects in 3: 22
```

1.3.1 ULP - Problema de Localização sem Capacidade

Consiste em resolver o ULP determinar os objetos representates de cada grupo e classificar cada objeto como sendo do grupo com representante mais próximo

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

http://cseweb.ucsd.edu/~dasgupta/291-geom/kmedian.pdf

1.3.2 JuMP

http://www.juliaopt.org/

http://jump.readthedocs.org/en/stable/

Modeling language for Mathematical Programming (linear, mixed-integer, conic, nonlinear)

```
In [3]: if Pkg.installed("JuMP") === nothing
            println("Installing JuMP...")
            Pkg.add("JuMP")
            Pkg.add("Cbc")
        end
```

```
In [4]: using JuMP
```

```
In [5]: function dist(dataset)
            data = map(first, dataset.data)
            n = length(data)
            d = zeros(n, n)
            for i=1:n, j=i+1:n
                dist = norm(data[i] - data[j])
                d[i,j] = dist
                d[j,i] = dist
            end
            d
        end

        dist(dataset)
```

```
Out[5]: 100x100 Array{Float64,2}:
         0.0      2.82843  2.64575  2.82843  ...  1.73205  2.82843  3.16228  2.82843
         2.82843  0.0      2.64575  2.0         2.64575  2.82843  2.82843  2.82843
         2.64575  2.64575  0.0      1.73205     2.44949  2.64575  3.0      3.0
         2.82843  2.0      1.73205  0.0         2.64575  2.82843  2.82843  2.82843
         3.16228  2.44949  2.64575  2.44949     3.31662  2.82843  2.0      3.74166
         2.23607  2.23607  2.44949  2.23607  ...  2.44949  3.0      3.0      3.0
         2.23607  3.0      2.0      2.23607     2.0      2.64575  3.0      3.31662
         3.31662  2.64575  2.44949  2.23607     3.16228  3.0      3.0      2.64575
         2.23607  2.23607  2.44949  2.23607     2.44949  2.64575  3.31662  3.0
         3.0      3.31662  3.16228  3.60555     2.82843  3.0      2.23607  3.31662
         2.44949  2.44949  3.0      2.82843  ...  2.64575  3.4641   3.16228  3.16228
         3.0      2.64575  2.82843  3.0         3.16228  2.23607  3.31662  2.64575
         2.64575  2.23607  2.0      2.23607     2.44949  3.0      3.0      3.0
         ⋮                                   ⋱
         2.82843  2.44949  3.0      2.44949     2.64575  2.0      2.82843  2.0
         3.31662  3.31662  2.82843  3.0         3.16228  2.64575  2.64575  3.0
         2.64575  3.0      2.44949  3.0      ...  2.82843  3.0      3.31662  2.64575
         3.0      2.64575  3.16228  2.64575     2.82843  2.23607  2.64575  2.23607
         3.31662  3.31662  2.82843  2.64575     3.16228  3.31662  2.23607  2.64575
```

```
3.16228  2.44949  3.0      2.44949     3.0      2.44949  2.0      2.44949
2.44949  2.44949  3.0      2.44949     2.23607  2.44949  3.16228  2.0
2.44949  2.44949  3.0      3.16228  …  2.23607  3.16228  3.16228  2.44949
1.73205  2.64575  2.44949  2.64575     0.0      2.64575  3.0      2.64575
2.82843  2.82843  2.64575  2.82843     2.64575  0.0      2.82843  2.82843
3.16228  2.82843  3.0      2.82843     3.0      2.82843  0.0      3.16228
2.82843  2.82843  3.0      2.82843     2.64575  2.82843  3.16228  0.0
```

```
In [6]: let
            _dataset = Clustering.Dataset(size=10, groups=3, features=16, slot=3)
            n = _dataset.size
            k = _dataset.groups
            d = dist(_dataset)

            m = Model()

            @defVar(m, 0 <= x[1:n,1:n] <= 1)
            @defVar(m, y[1:n], Bin)

            # add the constraint that the amount that facility j can serve
            # customer x is at most 1 if facility j is opened, and 0 otherwise.
            for i=1:n, j=1:n
                @addConstraint(m, x[i,j] <= y[j])
            end

            # add the constraint that the amount that each customer must
            # be served
            for i=1:n
                @addConstraint(m, sum{x[i,j], j=1:n} == 1)
            end

            # add the constraint that at most 3 facilities can be opened.
            @addConstraint(m, sum{y[j], j=1:n} <= k)

            # add the objective.
            @setObjective(m, Min, sum{d[i,j] * x[i,j], i=1:n, j=1:n})

            status = solve(m)
            if status != :Optimal
                error("Wrong status (not optimal): $status")
            end

            println("Solver:\n\n", typeof(getInternalModel(m)), "\n")

            println("Objective value:\n\n", getObjectiveValue(m), "\n")

            centers = getValue(y)[:]
            println("Centros:\n\n", centers, "\n")

            clusters = getValue(x)[:,:]
            println("Clusters:\n\n", clusters, "\n")

            centersj = zeros(Int, k)
            assignments = zeros(Int, n)
```

```
            _k = 0
            for j=1:n
                centers[j] == 0.0 && continue
                _k += 1
                centersj[_k] = j
                for i=1:n
                    clusters[i,j] == 0.0 && continue
                    assignments[i] = _k
                end
            end

            println("Atribuição de Cluster:\n\n", assignments, "\n")

            dt = 0.0
            for (kj, j) in enumerate(centersj)
                for (i, ki) in enumerate(assignments)
                    kj != ki && continue
                    dt += d[i,j]
                end
            end
            println("Custo reconstruído (verificação):\n\n", dt, "\n")

            sleep(0.2)
        end
```

```
Solver:

Cbc.CbcMathProgSolverInterface.CbcMathProgModel

Objective value:

15.573949718466846

Centros:

[1.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0]

Clusters:

[1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0]

Atribuição de Cluster:

[1,3,2,1,1,2,1,2,1,3]
```

```
Custo reconstruído (verificação):

15.573949718466844
```

```
In [7]: "Algoritmo de clusterização P-Median (Programan Inteiro, Facility Location Problem)."
        function pmedian(dataset, k)
            n = dataset.size
            k = dataset.groups
            d = dist(dataset)

            m = Model()

            @defVar(m, 0 <= x[1:n,1:n] <= 1)
            @defVar(m, y[1:n], Bin)

            # add the constraint that the amount that facility j can serve
            # customer x is at most 1 if facility j is opened, and 0 otherwise.
            for i=1:n, j=1:n
                @addConstraint(m, x[i,j] <= y[j])
            end

            # add the constraint that the amount that each customer must
            # be served
            for i=1:n
                @addConstraint(m, sum{x[i,j], j=1:n} == 1)
            end

            # add the constraint that at most 3 facilities can be opened.
            @addConstraint(m, sum{y[j], j=1:n} <= k)

            # add the objective.
            @setObjective(m, Min, sum{d[i,j] * x[i,j], i=1:n, j=1:n})

            status = solve(m)
            if status != :Optimal
                error("Wrong status (not optimal): $status")
            end

            centers = getValue(y)[:]
            clusters = getValue(x)[:,:]

            assignments = zeros(Int, n)
            _k = 0
            for j=1:n
                centers[j] == 0.0 && continue
                _k += 1
                for i=1:n
                    clusters[i,j] == 0.0 && continue
                    assignments[i] = _k
                end
            end
            assignments
        end
```

```
pmedian(dataset, 3)
```

Out[7]:
```
100-element Array{Int64,1}:
 2
 3
 2
 2
 1
 3
 2
 3
 2
 1
 1
 3
 2
 ⋮
 3
 1
 1
 3
 1
 3
 3
 2
 2
 3
 1
 3
```

In [8]:
```
import Clustering.mapping

"Algoritmo de clusterização P-Median (Programan Inteiro, Facility Location Problem) \
aproximado para os grupos pré-definidos do dataset."
function pmedian_approx(dataset, k)
    assignments = pmedian(dataset, k)
    centermap = mapping(dataset, assignments, k)
    map(c -> centermap[c], assignments)
end

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

```
Matriz de Confusão:

[32 2 3
 1 37 3
 1 1 20]

Tamanho: 100
Acertos: 89
```

```
Erros: 11
Accuracy: 89.0%

Cluster 1

Tamanho: 37
Accuracy: 93.0%
Precision: 94.12%
Recall: 86.49%
F-score: 0.9

Acerto positivo: 32 (86.49%)
Acerto negativo: 61 (96.83%)
Falso negativo: 5 (45.45%)
Falso positivo: 2 (18.18%)

Cluster 2

Tamanho: 41
Accuracy: 93.0%
Precision: 92.5%
Recall: 90.24%
F-score: 0.91

Acerto positivo: 37 (90.24%)
Acerto negativo: 56 (94.92%)
Falso negativo: 4 (36.36%)
Falso positivo: 3 (27.27%)

Cluster 3

Tamanho: 22
Accuracy: 92.0%
Precision: 76.92%
Recall: 90.91%
F-score: 0.83

Acerto positivo: 20 (90.91%)
Acerto negativo: 72 (92.31%)
Falso negativo: 2 (18.18%)
Falso positivo: 6 (54.55%)
```

```
In [9]: # Timeout
        # Clustering.test_dataset("small", pmedian_approx)
        # sleep(0.2)
```

```
In [10]: # Timeout
         # Clustering.test_dataset("large", pmedian_approx)
         # sleep(0.2)
```