P-Median em Julia

February 14, 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
```

```
WARNING: redefining constant srcdir
WARNING: redefining constant default_datasetdir
```

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

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

```
Clusters: 3
Dimension (features): 16
Features per Cluster: 3
Probability of Activation: 0.8

Size: 100
Min Cluster size: 20
Max Cluster size: 40
Cluster 1 size: 35
Cluster 2 size: 27
Cluster 3 size: 38
```

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(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
    Symmetric(d)
end

dist(dataset.input.data)
```

Out[5]:
```
100x100 Symmetric{Float64,Array{Float64,2}}:
 0.0      2.64575  2.64575  2.64575  3.0      3.16228  2.82843  …  2.82843  2.82843  2.44949
 2.64575  0.0      2.44949  2.82843  2.44949  1.73205  1.0         3.31662  2.23607  2.23607  2
 2.64575  2.44949  0.0      2.0      2.44949  2.64575  2.64575     2.64575  3.0      3.31662  2
 2.64575  2.82843  2.0      0.0      2.44949  3.0      2.64575     2.23607  2.64575  3.0      3
 3.0      2.44949  2.44949  2.44949  0.0      2.23607  2.64575     3.0      2.23607  2.64575  2
 3.16228  1.73205  2.64575  3.0      2.23607  0.0      2.0      …  3.16228  1.41421  2.44949
 2.82843  1.0      2.64575  2.64575  2.64575  2.0      0.0         3.16228  2.44949  2.44949  2
 2.64575  2.82843  2.44949  2.82843  2.82843  2.64575  3.0         2.64575  3.0      3.0      3
 2.82843  3.31662  2.64575  2.64575  3.0      3.4641   3.16228     2.82843  3.4641   2.82843  2
 3.0      3.16228  3.16228  2.44949  2.44949  3.0      3.0         2.64575  2.64575  2.64575  2
 2.82843  3.0      2.64575  2.64575  2.64575  3.16228  2.82843  …  2.0      3.16228  2.82843
 3.16228  2.64575  2.23607  2.64575  2.64575  2.82843  2.44949     2.0      3.16228  3.16228  2
 3.4641   3.0      2.64575  2.64575  2.64575  2.82843  2.82843     2.44949  2.82843  2.82843  2
 3.0      3.16228  3.16228  3.16228  2.82843  3.0      3.31662     3.31662  3.0      2.64575  2
 3.31662  3.16228  3.16228  2.44949  2.82843  2.64575  3.0         2.64575  2.23607  3.0      3
 2.64575  3.16228  2.82843  3.16228  2.82843  3.0      3.0      …  3.0      3.31662  3.31662
 2.64575  2.0      2.82843  2.44949  2.0      2.23607  1.73205     3.0      2.23607  2.64575  3
 2.23607  2.0      2.82843  2.82843  2.44949  2.23607  2.23607     2.64575  2.23607  2.64575  3
 ⋮                                            ⋮                 ⋱                     ⋮
 2.82843  2.23607  2.23607  3.0      3.0      2.82843  2.44949     2.82843  3.16228  2.82843  2
```

```
3.16228  2.23607  3.31662  3.31662  3.0      2.44949  2.44949     3.16228  2.44949  2.44949  2
3.4641   3.0      2.23607  2.64575  2.23607  2.44949  2.82843  …  2.44949  2.82843  3.16228
2.64575  2.0      2.82843  2.82843  2.0      2.23607  2.23607     3.60555  2.23607  2.23607  2
3.4641   3.0      3.0      3.0      2.64575  3.16228  2.82843     3.16228  3.4641   3.16228  2
2.44949  2.23607  2.23607  2.23607  2.23607  2.0      2.44949     2.82843  2.0      2.82843  3
2.82843  2.23607  2.64575  3.0      2.23607  1.41421  2.44949     3.4641   2.0      2.44949  2
2.82843  2.64575  3.31662  3.31662  3.0      3.16228  2.82843  …  3.16228  3.16228  2.44949
2.44949  3.0      3.0      3.31662  3.31662  3.16228  3.16228     3.16228  3.16228  2.82843  2
3.0      3.4641   2.82843  2.82843  3.16228  3.0      3.31662     2.23607  3.0      3.31662  3
2.82843  3.31662  2.64575  2.23607  3.0      3.16228  3.16228     0.0      2.82843  3.16228  3
2.82843  2.23607  3.0      2.64575  2.23607  1.41421  2.44949     2.82843  0.0      2.0      2
2.44949  2.23607  3.31662  3.0      2.64575  2.44949  2.44949  …  3.16228  2.0      0.0
3.4641   2.23607  2.64575  3.0      2.23607  2.44949  2.44949     3.16228  2.82843  2.44949  0
3.4641   3.31662  2.64575  3.31662  3.0      2.82843  3.4641      2.82843  3.16228  3.16228  2
3.31662  3.16228  2.82843  3.16228  2.82843  3.0      3.0         3.0      3.31662  3.31662  2
2.64575  2.44949  2.44949  2.82843  2.0      2.64575  2.64575     2.64575  2.64575  2.64575  2
```

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

    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:

15.667148291503413

Centros:

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

Clusters:

[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 1.0 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 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 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 1.0
 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 1.0]

Atribuição de Cluster:

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

Custo reconstruído (verificação):

15.667148291503413
```

In [7]:
```
import Clustering: Input, Dataset

"Algoritmo de clusterização P-Median (Programan Inteiro, Facility Location Problem)."
function pmedian(input::Input, k::Int)
    n = input.size
    d = dist(input.data)

    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
```

```julia
            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::Dataset, k::Int) = pmedian(dataset.input, k)

    pmedian(dataset, 3)
```

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

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

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

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

```
1.173253 seconds (224.03 k allocations: 15.618 MB, 1.54% gc time)
Confusion Matrix:

[25 7 3
 2 22 3
 2 3 33]

Size: 100
Correct: 80
Mistakes: 20
Accuracy: 80.0%

Cluster 1

Size: 35
Accuracy: 86.0%
Precision: 86.21%
Recall: 71.43%
F-score: 0.78

True Positive: 25 (71.43%)
True Negative: 61 (93.85%)
False Negative: 10 (50.0%)
False Positive: 4 (20.0%)

Cluster 2

Size: 27
Accuracy: 85.0%
Precision: 68.75%
Recall: 81.48%
F-score: 0.75

True Positive: 22 (81.48%)
True Negative: 63 (86.3%)
False Negative: 5 (25.0%)
False Positive: 10 (50.0%)

Cluster 3
```

```
Size: 38
Accuracy: 89.0%
Precision: 84.62%
Recall: 86.84%
F-score: 0.86

True Positive: 33 (86.84%)
True Negative: 56 (90.32%)
False Negative: 5 (25.0%)
False Positive: 6 (30.0%)
```

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

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