P-Median em Julia

February 11, 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
        dataset = Clustering.load_dataset("small")
        Clustering.summary(dataset)
        sleep(0.2)
```

```
Number of Groups: 3
Number of Features: 200
Number of Features (group): 40
Probability of Activation: 0.8
Number of Objects (total): 100
Number of Objects per Group (min): 7
Number of Objects per Group (max): 66
Number of Objects in 1: 39
Number of Objects in 2: 17
Number of Objects in 3: 45
```

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

```
In [3]: using JuMP
```

```
In [4]: 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[4]: 101x101 Array{Float64,2}:
          0.0       8.60233   8.7178   10.1489   ...   9.0       8.42615  11.1355
          8.60233   0.0       9.27362  10.8167       9.21954   8.77496  10.0995
          8.7178    9.27362   0.0      10.4403       9.94987   8.77496  10.4881
         10.1489   10.8167   10.4403   0.0          10.198    10.0      10.0499
         10.583    10.3923   10.0995   9.21954      10.4403   10.5357   10.3923
         10.4403   10.8167   10.5357   9.59166  ...  10.0995   10.198     8.30662
         10.3441   10.3441   10.7238   9.89949      10.2956   10.0995    8.544
         10.3923   10.7703   10.4881   8.18535      10.0499   10.6301   10.3923
         11.0454   10.0995   10.198   10.3441       10.4403   10.4403    8.24621
          7.93725   8.88819   9.11043  10.4881       8.7178    8.12404  10.6301
          7.93725   9.32738   8.66025  10.3923   ...   9.16515   9.38083  10.8167
          9.21954   8.88819   8.77496  10.583        9.05539   9.27362  10.5357
         10.2956   10.3923   10.7703   9.32738      10.0499   10.247    10.3923
          ⋮                                      ⋱                      ⋮
         10.8628   10.0995   10.6771   10.247       10.5357   10.6301    8.48528
         10.3441   10.4403   10.3441   8.83176  ...  10.0      10.198    10.4403
          8.60233   9.16515   8.0      10.7238       8.88819   8.66025  10.7703
          8.544     8.88819   8.66025  10.4881       9.48683   8.7178   10.7238
         11.0454   10.4881   10.0995   8.77496      10.6301   10.8167   10.0995
          8.06226   8.30662   9.0      10.9545       9.48683   8.7178   10.9087
          8.77496   9.0       8.42615  10.6771   ...   9.48683   8.60233  10.5357
          8.48528   9.27362   8.83176  10.1489       8.544     8.77496  10.0
         10.247     9.53939  10.0499   10.0          10.8628    9.89949   8.88819
          9.0       9.21954   9.94987  10.198        0.0       9.05539  10.4403
          8.42615   8.77496   8.77496  10.0          9.05539   0.0      10.3441
         11.1355   10.0995   10.4881   10.0499   ...  10.4403   10.3441    0.0
```

```
In [5]: 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:

16.056673160848906

Centros:

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

Clusters:

[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 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 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 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 1.0 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 0.0 0.0 0.0 1.0 0.0 0.0 0.0]

Atribuição de Cluster:

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

Custo reconstruído (verificação):

16.056673160848906
```

```
In [6]: 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[6]:
```
100-element Array{Int64,1}:
 1
 1
 1
 2
 2
 3
 3
```

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

```
In [7]: function pmedian_approx(dataset, k)
            assignments = pmedian(dataset, k)
            centermap = Clustering.mapping(dataset, assignments, k)
            map(c -> centermap[c], assignments)
        end

        let
            n = 100
            k = 3
            c = 16
            c_y = 3

            tiny = Clustering.Dataset(size=n, groups=k, features=c, slot=c_y)

            prediction = pmedian_approx(tiny, k)
            Clustering.evaluation_summary(tiny, prediction)
        end
```

```
Precision: 87.21%
Recall: 84.27%
F-score: 0.86

Número de predições: 100
Acertos: 75 (75.0%)
Falso negativo: 14 (14.0%)
Falso positivo: 11 (11.0%)

Cluster 1

Objetos: 41
Accuracy: 84.0%
Precision: 87.88%
Recall: 70.73%
F-score: 0.78
```

```
Acerto positivo: 29 (70.73%)
Acerto negativo: 55 (93.22%)
Falso negativo: 12 (85.71%)
Falso positivo: 4 (36.36%)

Cluster 2

Objetos: 17
Accuracy: 78.0%
Precision: 41.38%
Recall: 70.59%
F-score: 0.52

Acerto positivo: 12 (70.59%)
Acerto negativo: 66 (79.52%)
Falso negativo: 5 (35.71%)
Falso positivo: 17 (154.55%)

Cluster 3

Objetos: 42
Accuracy: 88.0%
Precision: 89.47%
Recall: 80.95%
F-score: 0.85

Acerto positivo: 34 (80.95%)
Acerto negativo: 54 (93.1%)
Falso negativo: 8 (57.14%)
Falso positivo: 4 (36.36%)
```