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

```
WARNING: type Dataset not present in workspace; reconstructing


        LoadError: MethodError: ‘summary‘ has no method matching summary(::JLD.##Dataset#8091)
    you may have intended to import Base.summary
    while loading In[1], in expression starting on line 5
```

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]: 100x100 Array{Float64,2}:
          0.0       9.38083  10.4403   10.198    ...  10.7238   10.0      10.5357
          9.38083   0.0      10.1489   10.4881       9.94987  10.2956   10.3441
         10.4403   10.1489    0.0       9.11043      9.89949   9.21954   9.38083
         10.198    10.4881    9.11043   0.0          9.94987   9.48683   9.32738
         10.198     9.89949  10.0499   10.2956       8.544     9.48683  10.5357
         10.9087   10.0499    8.3666    9.11043  ...  10.0995    9.11043   9.38083
          9.94987  10.1489    8.83176  10.1489      10.198     8.88819   9.38083
          9.0       9.11043  10.6771   10.4403      10.0995   10.4403   10.6771
         10.247    10.4403    8.94427   9.0         10.4881    9.53939   9.16515
         10.3441   10.3441    8.24621   8.77496     10.2956    9.0       9.27362
         10.3441   10.0499    8.7178    8.88819  ...  10.7703    8.88819   9.05539
         10.3441   10.247     8.83176   9.43398     10.583     8.66025   9.27362
         10.1489   10.3441    8.83176   9.0          9.89949   9.0       8.94427
          ⋮                                      ⋱
         10.5357   10.247    10.3923    9.84886      9.16515  10.3441   10.198
         10.7238   10.5357    8.60233   9.11043      9.79796   9.32738   9.38083
         10.198     9.79796   9.11043   8.3666   ...  10.3441    9.59166   8.544
         10.3923   10.8628    8.88819   9.38083     10.5357    8.83176   9.21954
         10.583    10.4881    8.77496   9.05539     10.7238    8.7178    9.32738
         10.3441   10.4403    8.94427   9.84886     10.6771    9.43398   9.27362
         10.4403   10.4403    8.60233   8.66025     10.2956    9.21954   8.7178
         10.198    10.583     9.11043   8.12404  ...  10.4403    9.05539   8.42615
         10.4881   10.0       9.94987   9.89949      8.66025  10.4881   10.6301
         10.7238    9.94987   9.89949   9.94987      0.0      10.5357   10.2956
         10.0      10.2956    9.21954   9.48683     10.5357    0.0       9.0
         10.5357   10.3441    9.38083   9.32738     10.2956    9.0       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:

14.350451132510427

Centros:

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

Clusters:

[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 0.0 1.0
 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 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 0.0 1.0]

Atribuição de Cluster:

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

Custo reconstruído (verificação):

14.350451132510429
```

```
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}:
 2
 2
 1
 1
 3
 1
 1
```

```
 2
 1
 1
 1
 1
 1
 ⋮
 3
 1
 1
 1
 1
 1
 1
 1
 3
 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; verbose=true)
        end
```

```
Matriz de Confusão:

[15 0 2
 5 30 5
 3 6 34]

Tamanho: 100
Acertos: 79
Erros: 21
Accuracy: 79.0%

Cluster 1

Tamanho: 17
Accuracy: 90.0%
Precision: 65.22%
```

```
Recall: 88.24%
F-score: 0.75

Acerto positivo: 15 (88.24%)
Acerto negativo: 75 (90.36%)
Falso negativo: 2 (9.52%)
Falso positivo: 8 (38.1%)

Cluster 2

Tamanho: 40
Accuracy: 84.0%
Precision: 83.33%
Recall: 75.0%
F-score: 0.79

Acerto positivo: 30 (75.0%)
Acerto negativo: 54 (90.0%)
Falso negativo: 10 (47.62%)
Falso positivo: 6 (28.57%)

Cluster 3

Tamanho: 43
Accuracy: 84.0%
Precision: 82.93%
Recall: 79.07%
F-score: 0.81

Acerto positivo: 34 (79.07%)
Acerto negativo: 50 (87.72%)
Falso negativo: 9 (42.86%)
Falso positivo: 7 (33.33%)
```