DMKM 2013 - LYON

Optimization lab #3

The cluster-median problem

Yue He – Marcello Benedetti

03/12/2013

1. k-medians clustering

In data mining we often want to partition objects without knowing "a priori" the label (or the color) that should be applied to each element. Such unsupervised learning can be achieved using *k-median*s which is one of the methods of cluster analysis.

Let us better define the problem. Given a matrix A of *m* points and *n* variables, we want to find *k* clusters such that the overall distance of all the points to the median of the clusters that they belong to is minimized. The median for a subset of points $I \subseteq \{1, \ldots, m\}$ is defined as the nearest point to all the points of *I*:

$$r \text{ is the median of } \mathcal{I} \text{ if } \sum_{i \in \mathcal{I}} d_{ir} = \min_{j \in \mathcal{I}} \sum_{i \in \mathcal{I}} d_{ij},$$

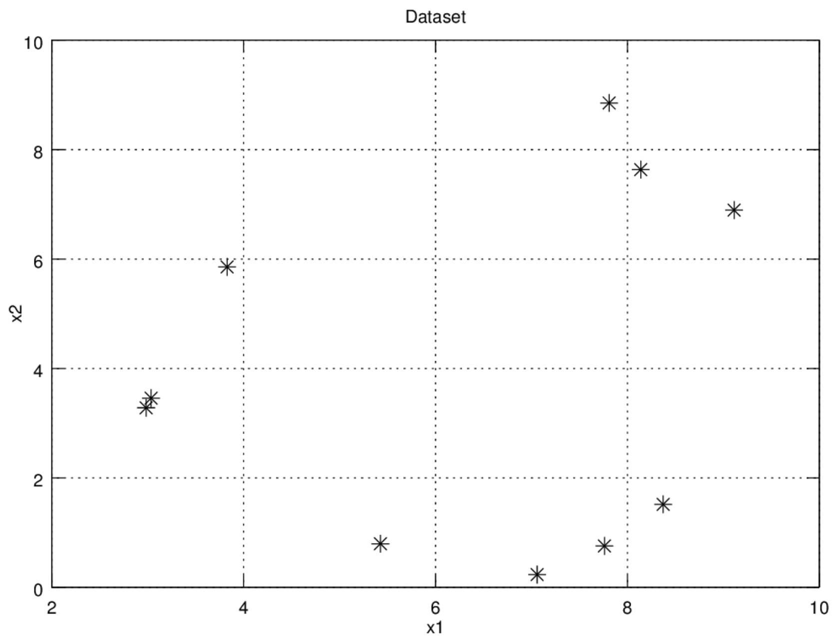
where $D = (d_{ij}),\ i = 1, \ldots, m,\ \ j = 1, \ldots, m$ is the matrix of the distances for each pair of points. *D* is computed from matrix *A*, using any available distance: Euclidean distance, Mahalanobis distance, etc.

In this report we will solve the k-medians problem with both an integer optimization approach and a heuristic approach. We will then provide a comparison and plot the results for better visualization of the 2 techniques.

2. The dataset

Firstly, we notice that "*k*" stands for the number of clusters we want to find among the data and it is up to the analyst to define the proper value. For the sake of simplicity, our dataset will be defined as a 10x2 matrix of random 2-dimentional points along with a 10x10 matrix of Euclidean distances. The aim is to partition the points in $k = 3$ clusters.
Here are the matrix *A* and the plot of generated points with coordinates *x1* and *x2*:

$$A = \begin{bmatrix} 2.9842 & 3.2808 \\ 3.0351 & 3.4567 \\ 3.8284 & 5.8555 \\ 7.8097 & 8.8520 \\ 8.1392 & 7.6337 \\ 8.3731 & 1.5165 \\ 5.4262 & 0.7961 \\ 7.0594 & 0.2333 \\ 7.7610 & 0.7567 \\ 9.1127 & 6.8939 \end{bmatrix}$$



The related Euclidean distances matrix is:

$$D = \begin{bmatrix} 0 & 0.1831 & 2{,}7096 & 7{,}3704 & 6{,}7469 & 5{,}6704 & 3{,}4838 & 5{,}0887 & 5{,}4026 & 7{,}1142 \\ 0.1831 & 0 & 2{,}5266 & 7{,}2046 & 6{,}5954 & 5{,}6797 & 3{,}5772 & 5{,}1561 & 5{,}4428 & 6{,}9822 \\ 2{,}7096 & 2{,}5266 & 0 & 4{,}9829 & 4{,}6631 & 6{,}2834 & 5{,}3057 & 6{,}4845 & 6{,}4391 & 5{,}3853 \\ 7{,}3704 & 7{,}2046 & 4{,}9829 & 0 & 1{,}2621 & 7{,}3571 & 8{,}4011 & 8{,}6513 & 8{,}0954 & 2{,}3520 \\ 6{,}7469 & 6{,}5954 & 4{,}6631 & 1{,}2621 & 0 & 6{,}1216 & 7{,}3561 & 7{,}4787 & 6{,}8873 & 1{,}2227 \\ 5{,}6704 & 5{,}6797 & 6{,}2834 & 7{,}3571 & 6{,}1216 & 0 & 3{,}0337 & 1{,}8364 & 0{,}9757 & 5{,}4280 \\ 3{,}4838 & 3{,}5772 & 5{,}3057 & 8{,}4011 & 7{,}3561 & 3{,}0337 & 0 & 1{,}7275 & 2{,}3351 & 7{,}1255 \\ 5{,}0887 & 5{,}1561 & 6{,}4845 & 8{,}6513 & 7{,}4787 & 1{,}8364 & 1{,}7275 & 0 & 0{,}8753 & 6{,}9699 \\ 5{,}4026 & 5{,}4428 & 6{,}4391 & 8{,}0954 & 6{,}8873 & 0{,}9757 & 2{,}3351 & 0{,}8753 & 0 & 6{,}2843 \\ 7{,}1142 & 6{,}9822 & 5{,}3853 & 2{,}3520 & 1{,}2227 & 5{,}4280 & 7{,}1255 & 6{,}9699 & 6{,}2843 & 0 \end{bmatrix}$$

3. The integer programming problem

Even though only *k* clusters are needed, *m* clusters will be considered in the formulation, such that *m-k* of them will be empty. The cluster whose median is the element *j* will be denoted as *cluster-j*. The variables of the formulation are:

- $x_{ij} = 1$ $i=1,\dots,m$ $j = 1,\dots,m$ if element *i* belongs to cluster-j;
- $x_{ij} = 0$ otherwise.

The goal is to make *k* and only *k* "correct" clusters (i.e. every element belongs to one and only one cluster-j), minimizing the overall distance of points to their cluster medians. Note that if cluster-j exists, then element *j* will be its median. In other words, if cluster-j exists then $x_{jj} = 1$. Otherwise, the total distance would not be minimized. This particular result is used also to force the partition in exactly *k* clusters. To approach the k-medians as an optimization problem we define the following cost function and constraints:

$$\begin{array}{lll} \min & \displaystyle\sum_{i=1}^{m}\sum_{j=1}^{m} d_{ij}x_{ij} & \text{[Distance of all points to their cluster medians]} \\ \text{subject to} & \displaystyle\sum_{j=1}^{m} x_{ij} = 1 \quad i = 1,\dots,m & \text{[Every point belongs to one cluster]} \\ & \displaystyle\sum_{j=1}^{m} x_{jj} = k & \text{[Exactly } k \text{ clusters]} \\ & x_{jj} \geq x_{ij} \quad i,j = 1,\dots,m & \text{[A point may belong to a cluster only if the cluster exists]} \\ & x_{ij} \in \{0,1\} & \end{array}$$

The problem translates in the following AMPL model (file *cluster_median.mod*).

```
# Number of points
param m;
set M:={1..m};

# Euclidean distances matrix
param D{M,M};

# Number of clusters
param k;
```

```
# Binary matrix of the results
var x{M,M} binary;

# Cost function
minimize obj: sum{i in M, j in M} D[i,j] * x[i,j];

# Constraint 1: every point belongs to one and only one cluster
subject to s1 {i in M}: sum{j in M} x[i,j]=1;

# Constraint 2: exactly k clusters exist
subject to s2 :sum{j in M} x[j,j]=k;

# Constraint 3: a point may belong to a cluster only if the cluster exists
subject to s3 {i in M, j in M}: x[j,j]>=x[i,j];
```

The Euclidean distances among the points have been stored in a proper format for the AMPL software (file *cluster_median.dat*). Then we choose to solve the problem with both MINOS and CPLEX solvers. We expect CPLEX to perform better because it is specialized for integer programming problems.

```
ampl: model cluster_median.mod;
ampl: data cluster_median.dat;
ampl: option solver minos;
ampl: solve;
MINOS 5.5: ignoring integrality of 100 variables
MINOS 5.5: optimal solution found.
94 iterations, objective 9.3806
ampl: display x;
x [*,*]
:    1   2   3   4   5   6   7   8   9  10    :=
1    0   1   0   0   0   0   0   0   0   0
2    0   1   0   0   0   0   0   0   0   0
3    0   1   0   0   0   0   0   0   0   0
4    0   0   0   0   1   0   0   0   0   0
5    0   0   0   0   1   0   0   0   0   0
6    0   0   0   0   0   0   0   0   1   0
7    0   0   0   0   0   0   0   0   1   0
8    0   0   0   0   0   0   0   0   1   0
9    0   0   0   0   0   0   0   0   1   0
10   0   0   0   0   1   0   0   0   0   0
;
ampl: option solver cplex;
ampl: solve;
CPLEX 12.5.1.0: optimal integer solution; objective 9.3806
12 MIP simplex iterations
0 branch-and-bound nodes
ampl: display x;
x [*,*]
:    1   2   3   4   5   6   7   8   9  10    :=
1    0   1   0   0   0   0   0   0   0   0
2    0   1   0   0   0   0   0   0   0   0
3    0   1   0   0   0   0   0   0   0   0
4    0   0   0   0   1   0   0   0   0   0
5    0   0   0   0   1   0   0   0   0   0
6    0   0   0   0   0   0   0   0   1   0
7    0   0   0   0   0   0   0   0   1   0
8    0   0   0   0   0   0   0   0   1   0
9    0   0   0   0   0   0   0   0   1   0
10   0   0   0   0   1   0   0   0   0   0
;
```

As expected the solvers created the same clustering. In particular, we can see how points 2, 5 and 9 have been selected as medians for each cluster. Also we notice how CPLEX found a solution after 12 iterations while MINOS needed 94 iterations.

To better understand the results we will visualize the clusters in a later chapter after solving the problem with a heuristic approach.

4. The minimum spanning tree problem

In this chapter we want to find a solution using a greedy algorithm. Indeed, the clustering problem can also be formulated as a *minimum spanning tree* problem. After finding the optimal path that connects all the points we will remove the *k-1* most costly edges. This will guarantee a local optimal solution to the problem of partitioning in *k* clusters.

We choose to use Kruskal's algorithm in AMPL (file *kruskal.mod*). This implementation is freely available at http://www.math.bme.hu/~bog/OKPR/kruskal.ampl.

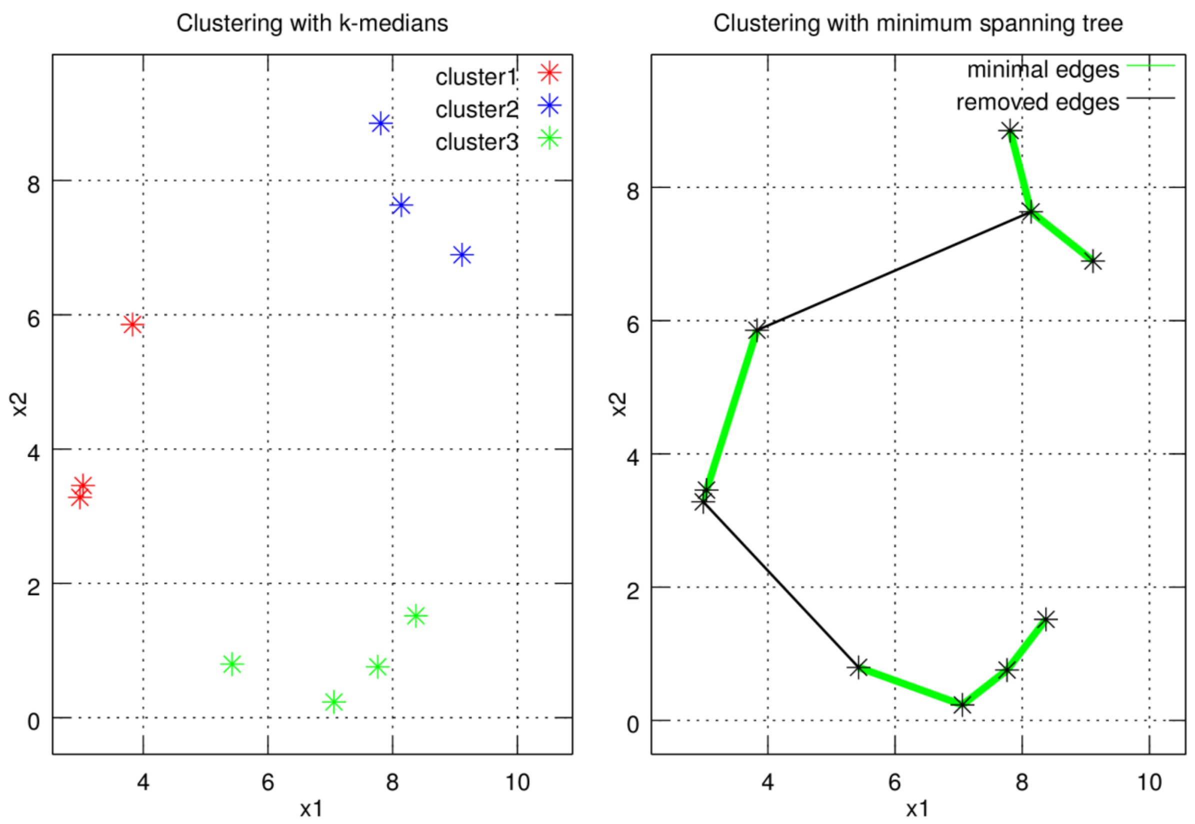
Before solving we format the dataset to be compatible with this implementation (file *kruskal.dat*). Finally we can proceed solving with CPLEX.

```
ampl: option solver cplex;
ampl: model kruskal.mod;

...
...  #15 iterations here
...

A MST for this graph has cost 16.
set T := (1,2) (8,9) (6,9) (5,10) (4,5) (7,8) (2,3) (1,7) (3,5);
ampl: printf "edge      cost\n"; for {(i,j) in T }{ printf "(%i %2i)    %f \n",
i, j, c[i,j]};
edge      cost
(1  2)    0.183100
(8  9)    0.875300
(6  9)    0.975700
(5 10)    1.222700
(4  5)    1.262100
(7  8)    1.727500
(2  3)    2.526600
(1  7)    3.483800
(3  5)    4.663100
```

After finding the optimal path we printed out the costs for each edge. In the next chapter we will remove the 2 most costly edges and plot the results to graphically see the clusters.

5. Results comparison

Finally we plotted the results in Octave and compared the two approaches. In the first plot the points have been colored according to the results given in chapter 3. That is to say for each column of the *X* matrix we gave a different label. In the second plot we can see how Kruskal's algorithm produced the same partitioning (represented with green lines).



6. Real world application

As extra activity we used the k-medians model to cluster real data. Our purpose is not to make conclusions about the original data, but rather to test the performance of model. We chose a dataset of geometrical properties about 3 kinds of wheat seeds: Kama, Rosa and Canadian. The dataset is made of 210 instances and 7 attributes. Database and description are available at http://archive.ics.uci.edu/ml/datasets/seeds.

According to the database, points should be clustered as following:

- Cluster 1: points from 1 to 70;
- Cluster 2: points from 71 to 140;
- Cluster 3: points from 141 to 210.

We calculated the Euclidean distance matrix and stored them in *seeds_distances.dat.* Given the big amount of variables we solved the problem via NEOS website using the MINTO solver since CPLEX was not available.

```
Executing /opt/neos/Drivers/minto-ampl/minto-ampl-driver.py at time: 2013-12-03
20:13:36.337002
File exists
You are using the solver mintoamp.
Executing AMPL.
processing data.
processing commands.

Presolve eliminates 210 constraints.
Adjusted problem:
44100 variables, all binary
44101 constraints, all linear; 132090 nonzeros
        211 equality constraints
        43890 inequality constraints
1 linear objective; 43890 nonzeros.

MINTO (AMPL) v3.1: MINTO arg string (from AMPL options): ''

MINTO, a Mixed integer Optimizer
Copyright (C) 1992-2007  --  M.W.P. Savelsbergh, J.T. Linderoth

MINTO: Solving problem amplprob
MINTO: Problem statistics:

    Number of constraints: 44101
    Number of variables:   44100 (0)
    Number of nonzero's:   132090

    Number of continuous variables: 0
    Number of binary variables:     44100
    Number of integer variables:    0

MINTO: Row structure analysis (after preprocessing):
    Number of constraints of type ALLBINEQ:     1
    Number of constraints of type BINSUM1UB:    43890
    Number of constraints of type BINSUM1EQ:    210

MINTO control parameters:
    Objective sense     : minimization
    Output level        : 1
    Maximum cpu time    : 100000000
    Maximum #nodes      : 100000000

MINTO system function activity levels:
    Bound improvement   : active
    Branching type      : 3
    Node selection type : 5
    Preprocessing level : 1
    Primal heuristic    : active
    Clique cuts         : active
    Implication cuts    : active
    Knapsack covers     : active
    GUB covers          : active
    Flow covers         : active
    Gomory cuts         : active
    Row management      : active
    Restarts            : active
```

```
    Force branching      : 1
    Advanced basis       : active
    Names mode level     : 0

MINTO: Updating primal (Integral solution)
       Value:      -314.25  Elapsed time:     1.77  Node:       1
MINTO: Set LP Cutoff Value to: -314.253272

MINTO: Value of solution: 314.253272

...
...     #Several statistics here
...

Clusters:
{1      2      3      4      5      6      7      8      9      10     11     12
13      14     15     16     18     19     21     22     23     25     26     29
30      31     32     33     34     35     36     37     39     41     42     43
44      45     46     47     48     49     50     51     52     53     54     55
56      57     58     59     65     66     67     68     69     101    123    125
133     134    135    136    138    139    140}

{38     71     72     73     74     75     76     77     78     79     80     81
82      83     84     85     86     87     88     89     90     91     92     93
94      95     96     97     98     99     100    102    103    104    105    106
107     108    109    110    111    112    113    114    115    116    117    118
119     120    121    122    124    126    127    128    129    130    131    132
137}

{17     20     24     27     28     40     60     61     62     63     64     70
141     142    143    144    145    146    147    148    149    150    151    152
153     154    155    156    157    158    159    160    161    162    163    164
165     166    167    168    169    170    171    172    173    174    175    176
177     178    179    180    181    182    183    184    185    186    187    188
189     190    191    192    193    194    195    196    197    198    199    200
201     202    203    204    205    206    207    208    209    210}
```

The outputs are remarkable:

- 183 seeds were correctly clustered (**89%**)
- highlighted in red, 23 seeds were wrongly clustered (**11%**).

However, we could probably improve the results using a different calculation of distances like Mahalanobis or using K-mean instead of K-median.