

## Help on Getting Started with Your Project

The different CD (community detection) algorithms have been implemented in R and this quick tutorial is meant to just get you started. You will need to explore more in your project. First, we cover the different representation of graphs that are needed by the different CD algorithms.

## Graphs, Adjacency Lists, and Distance matrices

There are several ways to create a distance matrix

1. Compute from table of features
  - A table of features is a data frame like,

	row.names	Murder	Assault	UrbanPop	Rape
1	Alabama	13.2	236	58	21.2
2	Alaska	10.0	263	48	44.5
3	Arizona	8.1	294	80	31.0
4	Arkansas	8.8	190	50	19.5
5	California	9.0	276	91	40.6
6	Colorado	7.9	204	78	38.7
7	Connecticut	3.3	110	77	11.1
8	Delaware	5.9	238	72	15.8
9	Florida	15.4	235	80	31.9

(You can load the above by entering `data(USArrests)`)

- Suppose your features are in a data frame called “feat”
 

Note: you can easily plot out pairwise features for inspection. Install the packages “ggplot2” and “calibrate” by `install.packages(c('ggplot2', 'calibrate'))`, `library('ggplot2')`, `library('calibrate')` then enter, e.g. `plot(USArrests$Murder, USArrests$Assault)`, followed by `textxy(USArrests$Murder, USArrests$Assault, row.names(USArrests))` enter `“dist(feat, method)”`, where method can be: “euclidean”, “maximum”, “manhattan”, “canberra”, “binary” or “minkowski”

2. Convert from a (symmetric) matrix which contains the pairwise distances

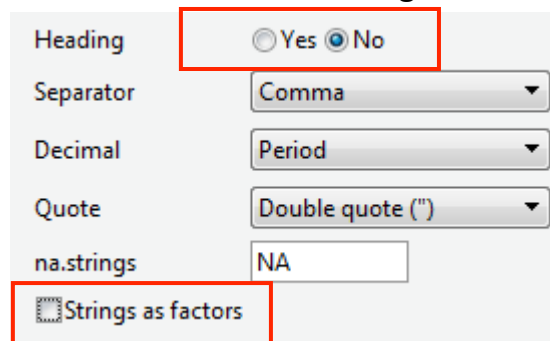
- Suppose the matrix is in a data frame called “mdat”
- `as.dist(mdat)`
- Example:
  - First, create a matrix
 

```
mdat <- matrix(c(1, 2, 1, 1, 7, 7, 2, 1, 7, 7,
1, 1, 1, 7, 1, 1, 7, 7, 1, 7, 1, 1, 7, 7, 1, 1), nrow = 6, ncol = 6)
```
  - Then, convert into distance matrix
 

```
as.dist(mdat)
```

## 3. From edge list

- Install `igraph`, `netmeta` by `install.packages(c('igraph', 'netmeta'))`, and `library('igraph')`, `library('netmeta')`.
- Load edges into a data frame called “edges”. Remember to uncheck “Strings as factors” to preserve the vertex labels. Also, remember to set “Heading” to “Yes” or “No” accordingly.



Heading ☐ Yes ☒ No

Separator Comma

Decimal Period

Quote Double quote (")

na.strings NA

☒ Strings as factors

- Then, you can generate a distance matrix with `as.dist(netdistance(get.adjacency(graph.edgelist(as.matrix(edges), directed=FALSE))))`
- Try with the included file “edges.csv”.

## Single Linkage

To perform single linkage, you can use the `hclust` function. The function accepts a distance matrix as input. So, you will have to prepare your data as a distance matrix.

## Clustering with hclust

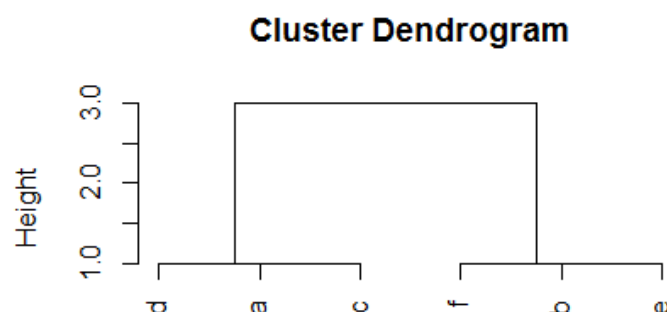
Assume that you already have a distance matrix called “`dist`”. To perform single linkage clustering with `hclust`, simply enter “`hclust(dist, "single")`”.

Note: you can also try other clustering methods beside “`single`”. Other methods are “`ward.D`”, “`ward.D2`”, “`single`”, “`complete`”, “`average`” (= UPGMA), “`mcquitty`” (= WPGMA), “`median`” (= WPGMC) or “`centroid`” (= UPGMC).

## Displaying hclust result

If you ran `hclust` with “`C <- hclust(dist)`” (the default method is “`complete`”) and hence stored the result in `C`, you can view it with “`plot(C)`”.

This will give you the clustering like below for the graph in “`edges.csv`”:



(See details in <http://www.inside-r.org/r-doc/stats/hclust> )

## Girvan-Newman (GN)

GN clustering is similarly performed.

The function call to perform GN is called “`edge.betweenness.community`”, and is included in the “igraph” package. If you have not installed igraph earlier, install it with `packages.install('igraph')` and `library('igraph')`.

The function accepts a graph as input.

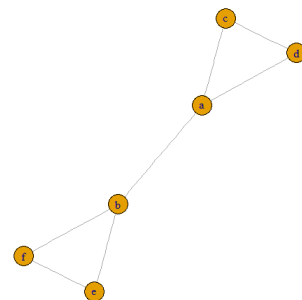
## Creating a graph with hclust

Graphs can be created from edge lists. Assume that your edge list is available as a data frame (e.g. loaded from “edges.csv”).

You generate the graph by

```
G <- graph_from_data_frame(d=edges, vertices=
unique(c(edges$V1, edges$V2)), directed=F)
```

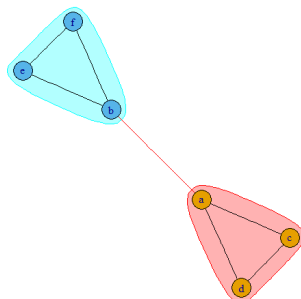
You can examine G by plotting it with “`plot(G)`”.



## Performing the GN clustering

Finally, you can call enter “`C <- edge.betweenness.community(G)`” to perform the clustering.

To display the community, enter “`plot(C, G)`”.



(See <http://www.inside-r.org/packages/cran/igraph/docs/edge.betweenness.community> )

## Markov Cluster Algorithm (MCL)

The function call to perform MCL is called `mcl`, and is included in the “MCL” package. Install it with `packages::install('MCL')` and `library('MCL')`

`mcl` accepts an adjacency matrix as input.

### Creating an adjacency matrix

Adjacency matrices can be created from edge lists. Assume that your edge list is available as a data frame (e.g. loaded from “edges.csv”).

You can obtain an adjacency matrix by

```
adj <- get.adjacency(graph.edgelist(as.matrix(edges),
directed=FALSE))
```

And you can convert the adjacency matrix into a graph by

```
G <- graph.adjacency(adj, mode="undirected")
```

Which will allow you to examine it via “`plot(G)`”.

### Performing the MCL clustering

Finally, you can call enter “`C <- mcl(x=adj, addLoops=T, ESM=T)`” to perform the clustering. The result is given in the following parameters:

K	the number of clusters
n.iterations	the number of iterations
Cluster	a vector of integers indicating the cluster to which each vertex is allocated
Equilibrium.state.matrix	a matrix; rows contain clusters

For “edges.csv”, you will find {a, d, c} in one cluster, and {b, e, f} in another.

(See details in <https://cran.r-project.org/web/packages/MCL/MCL.pdf>)

## Guide on getting clustering results into pals

Your clustering result may be stored in different data types, and you may need to convert it to make it suitable for the pals system. This guide tells you how.

### Girvan-Newman (GN)

Steps to convert a Girvan-Newman clustering result into pals format:

Assume that the command executed is

```
C <- edge.betweenness.community(G)
```

That is, C now stores the clustering result of graph G. Now, to convert the graph G and clusters C into pals input,

```
edgs <- get.edgelist(G)
clus <- data.frame(1:length(V(G)), C$membership)
clus <- clus[unique(c(edgs[,1], edgs[,2])),]
write.table(edgs, "G.dat", sep="\t", row.names=F, col.names=F)
write.table(clus, "C.dat", sep="\t", row.names=F, col.names=F)
```

The graph is then stored in G.dat while the clusters are stored in C.dat.

### Markov Cluster Algorithm (MCL)

To convert an MCL clustering result into pals format, use the following.

Assume that the command executed is

```
C <- mcl(x=adj, addLoops=T, ESM=T)
```

That is, C now stores the clustering result of graph adjacency matrix adj. Now, to convert adj and clusters C into pals input,

```
G <- graph.adjacency(adj, mode="undirected")
edgs <- get.edgelist(G)
clus <- data.frame(1:length(V(G)), C$cluster)
clus <- clus[unique(c(edgs[,1], edgs[,2])),]
write.table(edgs, "G.dat", sep="\t", row.names=F, col.names=F)
write.table(clus, "C.dat", sep="\t", row.names=F, col.names=F)
```

The graph is then stored in G.dat while the clusters are stored in C.dat.

## Single Linkage

Without loss of generality assume that your original data is in the form of a graph  $G$  (otherwise how are you going to input it into pals), which you converted into a dist object  $D$  through the function call `netdistance` as follows.

```
D <- as.dist(netdistance(get.adjacency(G)))
```

(You may have needed to perform `D[!is.finite(D)] <- length(V(G))` in order to get rid of the “Inf” entries in  $D$ , which signify the distance between unreachable vertices.)

To convert a single linkage clustering result into pals format, use the following.

Assume that the command executed is

```
hc <- hclust(D, method="single")
```

That is,  $hc$  now stores the hierarchical clustering result of graph  $G$ . Now, to convert the graph  $G$  and clusters  $C$  into pals input,

```
edgs <- get.edgelist(G)
clus <- data.frame(1:length(V(G)), cutree(hc, k = 15))
clus <- clus[unique(c(edgs[,1], edgs[,2])),]
write.table(edgs, "G.dat", sep="\t", row.names=F, col.names=F)
write.table(clus, "C.dat", sep="\t", row.names=F, col.names=F)
```

You will need to choose a suitable value of  $k$  for `cutree` – in the codes above, 15 is specified, but that may not be suitable for every case.