Test Weighting Matrix

Paul Harmon

12/15/2020

Introduction

We can either modify HierarchicalSparseCluster() or MonoClust() in order to induce sparse monothetic clustering. Ideally, if we can use the weight vector w to re-weight the original data, we can pass this reweighted version directly through to MonoClust.

Alternatively, we would need to allow monoclust to pass a dissimilarity matrix as input.

Example

Using a 15x3 NBA dataset.

```
nba <- read.csv("C:/Users/paulh/OneDrive/Documents/Utah Jazz Simulations/UtahJazzPredictions/Player_Sta
#scales the dataset
## Consider a dataframe
nba1 <- select(nba, FG, ORB, PTS) %>% apply(2,scale)
```

Applies Sparse Hierarchical Clustering.

```
hc1 <- HierarchicalSparseCluster(as.matrix(nba1), wbound = 1.01, dissimilarity = "squared.distance")
```

```
## 12
```

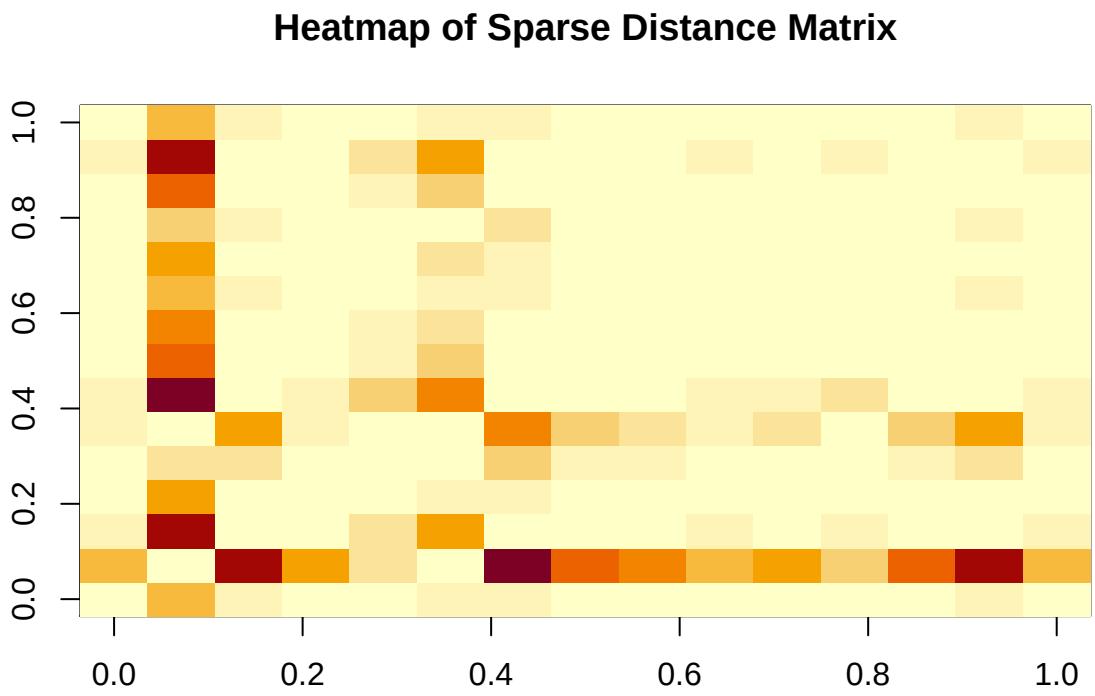
```
hc1$ws #weights
```

```
##             [,1]
## [1,] 0.01000225
## [2,] 0.00000000
## [3,] 0.99994998
```

```
#hc1$dists #gives the feature-wise dissimilarity matrix (nxn)*p
#hc1$u  %>% round(3)#this is the 'sparse' distance matrix


#hc1$dists %*% hc1$ws
#hc1$u

#heatmaps
#heatmap(hc1$u, main = "Heatmap of Sparse Distance Matrix")
image(hc1$u, main = "Heatmap of Sparse Distance Matrix")
```



Form of the nxn dissimilarity matrix u:

$u = (\sum_j w_j d_{ii'j})_{ii'}$

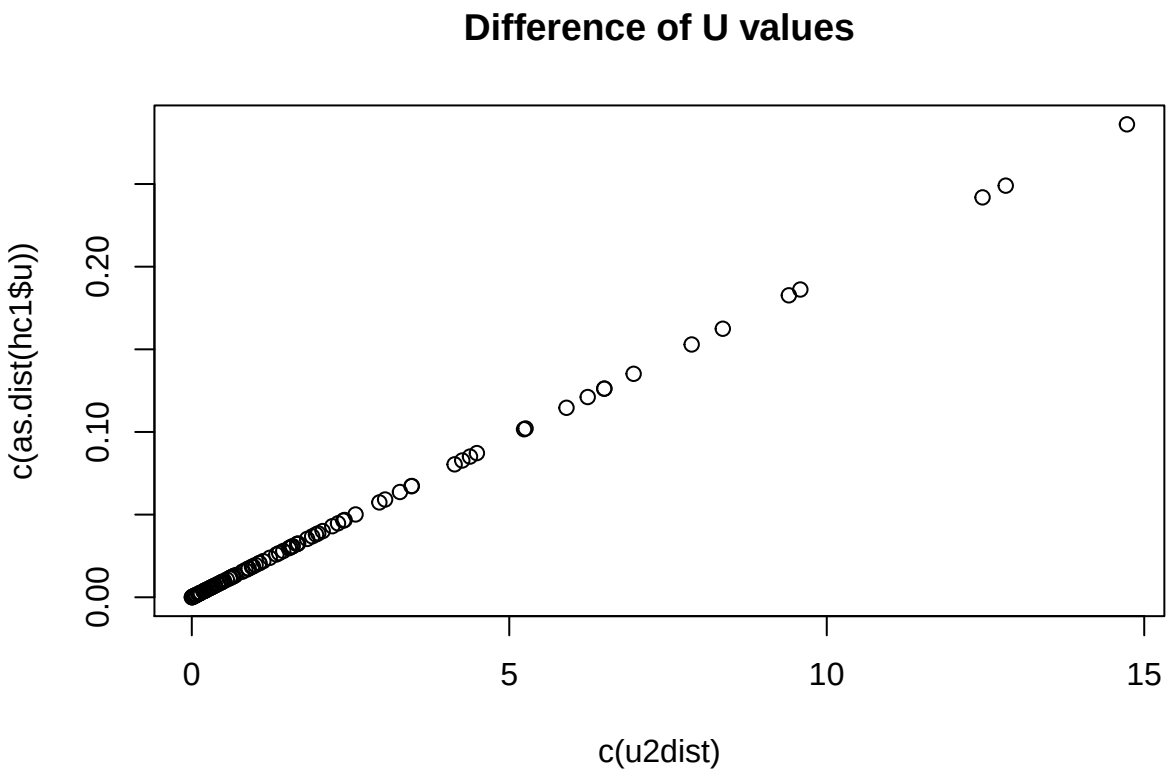
```
## Calculates the pairwise dissimilarities for EACH feature and vectorizes them
d1 <- c(dist(nba1[,1])^2)
d2 <- c(dist(nba1[,2])^2)
d3 <- c(dist(nba1[,3])^2)

#weights are given above - I think this is the lower/upper triangle
# Should be the form w_j * di,i',j
wv = (hc1$ws[1] * d1) + (hc1$ws[2] * d2) + (hc1$ws[3] * d3)
#u2 <- wv


u2 <- matrix(0, nrow = 15, ncol = 15)
u2[lower.tri(u2)] <- wv
u2[upper.tri(u2)] <- wv
u2dist <- as.dist(u2) # now in the form of distance matrix with 0's along diagonal

#image(u2, main = "Based on Original Data")


plot(c(u2dist), c(as.dist(hc1$u)), main = "Difference of U values")
```



Step 3: ADd the W's First and Calculate Dist

```
## Calculate the Weights:
hc1$ws
```

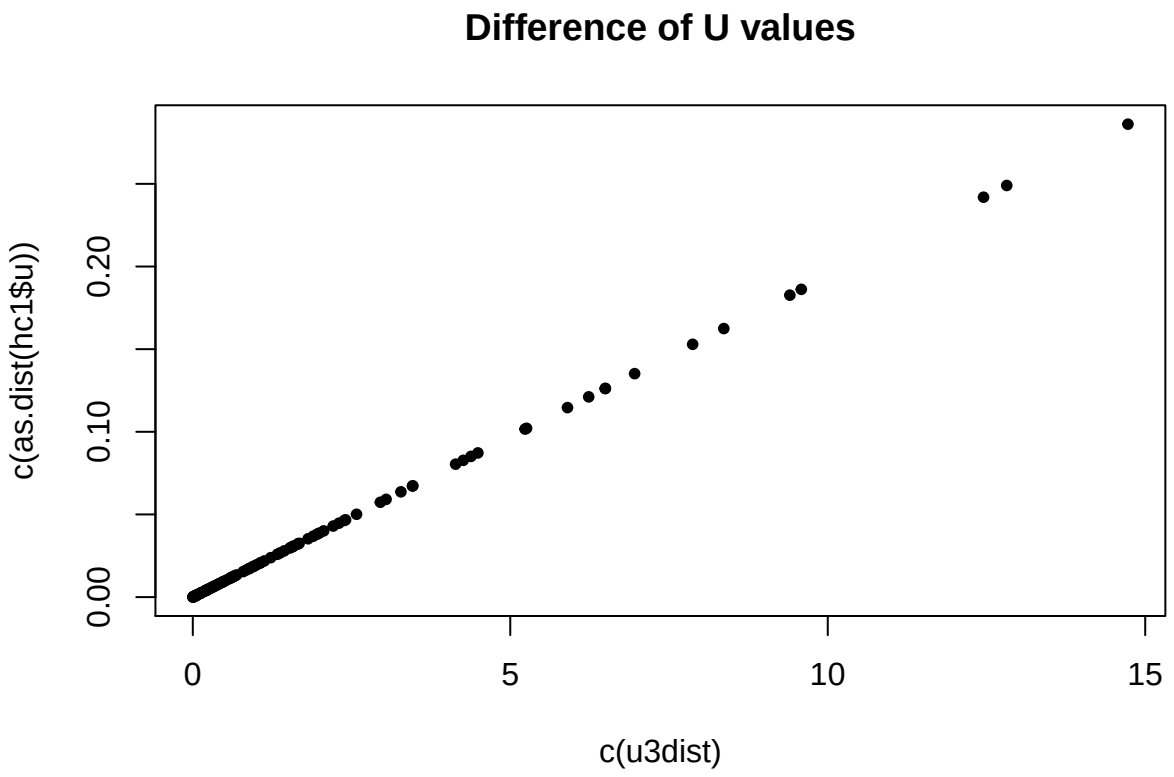
```
##             [,1]
## [1,] 0.01000225
## [2,] 0.00000000
## [3,] 0.99994998
```

```
# Multiply the weights times the original data features sqrt(wj)
nba_weight_1 <- sqrt(hc1$ws[1])*nba1[,1]
nba_weight_2 <- sqrt(hc1$ws[2])*nba1[,2]
nba_weight_3 <- sqrt(hc1$ws[3])*nba1[,3]

#Calculate New Distances
d1 <- c(dist(nba_weight_1)^2)
d2 <- c(dist(nba_weight_2)^2)
d3 <- c(dist(nba_weight_3)^2)

#Coerce into a Distance matrix
wv2 <- d1 + d2 + d3
u3 <- matrix(0, nrow = 15, ncol = 15)
u3[lower.tri(u2)] <- wv2
u3[upper.tri(u2)] <- wv2
u3dist <- as.dist(u3)
```

```r
# Plot for Comparison
plot(c(u3dist), c(as.dist(hc1$u)), main = "Difference of U values", pch = 20)
```



A Note on Documentation

In the documentation, it states that the $dists object is (nxn)xp. However, it not the case that this is nxn - this is created by calculating the pairwise disimilarities for each of the features - this is thus (nC2)xp.

```r
round(hc1$dists,3) == round(matrix(c(d1,d2,d3), ncol = 3),3)
```

```
##          [,1]  [,2]  [,3]
##    [1,] FALSE FALSE FALSE
##    [2,] FALSE FALSE FALSE
##    [3,] FALSE FALSE FALSE
##    [4,] FALSE FALSE FALSE
##    [5,] FALSE FALSE FALSE
##    [6,] FALSE FALSE FALSE
##    [7,] FALSE FALSE FALSE
##    [8,] FALSE FALSE FALSE
##    [9,] FALSE FALSE FALSE
##   [10,] FALSE FALSE FALSE
##   [11,] FALSE FALSE FALSE
##   [12,] FALSE FALSE FALSE
##   [13,] FALSE FALSE FALSE
##   [14,] FALSE FALSE  TRUE
```

```
##  [15,] FALSE FALSE FALSE
##  [16,] FALSE FALSE FALSE
##  [17,] FALSE FALSE FALSE
##  [18,] FALSE FALSE FALSE
##  [19,] FALSE FALSE FALSE
##  [20,] FALSE FALSE FALSE
##  [21,] FALSE FALSE FALSE
##  [22,] FALSE FALSE FALSE
##  [23,] FALSE FALSE FALSE
##  [24,] FALSE FALSE FALSE
##  [25,] FALSE FALSE FALSE
##  [26,] FALSE FALSE FALSE
##  [27,] FALSE FALSE FALSE
##  [28,] FALSE FALSE FALSE
##  [29,] FALSE  TRUE FALSE
##  [30,] FALSE FALSE FALSE
##  [31,] FALSE FALSE FALSE
##  [32,] FALSE FALSE FALSE
##  [33,] FALSE FALSE FALSE
##  [34,] FALSE FALSE FALSE
##  [35,] FALSE FALSE FALSE
##  [36,] FALSE FALSE FALSE
##  [37,] FALSE FALSE FALSE
##  [38,] FALSE  TRUE FALSE
##  [39,] FALSE FALSE FALSE
##  [40,] FALSE FALSE FALSE
##  [41,] FALSE FALSE FALSE
##  [42,] FALSE FALSE FALSE
##  [43,] FALSE FALSE FALSE
##  [44,] FALSE  TRUE FALSE
##  [45,] FALSE FALSE FALSE
##  [46,] FALSE FALSE FALSE
##  [47,] FALSE FALSE FALSE
##  [48,] FALSE FALSE FALSE
##  [49,] FALSE FALSE FALSE
##  [50,] FALSE FALSE FALSE
##  [51,] FALSE FALSE FALSE
##  [52,] FALSE FALSE FALSE
##  [53,] FALSE FALSE FALSE
##  [54,] FALSE FALSE FALSE
##  [55,] FALSE FALSE FALSE
##  [56,] FALSE FALSE FALSE
##  [57,] FALSE FALSE FALSE
##  [58,] FALSE FALSE FALSE
##  [59,] FALSE  TRUE FALSE
##  [60,]  TRUE FALSE FALSE
##  [61,] FALSE FALSE FALSE
##  [62,] FALSE FALSE FALSE
##  [63,] FALSE FALSE FALSE
##  [64,] FALSE FALSE FALSE
##  [65,] FALSE FALSE FALSE
##  [66,] FALSE FALSE FALSE
##  [67,] FALSE FALSE FALSE
##  [68,] FALSE FALSE FALSE
```

```
##  [69,] FALSE FALSE FALSE
##  [70,] FALSE FALSE FALSE
##  [71,] FALSE FALSE FALSE
##  [72,] FALSE FALSE FALSE
##  [73,] FALSE FALSE FALSE
##  [74,] FALSE FALSE FALSE
##  [75,] FALSE FALSE FALSE
##  [76,] FALSE FALSE FALSE
##  [77,] FALSE FALSE FALSE
##  [78,] FALSE FALSE FALSE
##  [79,] FALSE FALSE FALSE
##  [80,] FALSE FALSE FALSE
##  [81,] FALSE  TRUE FALSE
##  [82,] FALSE FALSE FALSE
##  [83,] FALSE FALSE FALSE
##  [84,] FALSE FALSE FALSE
##  [85,] FALSE FALSE FALSE
##  [86,] FALSE FALSE FALSE
##  [87,] FALSE FALSE FALSE
##  [88,] FALSE FALSE FALSE
##  [89,] FALSE FALSE FALSE
##  [90,] FALSE FALSE FALSE
##  [91,] FALSE FALSE FALSE
##  [92,] FALSE FALSE FALSE
##  [93,] FALSE FALSE FALSE
##  [94,] FALSE FALSE FALSE
##  [95,] FALSE FALSE FALSE
##  [96,] FALSE FALSE FALSE
##  [97,] FALSE FALSE FALSE
##  [98,] FALSE FALSE FALSE
##  [99,] FALSE FALSE FALSE
## [100,] FALSE FALSE FALSE
## [101,] FALSE FALSE FALSE
## [102,] FALSE FALSE FALSE
## [103,] FALSE FALSE FALSE
## [104,] FALSE FALSE FALSE
## [105,] FALSE FALSE FALSE
```

```
all.equal(hc1$dists, matrix(c(d1,d2,d3)))
```

```
## [1] "Attributes: < Component \"dim\": Mean relative difference: 1.962963 >"
## [2] "Mean relative difference: 0.9811699"
```

```
hc1$dists[1,1]
```

```
## [1] 2.201575
```

```
d1[1]
```

```
## [1] 0.0484802
```

Functionalization

```
## Wrapper for MonoClust
library(monoClust)
```

```
SparseMonoClust <- function(data, wbound = 1.1, dissimilarity = "squared.distance", nclusters = 3, scale

  ## Check Packages
  if(!require(sparcl)){install.packages('sparcl');library(sparcl)}
  if(!require(monoClust)){install.packages('monoClust');library(monoClust)}

  ## Optional Scaling of the original data
  if(scale ==TRUE){
    data <- apply(data, 2,scale)
  }

  ## First Step: Perform Sparse Clustering to Get W Vector
  hc1 <- HierarchicalSparseCluster(as.matrix(data), wbound = 1.01, dissimilarity = dissimilarity)
  ws <- sqrt(hc1$ws) #square root for squared Euclidean Distances

  ## Now - reweight the original data
  data_weight <- sweep(data, 2, ws, '*')


  ##
  mc <- MonoClust(as.data.frame(data_weight), nclusters = nclusters)

return(list(clustob = mc, u = data_weight, w = ws, sparclob = hc1))}
```

Some Testing on the function: Here we should see splits being made on the last feature and the first feature, (Pts and FG, respectively), as we are passing in a 0 weight on the ORB feature.

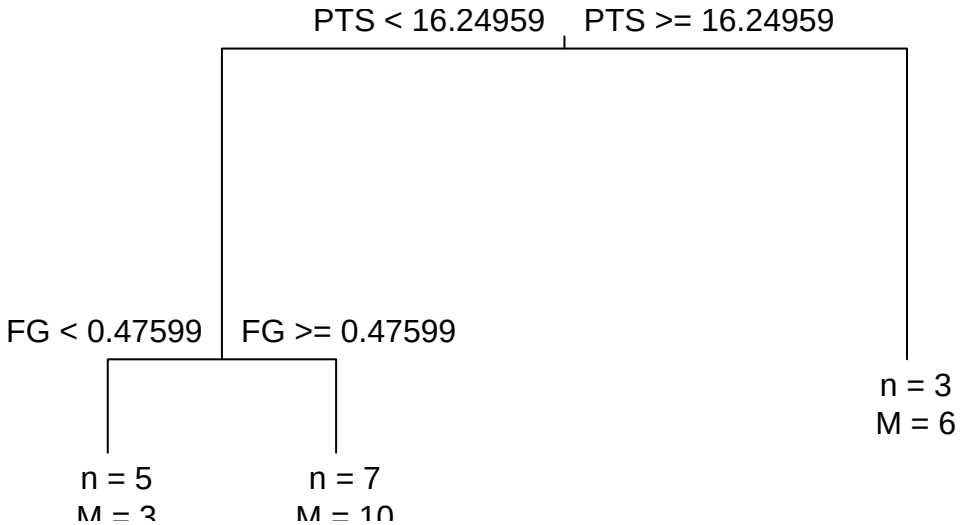
```
### Testing
nbatest <- select(nba, FG, ORB, PTS)
sp1 <- SparseMonoClust(nbatest, wbound = 1.1)
```

```
## 12
```

```
sp1$w
```

```
##           [,1]
## [1,] 0.1002095
## [2,] 0.0000000
## [3,] 0.9999748
```

```
plot(sp1$clustob)
```



More testing - this time with a slightly larger dataset.

```
nbatest2 <- select(nba, FG, ORB, PTS, Age, G, BLK, PF, DRB, AST, X3P)
sp2 <- SparseMonoClust(nbatest2, wbound = 1.1)
```

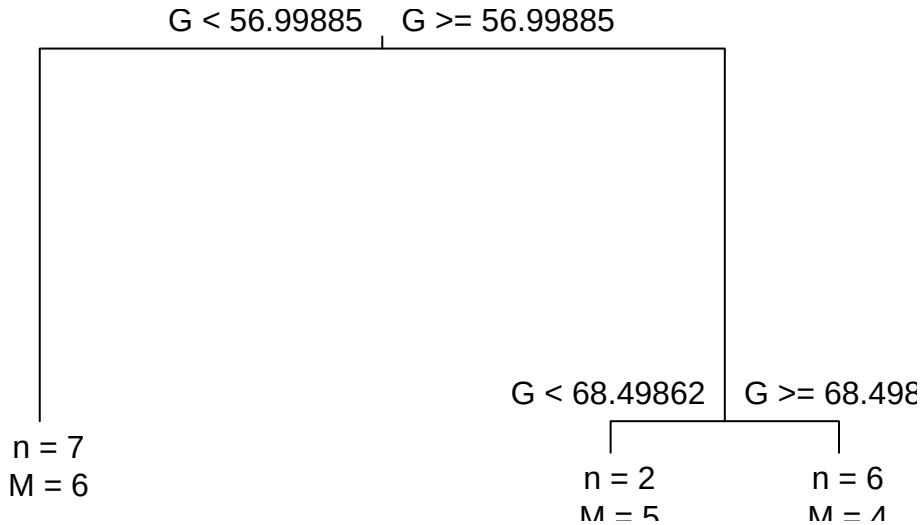
```
## 123
```

```
sp2
```

```
## $clustob
## n = 15
##
## Node) Split, N, Cluster Inertia, Proportion Inertia Explained,
##       * denotes terminal node
##
## 1) root 15 5080.54500 0.6684333
##   2) G < 56.9988512 7 1263.69600   *
##   3) G >= 56.9988512 8  420.84290 0.7250138
##     6) G < 68.4986194 2   24.55183   *
##     7) G >= 68.4986194 6  108.83170   *
##
## $u
##    FG ORB       PTS      Age        G BLK PF DRB AST X3P
## 1   0   0 0.4950290 2.265511 80.99837   0  0   0   0   0
## 2   0   0 0.7608164 2.454304 72.99853   0  0   0   0   0
## 3   0   0 0.3521683 2.737493 81.99835   0  0   0   0   0
## 4   0   0 0.4651279 3.303870 77.99843   0  0   0   0   0
## 5   0   0 0.5614759 2.265511 58.99881   0  0   0   0   0
```

```
## 6   0   0 0.6412121 2.831889 48.99901   0  0   0   0   0
## 7   0   0 0.3156225 3.209474 72.99853   0  0   0   0   0
## 8   0   0 0.4020034 1.982322 65.99867   0  0   0   0   0
## 9   0   0 0.4252598 2.454304 54.99889   0  0   0   0   0
## 10  0   0 0.4784173 2.359907 49.99899   0  0   0   0   0
## 11  0   0 0.4551609 1.982322 70.99857   0  0   0   0   0
## 12  0   0 0.5182854 2.359907 41.99915   0  0   0   0   0
## 13  0   0 0.4053258 2.454304 50.99897   0  0   0   0   0
## 14  0   0 0.3455236 2.265511 39.99919   0  0   0   0   0
## 15  0   0 0.4950290 2.171115 11.99976   0  0   0   0   0
##
## $w
##              [,1]
##  [1,] 0.00000000
##  [2,] 0.00000000
##  [3,] 0.03322342
##  [4,] 0.09439629
##  [5,] 0.99997984
##  [6,] 0.00000000
##  [7,] 0.00000000
##  [8,] 0.00000000
##  [9,] 0.00000000
## [10,] 0.00000000
##
## $sparclob
## Wbound is  1.01 :
## Number of non-zero weights:  3
## Sum of weights:  1.009974
```

```
plot(sp2$clustob)
```


G < 56.99885
G >= 56.99885
G < 68.49862
G >= 68.498
n = 7
M = 6
n = 2
n = 6