Cluster Analysis

Cavan Reilly

November 20, 2019

Table of contents

Overview

The goal of cluster analysis is to use multi-dimensional data to sort items into groups so that

1. items within the same group are similar across samples
2. items in distinct groups are dissimilar across samples

These groups are called "clusters".

In typical applications items are collected under different conditions and so one wants to find items that are similar as conditions change.

Overview

When items are genes and we have gene expression measurements for these genes, then we are looking for genes that go up or down together as conditions change.

We interpret such groups of genes as being part of a functional group whose activities are coordinated in response to biological stimuli.

For example, consider a collection of genes whose protein products are used at the start of cell replication.

Now consider an experiment where we take a collection of cells and promote repeated rounds of cell replication.

Overview

Suppose we collect a subset of these cells at regular time intervals and measure gene expression.

One would see the collection of genes whose protein products are used at the start of cell replication increase their level of gene expression at the start of replication, then decrease, then increase as the next round of cell replication starts.

Thus genes involved in the same cellular process would tend to move together over time.

Overview

More generally, if our samples were from different biological states then groups of genes involved in cellular processes that distinguish between these biological states would tend to "move together" across samples.

If the different states were "healthy" and "diseased" and there is a biological process that differed between the healthy and diseased states then these genes should form clusters, much like in the cell replication example.

As such, the goal of cluster analysis as applied to gene expression studies is much like the goal of creating the gene ontology: uncover groups of genes whose regulation is coordinated to meet some biological objective.

Overview

Hence the first question is what do we mean by similar and dissimilar.

We define similarity between 2 items in terms of the variables of the items: we assume that these are quantitative.

In the context of gene expression measurements, the items are usually genes and for each gene we measure gene expression for a number of conditions (or subjects).

So we measure distance in terms of how similar the gene expression measurements are for all conditions.

Distances

For example, consider 2 genes with gene expression levels given by x_{1i} and x_{2i} for $i, \ldots, n$ where n is the number of samples.

A simple measure of distance is just

$$d(x_1, x_2) = \sum_{i=1}^{n} |x_{1i} - x_{2i}|$$

If the level of gene expression was the same for a pair of genes across all conditions then this distance would be zero.

Distances

More generally we consider distance measures of the form

$$d(x_1, x_2) = \sum_i \left[|x_{1i} - x_{2i}|^p \right]^{\frac{1}{p}}$$

This is called the Minkowski distance with parameter p: if $p = 2$ we call this Euclidean distance.

There are other distances that are frequently used in certain applications.

Data Standardization

An important component of applying these methods is *data standardization*.

Typically one will alter the mean and standard deviation for all items so that all items have the same mean and standard deviation-usually mean 0 and standard deviation 1.

If one doesn't do this then the clusters one finds will usually just differ in terms of their overall expression level.

We don't want to find groups of genes that are simply expressed at higher levels over all conditions, we want to find groups that respond to stimuli in the same manner.

Data standardization

It is the correlation structure of genes across conditions that drives clusters: this isn't changed by standardizing the data across conditions.

We previously advocated for filtering based on the idea that many genes will not be expressed under some conditions.

Now there is an even greater need as cluster algorithms work much better with smaller data sets.

In fact many applications will first filter for testing, then test for differences across conditions, then use the results from testing as a filter prior to using cluster analysis.

Cluster Algorithms

If the goal is to assign clusters to minimize the average within cluster distance for a fixed number of clusters then there are only finitely many ways one can assign items to clusters.

Hence if we could just look at every way of assigning items to clusters we could find an assignment that minimizes the mean within cluster cluster distances: however there are too many possible assignments in typical applications.

This has lead to a tremendous number of algorithms for finding good clusters.

There are also many algorithms for trying to determine how many clusters to use.

Hierarchical clustering

In hierarchical clustering, one doesn't assign items to definitive clusters, rather one recursively groups items together so that items that are separated or brought together at some stage differ from other items in a similar fashion.

Agglomerative clustering-all items start as their own clusters and one successively merges clusters, merging clusters that are similar.

Divisive clustering-all items start in one big cluster and one splits off groups of items so that the items that are split off together are similar and different from the other items.

In order to achieve the goal of either of these sorts of algorithms one needs to define distances between clusters.

Hierarchical clustering

There are a number of commonly used methods for measuring distance between clusters (even more are available in R), these are:

1. single linkage
2. average linkage
3. complete linkage

An example

We previously examined an example looking for differences between 2 groups of cows using edgeR: we found that 130 transcripts differed at an FDR of 5%.

Here we will set up that data again and look at some different types of filters we can apply

```
> grp=factor(c(rep(1,5),rep(2,6)))
> bovCnts=read.table("bovineCounts.txt")

> bovCntsF1=bovCnts[apply(bovCnts,1,min)>4,]
> iqrs=apply(bovCnts,1,IQR)
> bovCntsF2=bovCnts[iqrs>median(iqrs),]
> bovCntsF3=bovCnts[iqrs>quantile(iqrs,.9),]
```

An example

Now we will read in a file with the ENSEMBL gene identities and use that to get some information on GO.

```
> bovIDs=scan("bov_ens_ids.txt",what="",sep="\n")
> library(biomaRt)
> mart=useMart("ensembl")
> ensembl=useDataset("btaurus_gene_ensembl",mart=mart)
> bov_bm=getBM(attributes=c('ensembl_gene_id',
+ 'hgnc_symbol','go_id','name_1006',
+ 'namespace_1003'),filters='ensembl_gene_id',
+ values=bovIDs,mart=ensembl)
```

An example

Now let's try some *t*-tests:

```
> f1=function(x){
+   t.test(x[grp==1],x[grp==2])$p.value
> }
> tt0=apply(log(bovCnts+1),1,f1)
> tt1=apply(log(bovCntsF1+1),1,f1)
> tt2=apply(log(bovCntsF2+1),1,f1)
> tt3=apply(log(bovCntsF3+1),1,f1)
> sum(p.adjust(tt0,method="BH")<0.1,na.rm=T)
[1] 0
> sum(p.adjust(tt1,method="BH")<0.1,na.rm=T)
[1] 0
> sum(p.adjust(tt2,method="BH")<0.1,na.rm=T)
[1] 0
> sum(p.adjust(tt3,method="BH")<0.1,na.rm=T)
[1] 0
```

An example

and try limma

```
> library(limma)
> mm=model.matrix(~grp)
> l0=eBayes(lmFit(bovCnts,design=mm))
> l1=eBayes(lmFit(bovCntsF1,design=mm))
> l2=eBayes(lmFit(bovCntsF2,design=mm))
> l3=eBayes(lmFit(bovCntsF3,design=mm))

> sum(p.adjust(l0$p.value[,2],method="BH")<.1)
[1] 0
> sum(p.adjust(l1$p.value[,2],method="BH")<.1)
[1] 0
> sum(p.adjust(l2$p.value[,2],method="BH")<.1)
[1] 0
> sum(p.adjust(l3$p.value[,2],method="BH")<.1)
[1] 0
```

An example

So let's stick with the edgeR analysis. Still multiple ways one can filter.

```
> delist <- DGEList(counts=bovCntsF1, group=grp)
> delist <- estimateCommonDisp(delist)
> delist <- estimateTagwiseDisp(delist)
> et <- exactTest(delist)
> padj <- p.adjust(et$table[,3], method="BH")
> sum(padj<0.05)
[1] 130
```

An example

```
> delist <- DGEList(counts=bovCntsF2, group=grp)
> delist <- estimateCommonDisp(delist)
> delist <- estimateTagwiseDisp(delist)
> et <- exactTest(delist)
> padj <- p.adjust(et$table[,3], method="BH")
> sum(padj<0.05)
[1] 166
```

An example

```
> delist <- DGEList(counts=bovCntsF3, group=grp)
> delist <- estimateCommonDisp(delist)
> delist <- estimateTagwiseDisp(delist)
> et <- exactTest(delist)
> padj <- p.adjust(et$table[,3], method="BH")
> sum(padj<0.05)
[1] 60
```

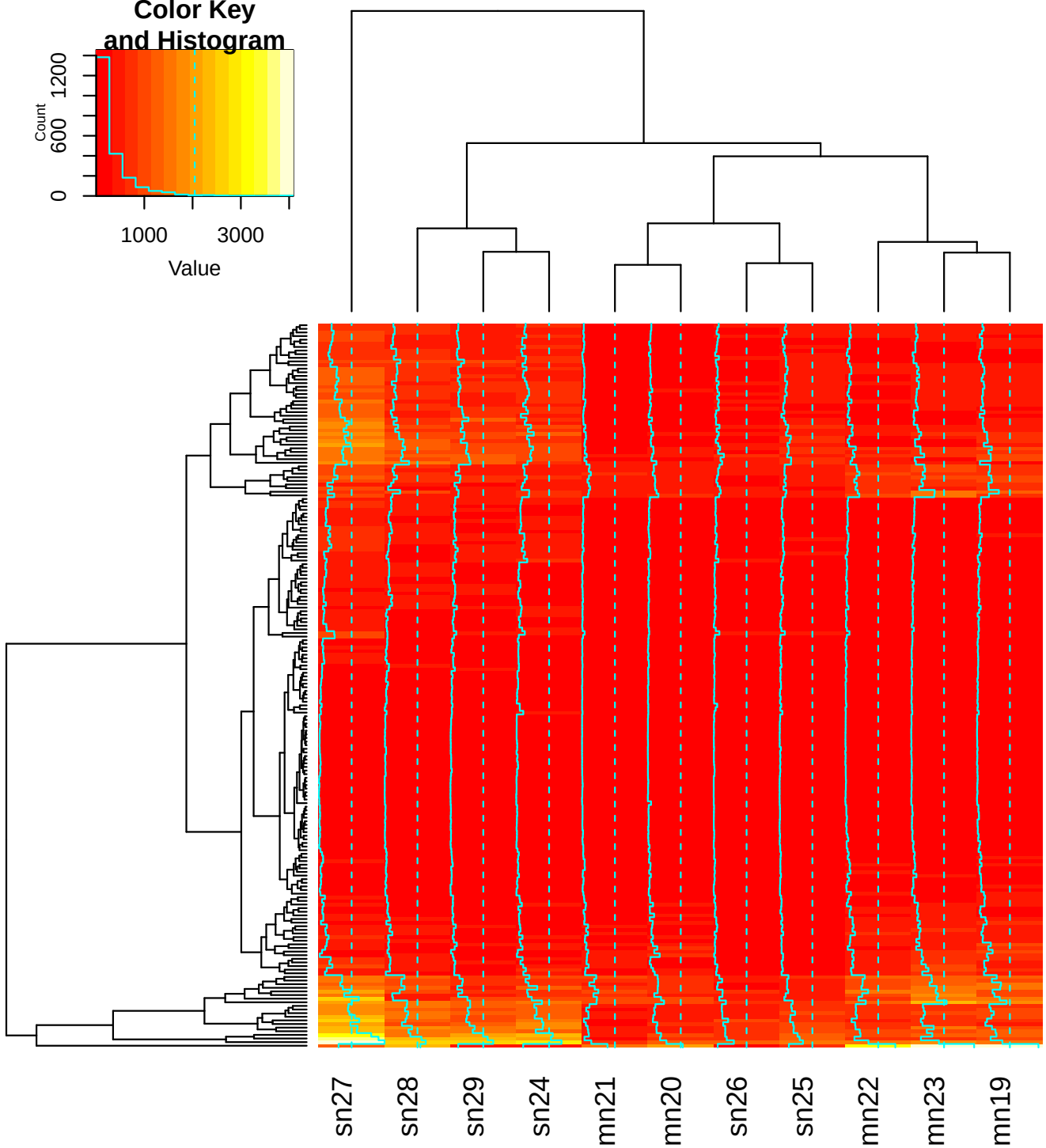
An example

So let's stick with the first analysis with 130 genes that differ.

A *heatmap* is a commonly used graphical technique for displaying data: here we will display the subset of genes that we are finding to differ.

Here we specify to not print the row labels-this means don't report the gene identifiers as there are too many to be able to read from the figure.

```
> library(gplots)
> colnames(bovCntsF1)=substr(names(bovCntsF1),1,4)
> pdf("heatmap1.pdf")
> heatmap.2(as.matrix(bovCntsF1[padj<0.1,]),labRow = F)
> dev.off()
```

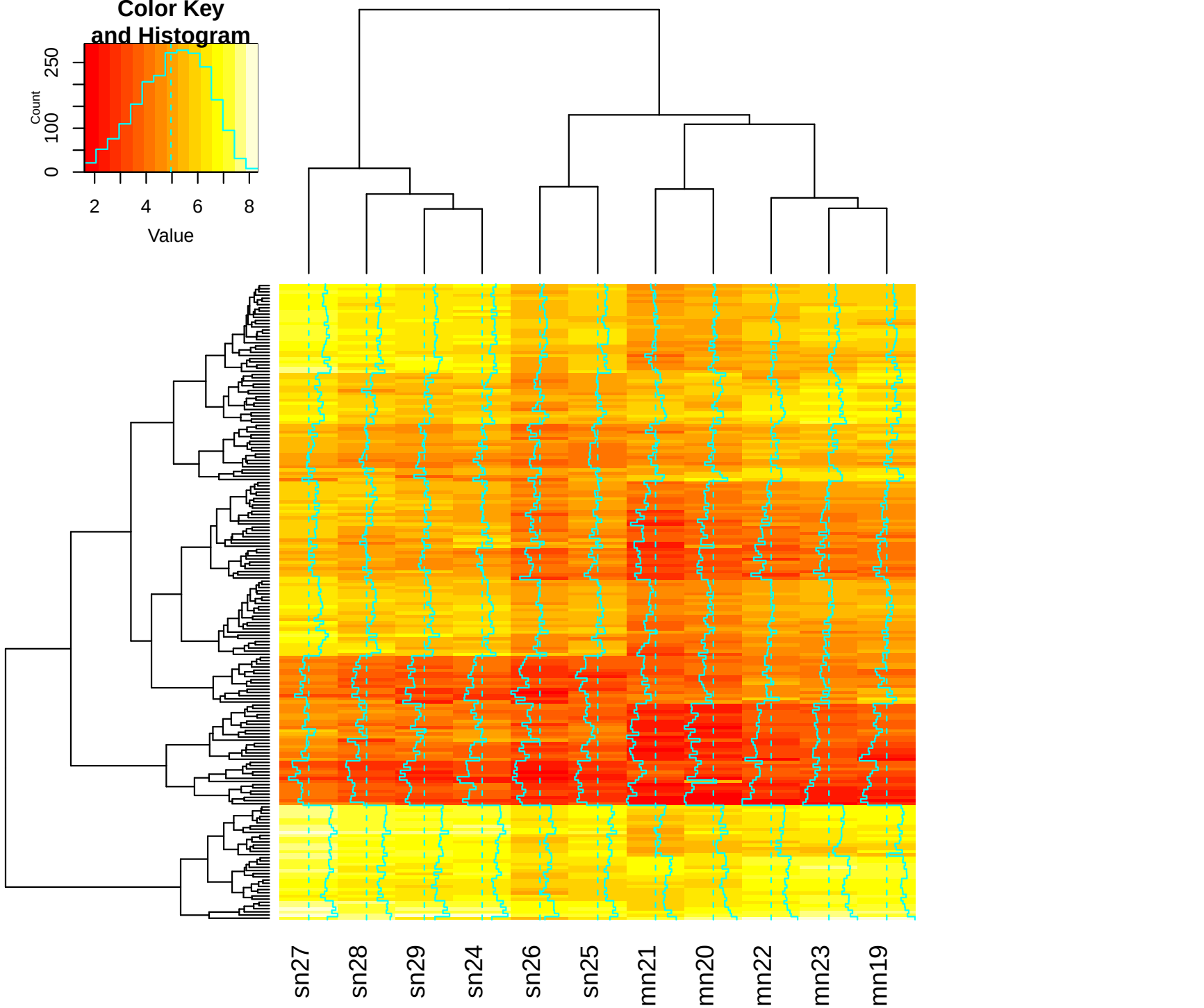

Color Key
and Histogram
Count
0
600
1200
1000
3000
Value
sn27
sn28
sn29
sn24
mn21
mn20
sn26
sn25
mn22
mn23
mn19

An example

Note that we don't separate the animals that well and the color key and histogram indicates that the distribution of gene expression is highly skewed-this all suggests taking the log first.

```
> pdf("heatmap2.pdf")
> heatmap.2(log(as.matrix(bovCntsF1[padj<0.1,])),
+ labRow = F)
> dev.off()
```

Which gives slightly better separation.


Color Key
and Histogram
Count
0
100
250
2
4
6
8
Value
sn27
sn28
sn29
sn24
sn26
sn25
mn21
mn20
mn22
mn23
mn19

An example

We will now apply some cluster analysis methods to this set of transcripts.

First select the genes that appear to differ, then standardize them so that all genes have mean zero and standard deviation 1.

```
> bovSub1=bovCntsF1[padj<0.05,]
> bovSub1s=as.matrix((bovSub1-apply(bovSub1,1,
+  mean))/apply(bovSub1,1,sd))
```

An example

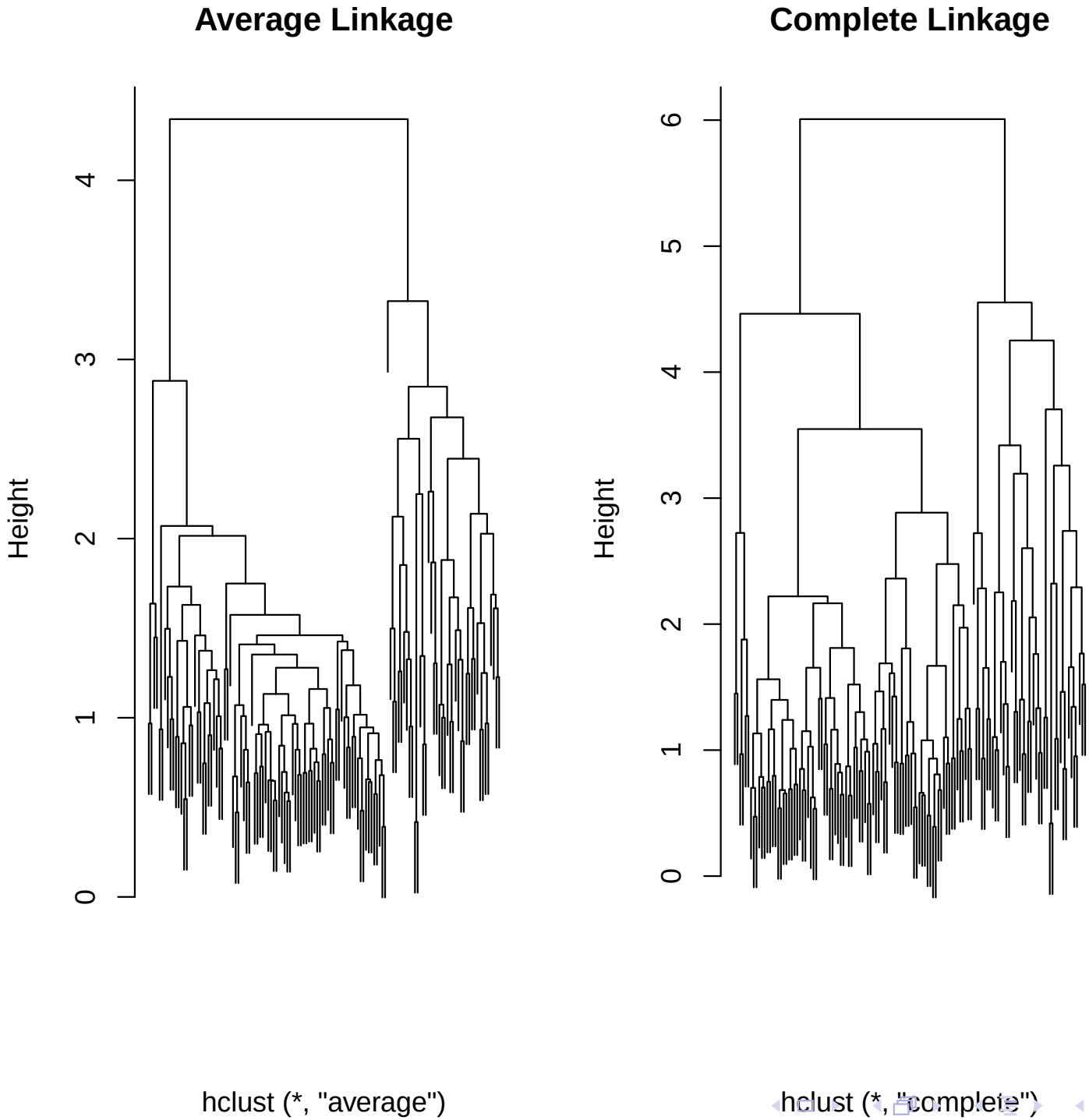
Next we compute the distance between all genes, then use the `hclust` function, which performs agglomerative hierarchical clustering: we consider both complete and average linkage

```
> d1=dist(bovSub1s)
> h1a=hclust(d1,method="average")
> h1c=hclust(d1,method="complete")
```

An example

We usually plot the results of this using *dendograms* which are tree like structures.

```
> pdf("hclust-plot.pdf")
> par(mfrow=c(1,2))
> plot(h1a,labels=F,xlab="",main="Average Linkage")
> plot(h1c,labels=F,xlab="",main="Complete Linkage")
> dev.off()
```


Average Linkage
Height
0
1
2
3
4
hclust (*, "average")
Complete Linkage
Height
0
1
2
3
4
5
6
hclust (*, "complete")

Cutting dendograms

We can also cut a dendogram off at a certain point to get an assignment of items to clusters.

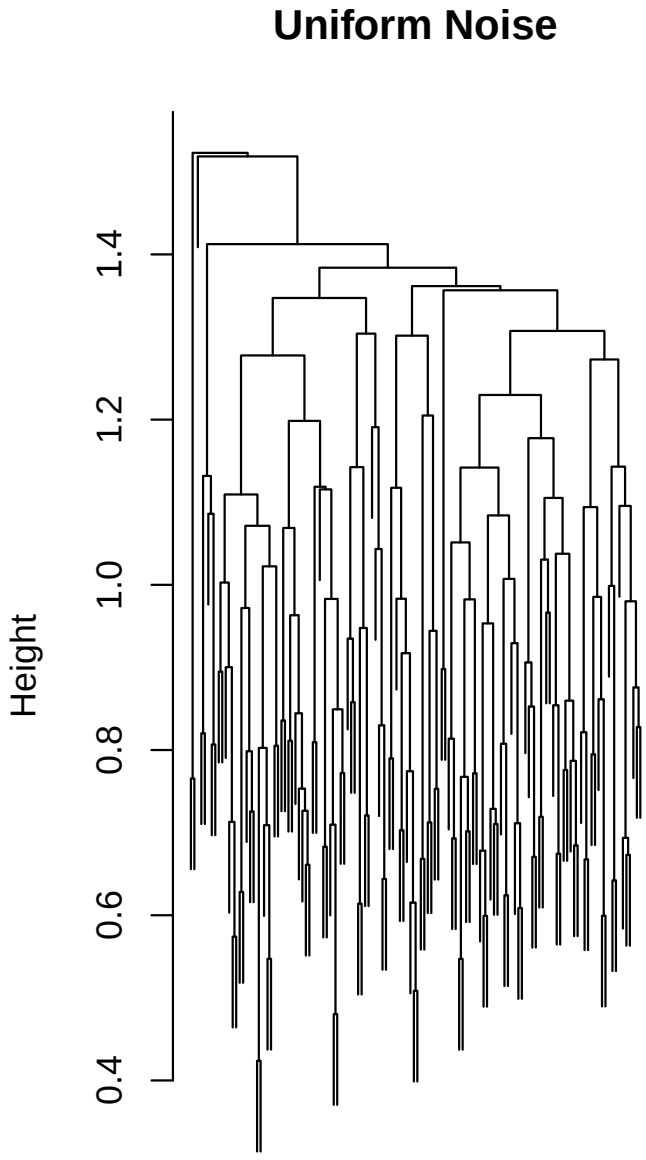
By default the `cutree` function cuts the tree based on the number of clusters but can use the `h` argument to cut at a certain height.

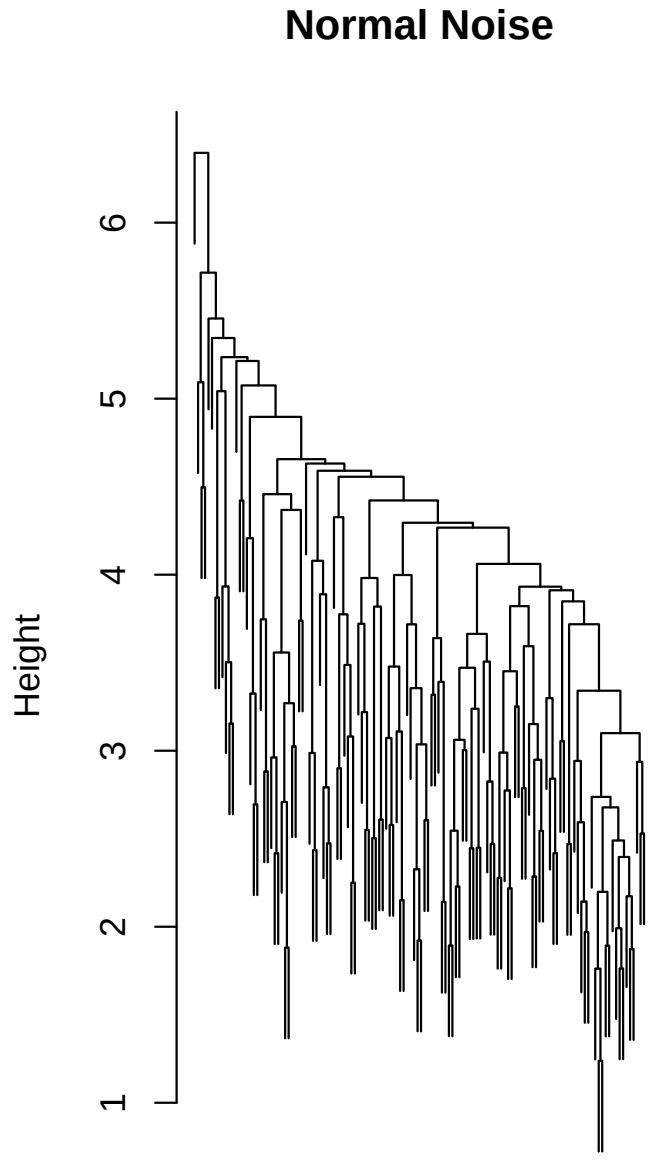
```
> ch1a=cutree(h1a,4)
> table(ch1a)
ch1a
 1  2  3  4
 1 84 41  4
> ch1a=cutree(h1a,h=1.8)
> table(ch1a)
ch1a
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
 1 60 22  5  2  2  4  7  4  4  3  2  3  2  3  4  1  1
```

Clustering Noise

So what would clustering noise look like: let's simulate uniformly and normally distribute data and try to cluster that

```
> noiseData1=matrix(runif(130*11),ncol=11)
> noiseData2=matrix(rnorm(130*11),ncol=11)
> dn1=dist(noiseData1)
> dn2=dist(noiseData2)
> hn1=hclust(dn1,method="average")
> hn2=hclust(dn2,method="average")
> pdf("noise-hclust.pdf")
> par(mfrow=c(1,2))
> plot(hn1,main="Uniform Noise",xlab="",labels=F)
> plot(hn2,main="Normal Noise",xlab="",labels=F)
> dev.off()
```


Uniform Noise
Height
0.4
0.6
0.8
1.0
1.2
1.4
hclust (*, "average")


Normal Noise
Height
1
2
3
4
5
6
hclust (*, "average")

Fastcluster

For large data sets `hclust` can be slow, so there is a package that has a faster implementation: `fastcluster`

We can experiment to see how much faster this is.

When you load this library it simply writes over the the `hclust` function, masking the usual function and thereby automatically substituting the newer version so no code has to change.

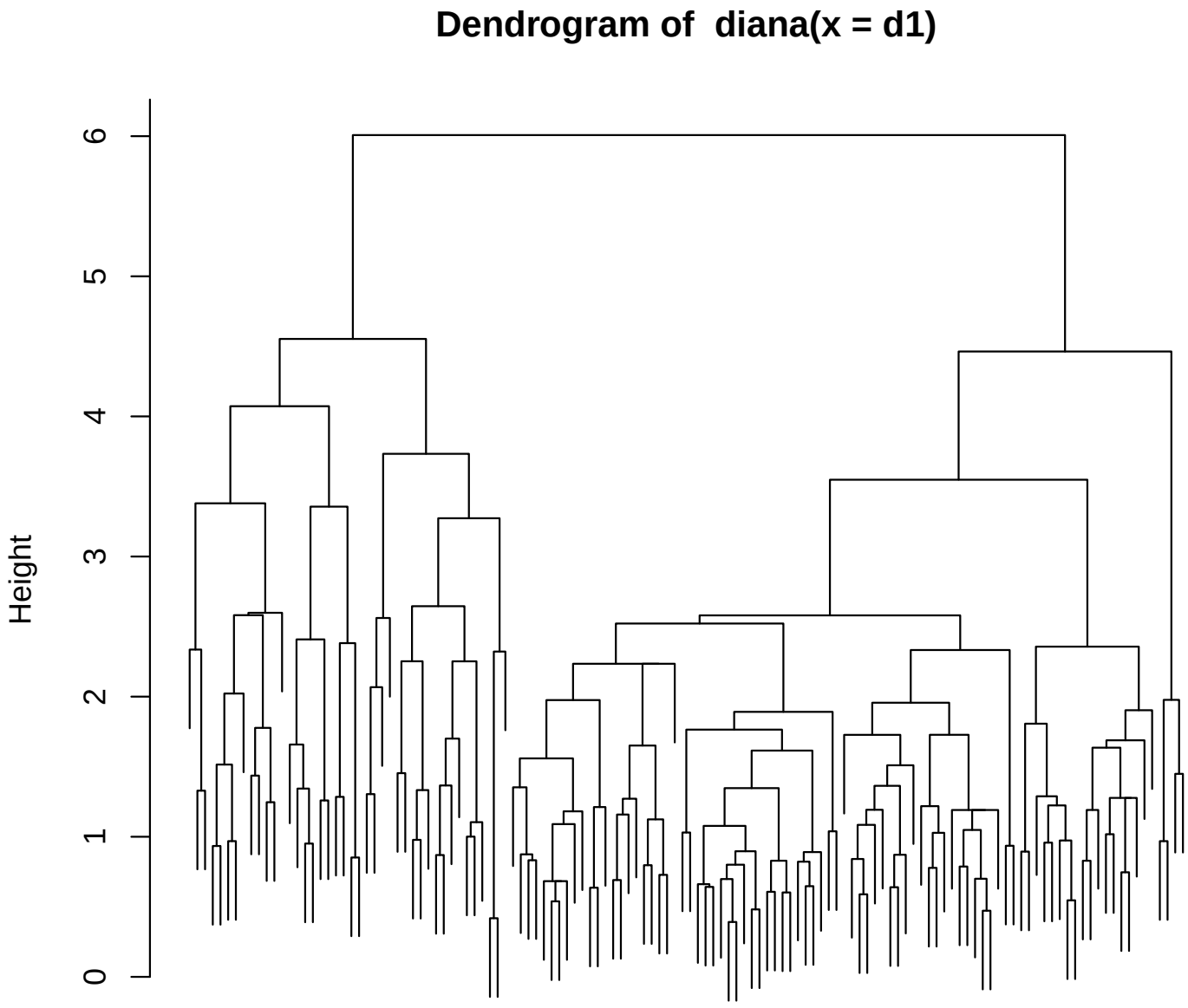
We can test this in terms of real time savings.

```
> ddb=dist(bovCntsF1)
> system.time(hclust(ddb))
> library(fastcluster)
> system.time(hclust(ddb))
```

Divisive Hierarchical Clustering

There are also methods for divisive clustering available in R: these are used less commonly.

```
> diana1=diana(d1)
> pdf("divisive-clust.pdf")
> plot(diana1,which.plots=2,labels=F,xlab="")
> dev.off()
```



Divisive Coefficient = 0.83

p-values for hierarchical clustering

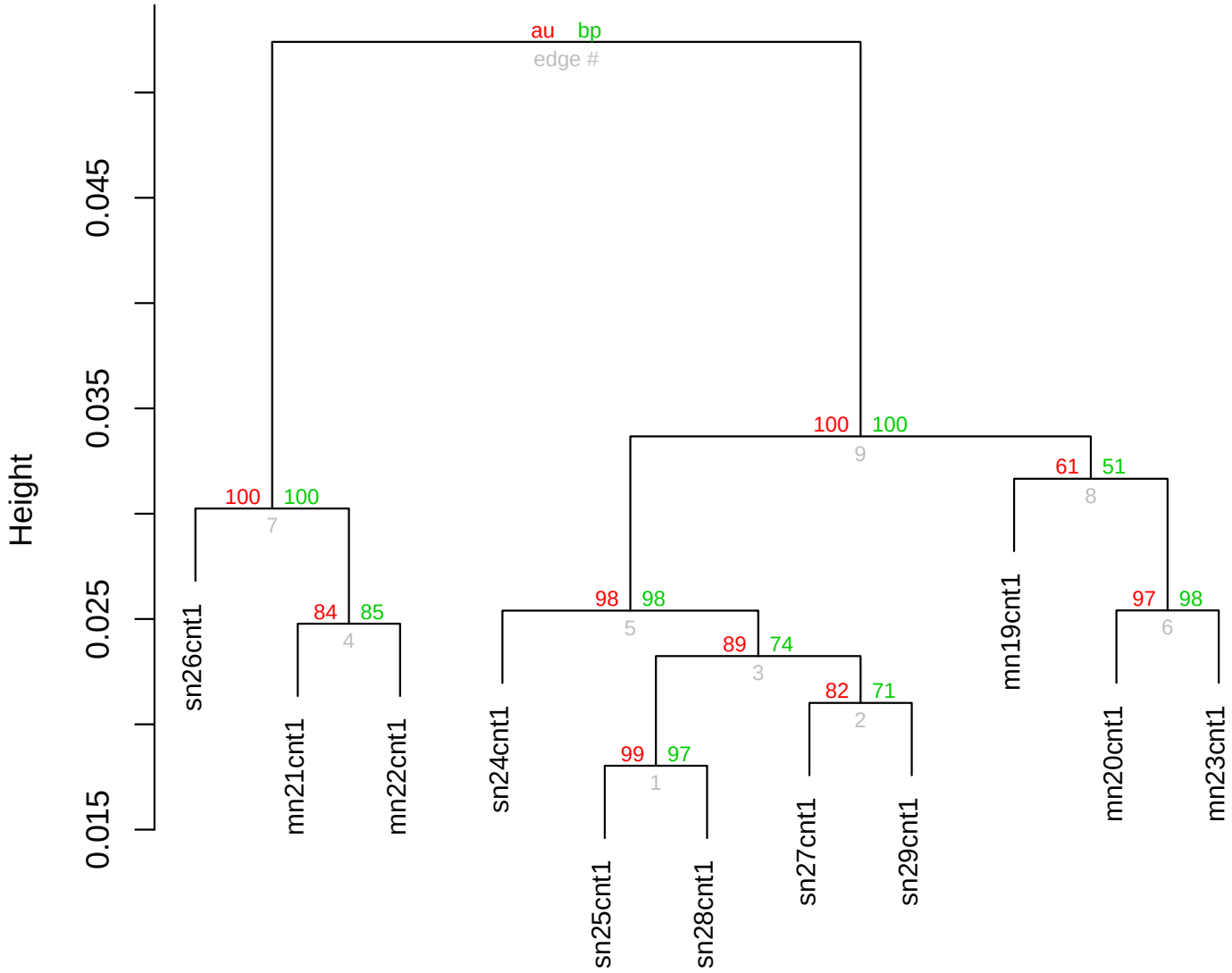
Here is an example where we cluster samples instead of genes.

With the pvclust package one can obtain *p*-values that test if there is evidence for distinct clusters at each split.

```
> library(pvclust)
> # this takes a couple of minutes
> bov.pv <- pvclust(bovCntsF1, nboot=1000)
> pdf("hier-pval.pdf")
> plot(bov.pv, cex=0.8, cex.pv=0.7)
> dev.off()
```

In this plot the red numbers indicate the support for the split using an unbiased estimate (the green values are less reliable): au is for approximately unbiased.

Cluster dendrogram with AU/BP values (%)



Distance: correlation
Cluster method: average

p-values for hierarchical clustering

We can also directly examine the *p*-values for the edges by examining the output of `pvclust` directly.

```
> bov.pv$edges[,1]
 [1] 0.9869109 0.8166574 0.8934202 0.8381512 0.9834966
 [6] 0.9651529 1.0000000 0.6077128 1.0000000 1.0000000
```

So not much evidence for clusters in this example.

k-means clustering

Perhaps the first algorithm for cluster analysis was the k-means algorithm.

The algorithm proceeds as follows: suppose I have an initial set of cluster centers.

1. for each item, determine which cluster center to which it is closest-assign that item to that cluster
2. once all items are assigned to clusters, compute the center of each cluster by finding the mean of all coordinates
3. go back to step 1 and reassign items to clusters

This process continues until no items are assigned to new clusters.

k-means clustering

Usually the algorithm converges quite quickly, perhaps too quickly.

It is advisable to use multiple starting points and select the solution to use based on some metric of cluster quality, such as mean within cluster variance.

The algorithm can fail if there is a cluster with no objects.

This can happen if there are too many clusters or if the choice of initial clusters is unfortunate.

This can best be avoided if the initial cluster centers are chosen to be the locations of actual items.

k-means clustering

Here is an example using our cow example, here we use 50 starting values

```
> set.seed(1245)
> k1=kmeans(bovSub1s,centers=4,nstart=50)
> k2=kmeans(bovSub1s,centers=4,nstart=50)
```

Then we can compare the solutions-we get the same answer in a sense

```
> table(k1$cluster)

 1  2  3  4
22 40 20 48
```

k-means clustering

```
> table(k2$cluster)

 1  2  3  4
40 20 22 48
> table(k1$cluster,k2$cluster)

    1  2  3  4
  1  0  0 22  0
  2 40  0  0  0
  3  0 20  0  0
  4  0  0  0 48
```

Partitioning around Medoids

A natural extension in some ways is to use the median instead of the mean.

This doesn't quite work out because the median is characterized by the observation with the value in the middle of the other observations, and this can't work with data in multiple dimensions.

Nonetheless there are multiple implementations of this idea.

Partitioning around Medoids

The function `pam` in the cluster library is one such implementation.

```
> p1=pam(bovSub1s,k=4)
```

We can see that this agrees pretty well with the k-means result.

PAM

```
> table(p1$clustering,k1$cluster)

    1  2  3  4
1  14  0  5  0
2   0 21  0 47
3   0 19  0  1
4   8  0 15  0
```

This is deterministic and will always provide the same solution-not clear how to specify different initial values.

PAM

One can permute the rows-this gives a different solution.

```
> permbov=bovSub1s[sample(1:130,replace=T),]
> p2=pam(permbov,k=4)
> table(p1$clustering,p2$cluster)
```

```
   1  2  3  4
1  9  2  5  3
2 36 11  8 13
3 13  3  3  1
4 10  6  4  3
```

PAM

So a little problematic that the solution is not invariant with respect to the order of the rows of the data matrix.

Nonetheless this is considered the go-to cluster analysis method in some circles.

There are also faster implementations of this method available through the `clara` function in the cluster package.

This works by sampling the data, finding clusters, then assigning the items not used for clustering to the cluster where they fit best.

Related techniques are used in the Spade package which is designed for the analysis of mass cytometry data sets.

Model based clustering

There are a large number of methods based on the connection between mixture models and cluster analysis.

A *mixture model* is a probability model for a continuous random variable which assumes that the probability density is a linear combination of simpler densities.

For example, a 2 component normal mixture distribution is a weighted average of 2 normal densities. For example, if $0 < \lambda < 1$

$$\lambda \mathrm{N}(\mu_1, \sigma_1^2) + (1 - \lambda)\mathrm{N}(\mu_2, \sigma_2^2)$$

is a univariate normal mixture distribution.

Model based clustering

We can draw samples from such a distribution by sampling which mixture distribution we draw from then sampling from that distribution.

```
> dev=rep(NA,100)
> for(i in 1:100){
+   comp=sample(1:2,1)
+   if(comp==1) dev[i]=rnorm(1,-1,1)
+   if(comp==2) dev[i]=rnorm(1,4,.5)
+ }
> hist(dev)
```

Model based clustering

So mixture models give rise to observations that exhibit clustering.

So we turn this idea around: if I want to fit a model to data that exhibits clustering I can fit a mixture model.

For more than one dimensional data I need to use what's called the *multivariate normal distribution*, which is an extension of the normal distribution to higher dimensions.

The big difference between the regular (univariate) normal distribution and the multivariate normal distribution is that the elements in the multivariate version can have non-zero correlation.

This correlation gives rise to different shaped clusters.

I can even have different correlation structures in my different clusters, so these are very flexible models.

Model based clustering

So rather than using some algorithm to find clusters I treat my observed data as observations from a multivariate normal mixture model and estimate the parameters in that model.

One can use the EM algorithm to estimate the parameters in these sorts of models.

We saw this when we talked about population structure in genetic association studies.

There we used the `mclust` package, as we will here.

This package considers a large number of possible models and selects the best one in terms of the Bayesian Information Criterion (BIC).

Model based clustering

The collection of models is as follows:

1. "E": equal variance (univariate)
2. "V": variable variance (univariate)
3. "EII": spherical, equal volume
4. "VII": spherical, unequal volume
5. "EEI": diagonal, equal volume and shape
6. "VEI": diagonal, varying volume equal shape
7. "EVI": diagonal, equal volume, varying shape
8. "VVI": diagonal, varying volume and shape
9. "EEE": ellipsoidal, equal volume, shape and orientation
10. "EEV": ellipsoidal, equal volume and shape
11. "VEV": ellipsoidal, equal shape
12. "VVV": ellipsoidal, varying volume, shape and orientation

Model based clustering

```
> m1=Mclust(bovSub1s)
> m1
'Mclust' model object:
 best model: ellipsoidal multivariate normal (XXX)
 with 1 components
```

So for this data set this algorithm concludes that there aren't really any clusters.

Self Organizing Maps

Self organizing maps is another algorithm that can be used to cluster items.

The basic idea: start with a collection of randomly initialized "cluster centers" and compare each item to be clustered to this collection of cluster centers-find the cluster center that is closest.

Modify the closest cluster center so that it is more similar to this item.

Slightly modify cluster centers that are also close to the item's closest cluster center.

Repeat many times, but as the algorithm proceeds, change the cluster centers less.

Self Organizing Maps

There are multiple packages that implement this algorithm: som and kohonen.

They provide the same basic functionality, but kohonen has more graphical options.

While there are a number of inputs one can adjust, the SOM grid is perhaps the most critical.

In many ways this is like specifying the number of clusters.

We will examine the impact of varying this in the following.

Self Organizing Maps

The `som` function provides the basic functionality: first we will cluster subjects.

```
> s1=som(bovSub1s,xdim=2,ydim=2)
> s2=som(bovSub1s,xdim=6,ydim=2)
> s3=som(bovSub1s,xdim=2,ydim=6)
```

The `code` output has information on how each item relates to the identified cluster centers: items with similar values for the codes are grouped together by the algorithm.

Self Organizing Maps

Here is how one can use `hclust` to obtain clusters of subjects from this output.

```
> som_cluster=cutree(hclust(dist(t(s1$code))),4)
> som_cluster
 [1] 1 2 2 1 1 3 2 2 4 3 3
```

and here is how to cluster genes

```
> s1a=som(t(bovSub1s),xdim=2,ydim=2)
> s2a=som(t(bovSub1s),xdim=6,ydim=2)
> s3a=som(t(bovSub1s),xdim=2,ydim=6)
> s4a=som(t(bovSub1s),xdim=20,ydim=40)
```

Self Organizing Maps

Then, as before, use hierarchical clustering on the codes

```
> som_cluster1a=cutree(hclust(dist(t(s1a$code))),4)
> som_cluster2a=cutree(hclust(dist(t(s2a$code))),4)
> som_cluster3a=cutree(hclust(dist(t(s3a$code))),4)
> som_cluster4a=cutree(hclust(dist(t(s4a$code))),4)
```

Then we can compare to results from the k-means algorithm.

```
> table(k1$cluster,som_cluster1a)
   som_cluster
     1  2  3  4
  1  0 40  0  0
  2  0 48  0  0
  3 13  0  0  7
  4  9  0 13  0
```

Self Organizing Maps

Other solutions also similar to k-means.

Note that when the product of the 2 dimensions is the same we get the same solution.

```
> table(som_cluster2a,som_cluster3a)
             som_cluster3a
som_cluster2a  1  2  3  4
            1 31  0  0  0
            2  0 70  0  0
            3  0  0 18  0
            4  0  0  0 11
```

Self Organizing Maps

Although we can get the same solution using very different som grid sizes.

```
> table(som_cluster1a,som_cluster4a)
             som_cluster4a
som_cluster1a  1  2  3  4
            1 22  0  0  0
            2  0 88  0  0
            3  0  0 13  0
            4  0  0  0  7
```

Determining the Number of Clusters

Determining the number of clusters is a notoriously difficult problem.

While many approaches have been developed none have ever been shown to be better than other approaches mathematically and probably can't be.

If one thinks of this problem in the context of fitting mixture distributions, then the likelihood is unbounded if one takes the number of clusters equal to the number of observations and has the variance of each cluster go to zero.

So the MLE of the number of clusters is the sample size.

One can impose constraints but then your solution will just be determined by the arbitrary constraints.

Determining the Number of Clusters

Nonetheless there is considerable interest in some sort of solution to this problem.

The package `NbClust` computes 30 different metrics for selecting the number of clusters, however you must tell it a method in which you are interested.

Usage is as follows: it automatically generates some plots.

```
> nb1=NbClust(bovSub1s,method="kmeans")
*******************************************************************
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 4 proposed 3 as the best number of clusters
* 3 proposed 4 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 1 proposed 6 as the best number of
...
```

Determining the Number of Clusters

So it appears that 2 is the best number of clusters, however this package, like most of the metrics that have been developed for this purpose, assumes there are at least 2 clusters.

So this does not provide an answer to the question of if there is any clustering at all in one's data.

Biclustering

Thus far we have largely discussed clustering genes as we are interested in finding genes that act in a coordinated fashion to carry out some biological function.

We may also be interested in the question of the extent to which our samples are similar to each other in terms of their gene expression profiles (i.e. the collection of gene expression measurements).

We can use all of the previously discussed methods to do this: we simply transpose our gene expression data matrix (which is the transpose of the usual statistical data matrix-so one needs to be careful).

But suppose one wants to cluster genes and samples at the same time: biclustering.

Biclustering

The biclust package has a wide selection of methods to choose from for biclustering.

Basic usage is a call to the function `biclust` and the user supplies the data matrix and `method=` for one of the available methods.

Some methods require additional preprocessing: we'll examine some that don't.

Some methods require manual specification of parameters.

```
> b1=biclust(bovSub1s,method=BCCC(),
+   delta=1.5,  alpha=1, number=10)
> b2=biclust(bovSub1s,method=BCPlaid())
> b3=biclust(bovSub1s,method=BCSpectral(),
+   numberOfEigenvalues=3)
Warning message:
In spectral(x, normalization, numberOfEigenvalues, minr,
  minc, withinVar) : No biclusters found
```

Biclustering

Then if we inspect the output we can see how many clusters were found.

```
> b1

An object of class Biclust
call:
biclust(x = bovSub1s, method = BCCC(), delta = 1.5, alpha = 1,
    number = 10)
There was one cluster found with
  130 Rows and  11 columns

> b2

An object of class Biclust
call:
biclust(x = bovSub1s, method = BCPlaid())
There was one cluster found with
  15 Rows and  3 columns
```

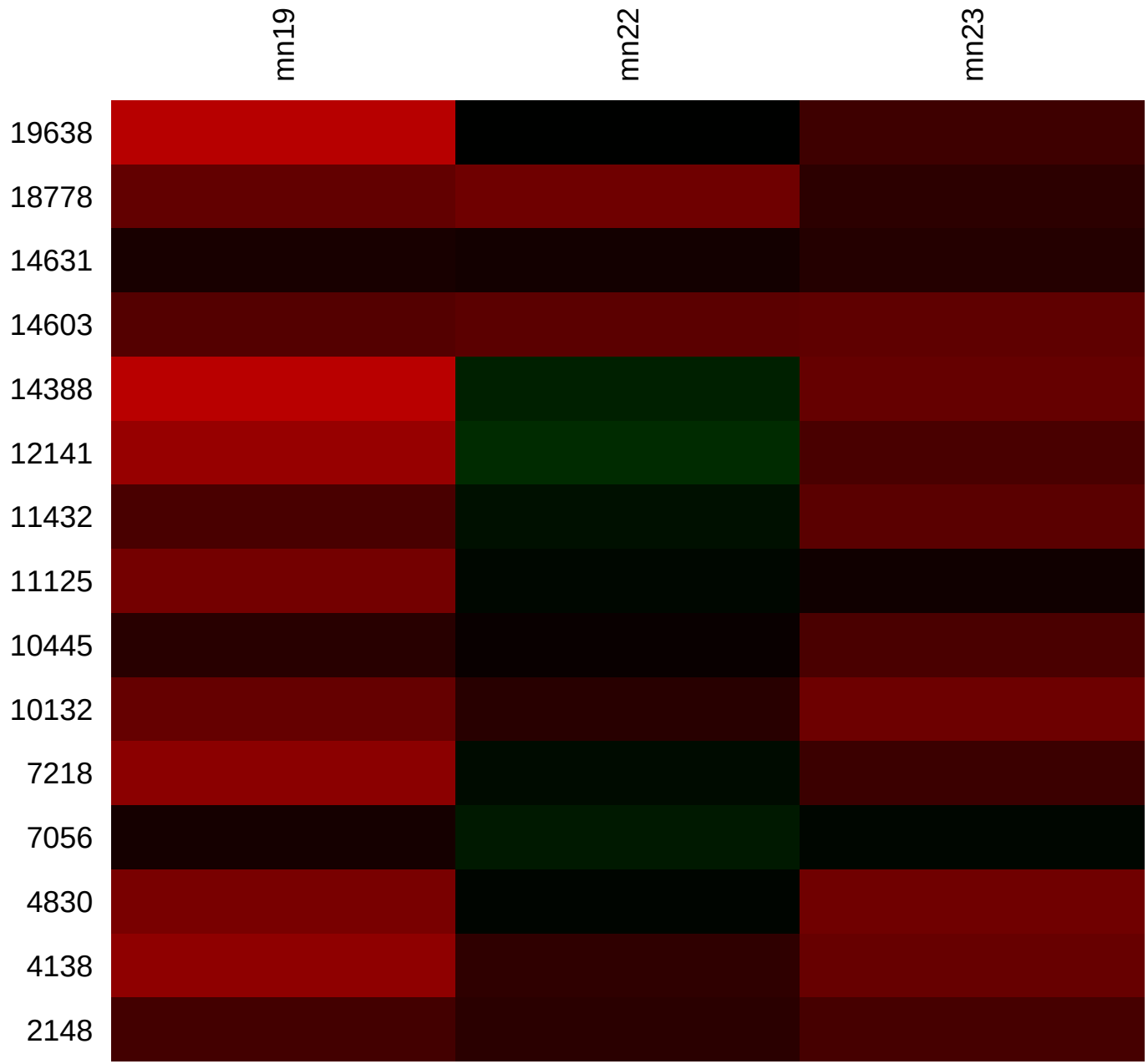
Biclustering

The package also has a function that can be used to visualize the results

```
pdf("biclust-hm-bccc.pdf")
drawHeatmap(bovSub1s,bicResult=b1,1)
dev.off()

pdf("biclust-hm-bcplaid.pdf")
drawHeatmap(bovSub1s,bicResult=b2,1)
dev.off()
```

Bicluster 1 (size 130 x 1)



Bicluster 1 (size 15 x 3)

Comparing cluster solutions

A number of metrics have been proposed for comparing the agreement among a set of clustering solutions.

The Rand index and its extensions (the adjusted Rand statistic and the corrected Rand statistic) examine how frequently 2 items are placed in the same cluster by 2 clustering solutions.

This functionality is implemented in multiple packages: for example in mclust.

Comparing cluster solutions

To use this you just extract the information on cluster assignment from the output of a cluster object.

```
> adjustedRandIndex(k1$cluster,k2$cluster)
[1] 1

> adjustedRandIndex(p1$clustering,p2$clustering)
[1] -0.001278062
```