BIO 754 - Lecture 12

11-05-2017

Contents

Functional enrichment of clusters

```
load("liver_transcriptome_v3.Rdata")  # nmat6, sex2, species2
load("liver_transcriptome_v4.Rdata")  # nmat6_aovp, nmat6_aovq
load("liver_transcriptome_v5.Rdata")  # nmat6_aovp_perm
load("liver_transcriptome_v6.Rdata")  # nmat6_aovp_perm2, toomuchrandomization, sexeffectconfounded
```

Interpreting cluster profiles

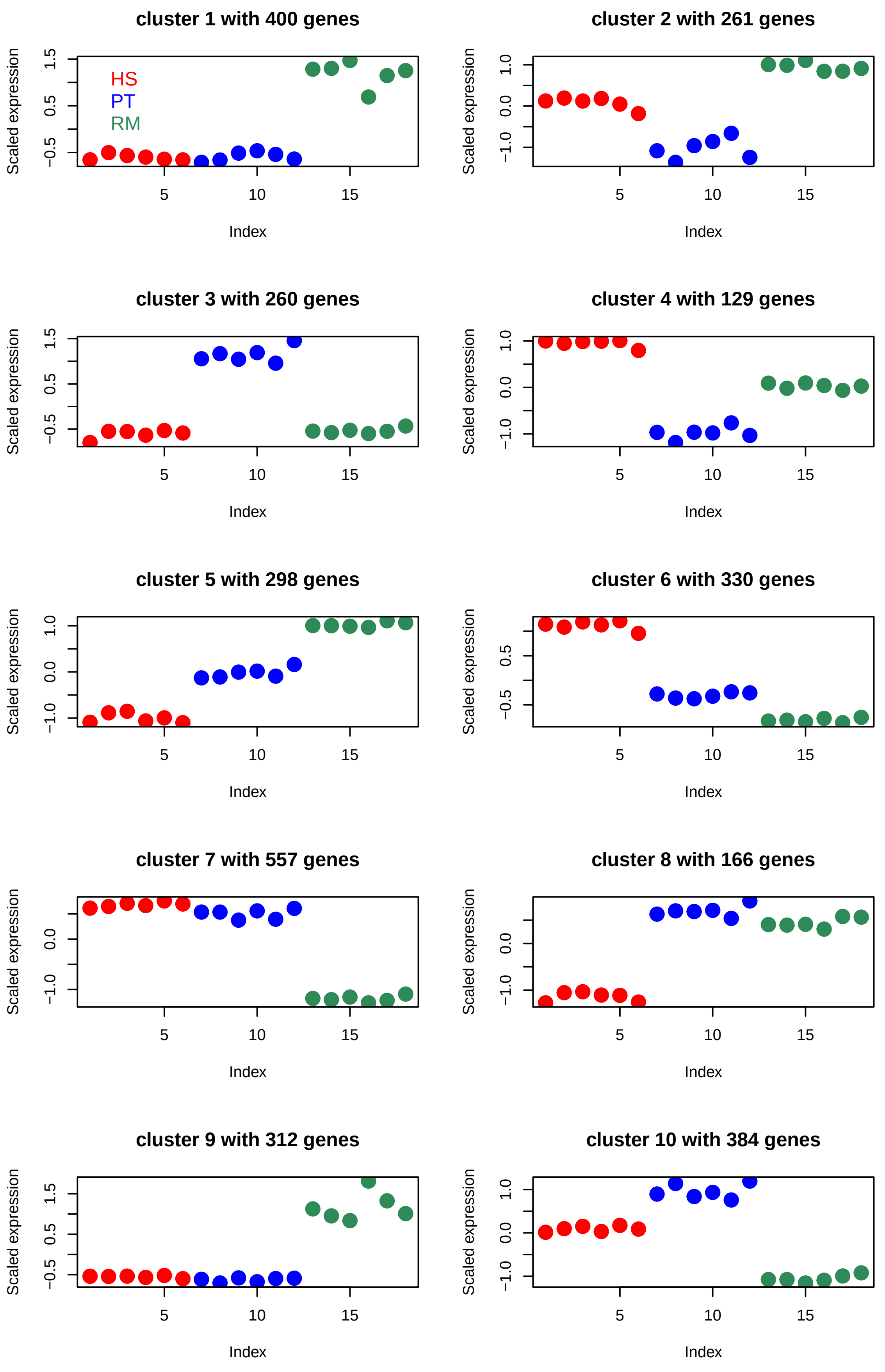
Now we will be testing the 2 clusters with *human-specific profiles* for enrichment in all GO categories. We can here use the `topGO` package. The package retrieves GO database information and runs Fisher's exact tests (also called hypergeometric tests). Note that we use only DE genes as **background**, which is more appropriate (more straightforward to interpret) than using all genes in the genome or all expressed genes (which is sometimes done). We then correct for multiple testing using the BY method.

Let's first recreate the kmeans clusters again and plot, using the code from last week:

```
snmat6 = t( scale( t( nmat6 ) ) )
spDEGenes = rownames(nmat6)[nmat6_aovq[,1] < 0.05]
set.seed(1)
km10 = kmeans( snmat6[spDEGenes,] , centers = 10)
# number of genes
km10$size
```

```
##  [1] 400 261 260 129 298 330 557 166 312 384
```

```r
# vector to group together species
reorder = c(which(species2 == 'HS'), which(species2 == 'PT'), which(species2 == 'RM') )
# color vector
colx = c("red", "blue", "seagreen")[as.numeric(species2)]
colx2 = colx[reorder]

par( mfrow=c(5,2) )
for( i in 1:10)  {
  maintext = paste( 'cluster', i, 'with', km10$size[i], 'genes')
  plot( km10$centers[i, reorder],
        col=colx2,
        pch=19, cex=2,
        main=maintext,
        ylab="Scaled expression")
  # plot the legend for the 1st graph
  if (i == 1) legend("topleft", text.col = unique(colx2), legend = levels(species2), bty="n", cex=1.2)
}
```


cluster 1 with 400 genes
HS
PT
RM
cluster 2 with 261 genes
cluster 3 with 260 genes
cluster 4 with 129 genes
cluster 5 with 298 genes
cluster 6 with 330 genes
cluster 7 with 557 genes
cluster 8 with 166 genes
cluster 9 with 312 genes
cluster 10 with 384 genes
Scaled expression
Index

Question 1: Are the cluster mean profiles trustable? This we can check by comparing the profiles of a few clusters and their members:

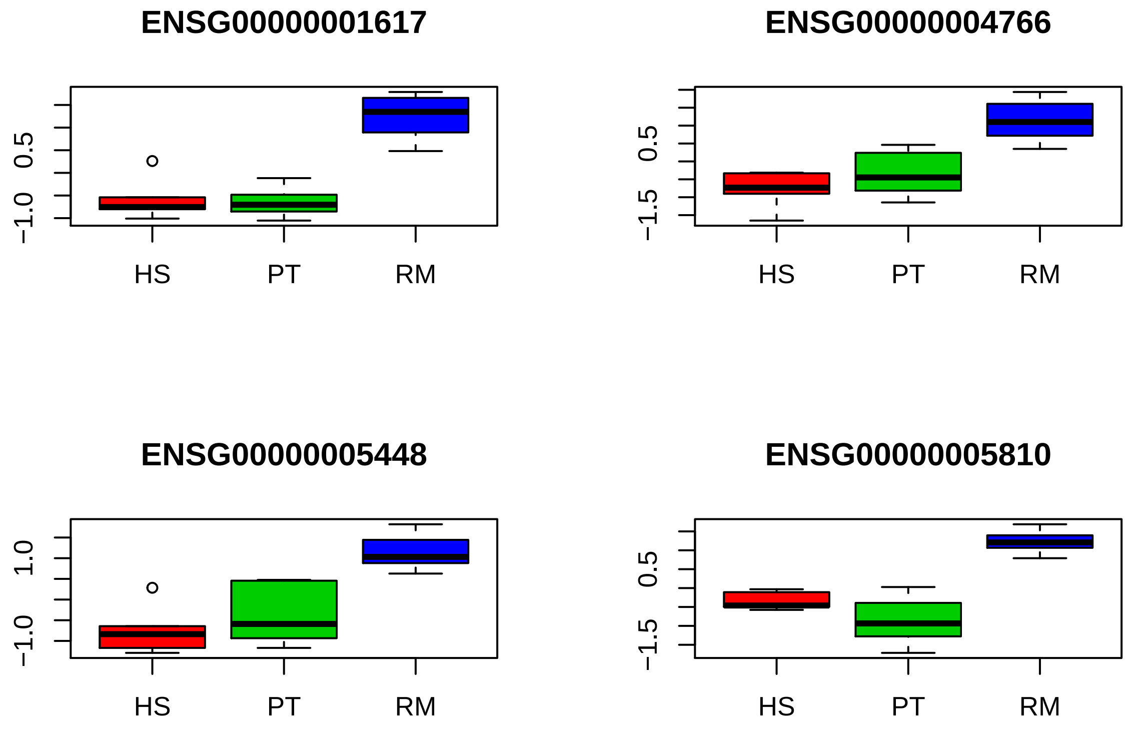
```r
# genes in cluster 1
head( which(km10$cluster == 1) )
```

```
## ENSG00000001617 ENSG00000004766 ENSG00000005448 ENSG00000005810
##               3              16              21              23
## ENSG00000006747 ENSG00000012963
##              33              61
```

```r
# their names
head( names( which(km10$cluster == 1) ) )
```

```
## [1] "ENSG00000001617" "ENSG00000004766" "ENSG00000005448" "ENSG00000005810"
## [5] "ENSG00000006747" "ENSG00000012963"
```

```r
# plot the first 4
par(mfrow=c(2,2))
for (i in 1:4) {
  genex = names(which(km10$cluster == 1))[i]
  boxplot(snmat6[genex,] ~ species2,
        col=2:4, main = genex)
}
```



Not bad. We do observe variation from gene to gene, but the pattern of macaque up-regulation is consistent.

Question 2: How to interpret these profiles?

Note that they represent groupings of genes with similar expression profiles (irrespective of genes' absolute expression levels). Also note that we used an **unsupervised clustering algorithm** (kmeans did not know about species identities), but we see that the cluster means mainly reflect species differences.

Some clusters are **species-specific**, such that one species is different from the others. E.g. cluster 8 genes represent human-specific down-regulation, in that the other two species behave differently. Other clusters, such as cluster 4, are mixed.

You may also notice that cluster 1 and 9 do not differ with respect to species differences, but the behaviour of one macaque individual (this happens to be the outlier identified earlier). Thus, the clustering will be influenced by any major source of variation in the data.

One note: this figure does not represent how much variation among genes there is **within clusters**. We could have added this info by e.g. calculating standard deviation among genes for each individual and adding this to the plot using the `segments` function.

Using the `topGO` package

Now we will learn to run Gene Ontology (GO) enrichment using a package. The test is the same as we had performed for `catabolism` on the 10th week, but performed on all GO Biological Process categories. The package is useful for obtaining all the GO information our genes of interest.

Packages can be very useful, but also *dangerous* in that the package's default behaviour may be different from what you would assume or prefer. E.g. is the package treating each GO independently, or taking into account dependence? Is it correcting for multiple testing or not? Does it require a minimum number of genes per GO group to run the test (a common method used to increase power)?

Usually you can find the information in the related paper and the package help documentation. For topGO this is helpful:

https://www.bioconductor.org/packages/devel/bioc/vignettes/topGO/inst/doc/topGO.pdf

```
library(topGO, verbose = )
```

```
## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##     IQR, mad, xtabs

## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##     as.data.frame, cbind, colnames, do.call, duplicated, eval,
##     evalq, get, grep, grepl, intersect, is.unsorted, lapply,
##     lengths, mapply, match, mget, order, paste, pmax, pmax.int,
##     pmin, pmin.int, rank, rbind, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit

## Loading required package: graph

## Loading required package: Biobase

## Welcome to Bioconductor
##
```

```
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package: GO.db

## Loading required package: AnnotationDbi

## Warning: package 'AnnotationDbi' was built under R version 3.3.1

## Loading required package: stats4

## Loading required package: IRanges

## Loading required package: S4Vectors

## Warning: package 'S4Vectors' was built under R version 3.3.1

##
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:base':
##
##     colMeans, colSums, expand.grid, rowMeans, rowSums

##

## Loading required package: SparseM

## Warning: package 'SparseM' was built under R version 3.3.2

##
## Attaching package: 'SparseM'

## The following object is masked from 'package:base':
##
##     backsolve

##
## groupGOTerms:    GOBPTerm, GOMFTerm, GOCCTerm environments built.

##
## Attaching package: 'topGO'

## The following object is masked from 'package:IRanges':
##
##     members
```

```r
# database connection
library(org.Hs.eg.db)
```

```
##
```

```r
# define the clusters
geneList = km10$cluster

# collect GO data for the 8th cluster genes and all DE genes
i = 8
GOdata = new( "topGOdata",
# run the analysis with Biological Process categories
              ontology = "BP",
# the universe of all DE genes (foreground + background)
              allGenes = geneList,
# select the foreground (cluster 8 genes)
```

```
               geneSel = function(p) { p == i },
               description = "Test", annot = annFUN.org,
               mapping="org.Hs.eg.db", ID="Ensembl")
```

```
##
## Building most specific GOs .....

##  ( 5256 GO terms found. )

##
## Build GO DAG topology .........

##  ( 9232 GO terms and 21899 relations. )

##
## Annotating nodes ...............

##  ( 2497 genes annotated to the GO terms. )
```

Now we can run the Fisher's exact test on each category, comparing cluster 8 genes with all other 3000+ DE genes. Note that using the `classic` algorithm runs the test on each GO group independently. Others, such as `weight01` will remove genes in child terms from higher terms:

```
resultFisher = runTest( GOdata, algorithm = "classic", statistic = "fisher")
```

```
##
##          -- Classic Algorithm --
##
##      the algorithm is scoring 2544 nontrivial nodes
##      parameters:
##          test statistic: fisher
```

```
resultFisher
```

```
##
## Description: Test
## Ontology: BP
## 'classic' algorithm with the 'fisher' test
## 9232 GO terms scored: 60 terms with p < 0.01
## Annotation data:
##     Annotated genes: 2497
##     Significant genes: 136
##     Min. no. of genes annotated to a GO: 1
##     Nontrivial nodes: 2544
```

Here `Annotated genes` refers to all DE genes with GO annotation. `Significant genes` refers to cluster 8 genes (the foreground) with GO annotation. `Nontrivial nodes` refers to the number of GO categories with min. one `Annotated genes` assigned to them.

```
# number of "Nontrivial nodes" ("SigTerms" is a misnomer)
numberGO = resultFisher@geneData["SigTerms"]
numberGO
```

```
## SigTerms
##     2544
```

```
# make a summary of the results, for all GO groups
gores = GenTable(GOdata, classicFisher = resultFisher, topNodes = numberGO)
dim(gores)
```

```
## [1] 2544    6
```

```
head(gores, 10)
```

```
##         GO.ID                                        Term Annotated
## 1  GO:0045184       establishment of protein localization       346
## 2  GO:0015031                           protein transport       316
## 3  GO:0008104                        protein localization       420
## 4  GO:0006412                                 translation        88
## 5  GO:0043043                peptide biosynthetic process        91
## 6  GO:0019058                            viral life cycle        59
## 7  GO:0051702                     interaction with symbiont         5
## 8  GO:0051851 modification by host of symbiont morphol...         5
## 9  GO:0043604                  amide biosynthetic process       107
## 10 GO:1903900               regulation of viral life cycle        25
##    Significant Expected classicFisher
## 1           34    18.85       0.00025
## 2           31    17.21       0.00053
## 3           38    22.88       0.00057
## 4           13     4.79       0.00074
## 5           13     4.96       0.00102
## 6           10     3.21       0.00104
## 7            3     0.27       0.00146
## 8            3     0.27       0.00146
## 9           14     5.83       0.00158
## 10           6     1.36       0.00175
```

How to interpret the table? E.g. among all DE genes n=346 were associated with "establishment of protein localization". Becase cluster 8 is small (~1/20 of all DE genes), under the H0 that GO categorization and being in cluster 8 are independent, we also *expect* ~1/20 of the 346 (<20 genes) in "establishment of protein localization" in cluster 8. But we find >30, which appears significant in the Fisher's exact test. Thus we *might* reject the H0 of independence (but see below).

This may be interesting. It shows that among genes down-regulated in humans, protein transport/localization and some viral-related functions are enriched. Here two points worth mention:

Dependence among GO groups

The test here did not correct for the fact that the GO has a tree-like structure and categories are listed within each other. Thus genes in "establishment of protein localization" are also members of "protein localization". Therefore an enrichment signal arising from a lower GO group will also spill into upper groups. This can be likened to chimpanzees being known as aggressive compared to other apes, although it is mainly adult male chimps who are aggressive. If you tested all chimps agains all other ape species, you would find a difference on average, but if you had removed male chimps, perhaps there might be no difference left. Alternative algorithms in topGO such as `elim` or `weight01` run the test, removing lower GO term members of significant groups from upper terms.

Multiple testing

Note that we ran 2000+ tests and normally we need to correct for multiple testing. The p-values listed are only **nominal** and cannot be taken at face value. Although `topGO` authors suggest to be liberal with respect to correction, many other biologists would believe this is necessary.

Now we'll run multiple testing correction using the `BY` method (which does not assume independence of tests either), and collect the most extreme terms:

```r
qval = p.adjust( gores$classicFisher, met='BY')
gores2 = cbind(gores, qval)
head( gores2 )
```

```
##        GO.ID                                   Term Annotated Significant
## 1 GO:0045184 establishment of protein localization       346          34
## 2 GO:0015031                      protein transport       316          31
## 3 GO:0008104                   protein localization       420          38
## 4 GO:0006412                            translation        88          13
## 5 GO:0043043          peptide biosynthetic process        91          13
## 6 GO:0019058                      viral life cycle        59          10
##   Expected classicFisher qval
## 1    18.85       0.00025    1
## 2    17.21       0.00053    1
## 3    22.88       0.00057    1
## 4     4.79       0.00074    1
## 5     4.96       0.00102    1
## 6     3.21       0.00104    1
```

So as you notice, the Fisher's exact test results are not significant after multiple testing correction. In other words, we have no strong evidence that these genes down-regulated in the human liver compared to those of other primates, have specific functions.

Other functional effects

- In addition to using GO, we could also have tested for **common regulatory properties**: e.g. transcription factor binding sites in promoters, common enhancer types, miRNA binding sites, etc.
- We could have compared our data with other datasets, such as the mouse diet experiment: Are human-chimpanzee differences correlated with those induced by diet in mouse liver?
- We could check the evolutionary properties of these genes. E.g. Are genes that are differentially expressed in the liver among primates evolving faster than non-DE genes in their protein coding sequence?

Choice of background in enrichment tests

We tested enrichment in cluster 8 genes compared to all other DE genes with GO annotation. The former set is sometimes called **foreground** and the latter set the **background**. It is important how we choose our background. This could theoretically be:

- all genes in Ensembl (60k genes, including non-protein coding genes and pseudogenes),
- all 1:1 orthologs in human, chimp and macaque (the 20k genes in the original dataset),
- all expressed orthologs in the primate liver (15k).

The background sets actually reflect your null hypothesis. One will find different results using different backgrounds.

In our case, we want to know whether cluster 8 genes, compared to all other possible patterns of differential expression, have unique functional properties. Thus, it makes sense that our background is all DE genes, but not other sets. Otherwise we might find significant GO enrichment in cluster 8, but that reflects being a protein-coding gene, having primate ortologs, being expressed in the liver, but *not* being down-regulated in humans!

Take this example: If I ask whether **people with a beard tend to smoke more often** than average, what should be my background? It should be other adult males, not all individuals. Otherwise, a significant enrichment signal I may find could be just reflecting the fact that males (or adults) smoke more often.

Pseudoreplication in transcriptome/genome analyses

A commonly ignored problem with this functional enrichment test strategy (running the Fisher's exact test and applying multiple testing correction) is that it treats each gene as an **independent observation**. But as we have seen, genes are co-regulated and show similar expression patterns - gene expression patterns are by default *not* independent. This violates the assumption of the test. Plus, it cannot be corrected by multiple testing.

An alternative approach (which we will not perform here) is **permuting the individual labels**, sorting genes based on the p-value, choosing the same number of genes as originally chosen, repeating the enrichment, and comparing how many GO categories show significant enrichment in the real case vs. permutations. This should be yield more reproducible results.

Chromosomal enrichment

The last exercise is testing for chromosomal enrichment. This was part of the homework, and we repeat it here.

Let's test the putative sex-related DE genes for sex chromosomal enrichment. If these genes are indeed non-significant / random, we expect no enrichment. But perhaps they are not.

Here we will obtain the chromosome location table for Ensembl genes using the `biomaRt` package, and run our own Fisher's exact test.

```r
library(biomaRt)
# connect to the database
ens = useMart("ensembl", dataset = "hsapiens_gene_ensembl")
ens_chr = unique(getBM(attributes=c('ensembl_gene_id', 'chromosome_name'), values=T, mart=ens))
save(ens_chr, file="ens_chr.RData")
```

```r
load("ens_chr.RData")
head(ens_chr)
```

```
##   ensembl_gene_id chromosome_name
## 1 ENSG00000264452              21
## 2 ENSG00000278324               8
## 3 ENSG00000283502               2
## 4 ENSG00000241226              19
## 5 ENSG00000252604               4
## 6 ENSG00000274494               6
```

```r
dim(ens_chr)
```

```
## [1] 63898     2
```

```r
summary(ens_chr)
```

```
##  ensembl_gene_id    chromosome_name
##  Length:63898       Length:63898
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

```r
length(unique(ens_chr[,1]))
```

```
## [1] 63898
```

```r
# how come the table is so large?
length(unique(ens_chr[,2]))
```

```
## [1] 381
```

```r
head( table( ens_chr[,2]), 30)
```

```
## 
##                        1                       10 
##                     5224                     2208 
##                       11                       12 
##                     3248                     2952 
##                       13                       14 
##                     1312                     2214 
##                       15                       16 
##                     2155                     2509 
##                       17                       18 
##                     3018                     1174 
##                       19                        2 
##                     2951                     3971 
##                       20                       21 
##                     1391                      837 
##                       22                        3 
##                     1339                     3019 
##                        4                        5 
##                     2504                     2869 
##                        6                        7 
##                     2860                     2884 
##                        8                        9 
##                     2367                     2246 
##           CHR_HG107_PATCH          CHR_HG126_PATCH 
##                        4                        5 
##          CHR_HG1311_PATCH    CHR_HG1342_HG2282_PATCH 
##                        7                       36 
##          CHR_HG1362_PATCH CHR_HG142_HG150_NOVEL_TEST 
##                       18                       16 
##      CHR_HG151_NOVEL_TEST          CHR_HG1651_PATCH 
##                       11                        2
```

How many genes are on regular chromosomes?

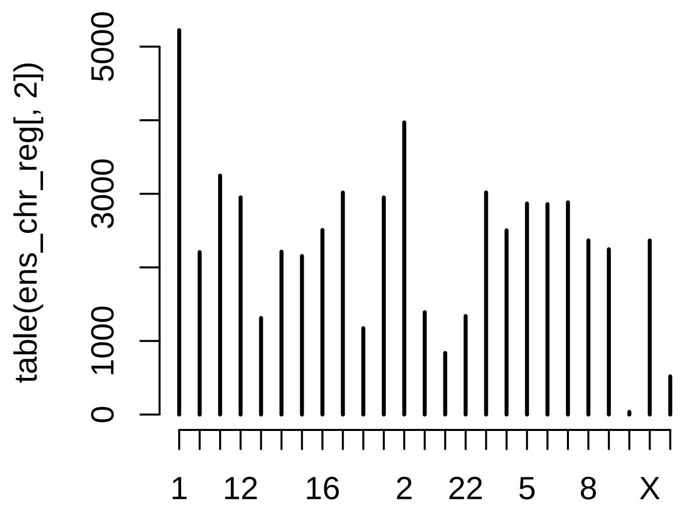
```r
sum( ens_chr[,2] %in% c(1:22, "X", "Y", "MT") )
```

```
## [1] 58174
```

```r
ens_chr_reg = ens_chr[ ens_chr[,2] %in% c(1:22, "X", "Y", "MT"), ]
table ( ens_chr_reg[,2] )
```

```
## 
##    1   10   11   12   13   14   15   16   17   18   19    2   20   21   22 
## 5224 2208 3248 2952 1312 2214 2155 2509 3018 1174 2951 3971 1391  837 1339 
##    3    4    5    6    7    8    9   MT    X    Y 
## 3019 2504 2869 2860 2884 2367 2246   37 2366  519
```

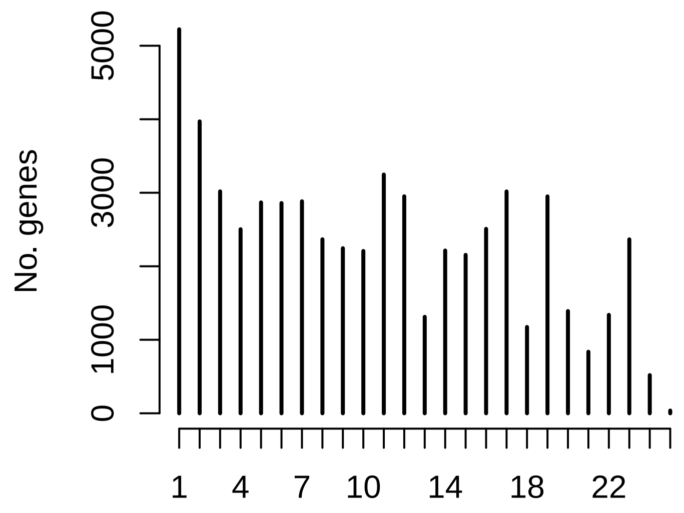
```r
plot( table ( ens_chr_reg[,2] ) )
```



```r
# but here the order is not from 1 to 22
# we can change this using the levels argument in factor()
table( factor( ens_chr_reg[,2], levels = c(1:22, "X", "Y", "MT")) )
```

```
##
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 5224 3971 3019 2504 2869 2860 2884 2367 2246 2208 3248 2952 1312 2214 2155
##   16   17   18   19   20   21   22    X    Y   MT
## 2509 3018 1174 2951 1391  837 1339 2366  519   37
```

```r
plot( table( factor( ens_chr_reg[,2], levels = c(1:22, "X", "Y", "MT")) ), ylab="No. genes")
```



Now we need to define gene lists:

- sex-related,
- not sex-related,
- X/Y-linked,
- autosomal.

```
# genes on the sex chromosomes
head(ens_chr_reg)
```

```
##   ensembl_gene_id chromosome_name
## 1 ENSG00000264452              21
## 2 ENSG00000278324               8
## 3 ENSG00000283502               2
## 4 ENSG00000241226              19
## 5 ENSG00000252604               4
## 6 ENSG00000274494               6
```

```
# note the use of %in% operator
sex_chr_genes = ens_chr_reg$ensembl_gene_id [ ens_chr_reg$chromosome_name %in% c('X', 'Y') ]
nosex_chr_genes = ens_chr_reg$ensembl_gene_id [ ens_chr_reg$chromosome_name %in% as.character(1:22) ]
length(sex_chr_genes)
```

```
## [1] 2885
```

```
length(nosex_chr_genes)
```

```
## [1] 55252
```

```
intersect(sex_chr_genes, nosex_chr_genes)
```

```
## character(0)
```

```
# genes DE with sex
de_sex_genes = names( which( nmat6_aovp[,"sex"] < 0.05 ) )  # nominal (uncorrected) p-values
node_sex_genes = names( which( nmat6_aovp[,"sex"] >= 0.05 ) )
length(de_sex_genes)
```

```
## [1] 867
```

```
length(node_sex_genes)
```

```
## [1] 14728
```

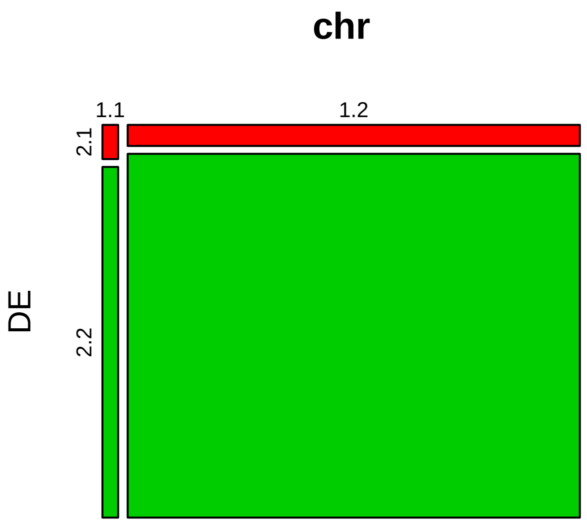
```
# check that there is no overlap
intersect(de_sex_genes, node_sex_genes)
```

```
## character(0)
```

Now we need the size of overlaps between these vectors, construct a 2X2 matrix, and run the FET. For this can write a small function based on `length` and `intersect`, and a larger function using the smaller one inside:

```
loiloi = function(x1, x2, y1, y2) {
  # function to calculate a 2X2 matrix and the corresponding 1-sided FET p-value
  # from the intersections between the vectors x1/x2 and y1/y2

  # first define the intersect function
  loi = function(x, y) {
    length( intersect( x, y ) )
  }
  overlaps = c(
    loi( x1, y1 ),
    loi( x1, y2 ),
    loi( x2, y1 ),
    loi( x2, y2 ) )
  matx = matrix ( overlaps, 2, 2 )
  fet = fisher.test( matx, alt='greater')
  return(list(matx, fet))
}
```

```
sexde_sexchr = loiloi( de_sex_genes, node_sex_genes, sex_chr_genes, nosex_chr_genes )
mosaicplot(sexde_sexchr[[1]], col=2:3, ylab = "DE", "chr")
```



And is this significant?

```
sexde_sexchr[[2]]
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  matx
## p-value = 0.001757
## alternative hypothesis: true odds ratio is greater than 1
## 95 percent confidence interval:
##  1.260026      Inf
## sample estimates:
## odds ratio
##   1.684189
```

So we can reject the H0: genes on sex chr do tend to be more DE with sex, than other genes (note that the test is already one-sided).

Could it be that genes on sex chr tend to be DE in general, such as also due to the species effect?

```
# the same, this time using genes DE with species
spDEGenes = rownames(nmat6)[nmat6_aovq[,1] < 0.05]    # as defined earlier
no_spDEGenes = setdiff(rownames(nmat6), spDEGenes)
loiloi( spDEGenes, no_spDEGenes, sex_chr_genes,nosex_chr_genes )[[2]]
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  matx
## p-value = 0.8524
## alternative hypothesis: true odds ratio is greater than 1
## 95 percent confidence interval:
##  0.7235082       Inf
## sample estimates:
## odds ratio
##   0.888982
```

No, apparently not.

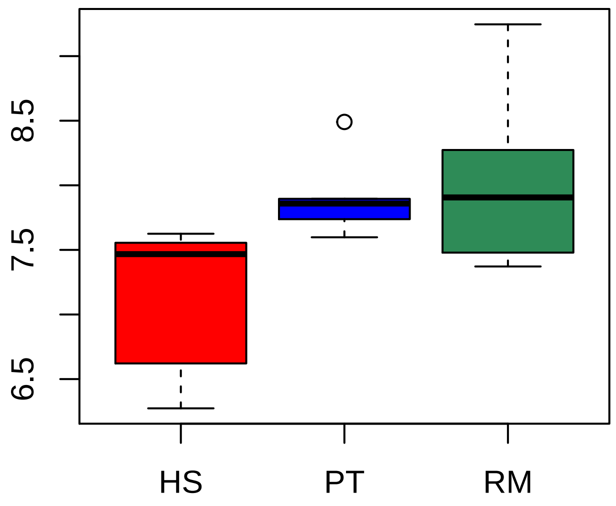
Classification and prediction - supervised clustering

Until now we only learned about **unsupervised** clustering, where we provided no information with respect to the groups. k-means and hierarchical clustering are methods to summarise the data and create groups without defining the groups in advance (note that there are many clustering methods we did not cover, such as fuzzy clustering).

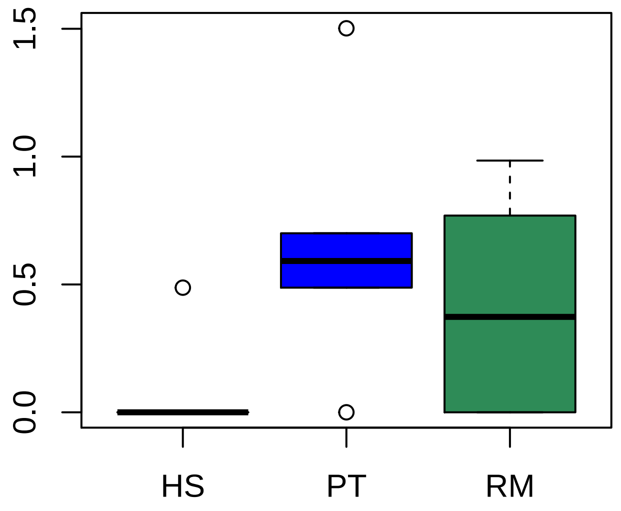
Instead methods also exist where we define groups a priori, and describe **predictor functions** that can be used to classify novel cases. Let's take the primate liver dataset - can we classify species accurately?

But this may be difficult using data from just one species:

```
colx = c("red", "blue", "seagreen")[as.numeric(species2)]
boxplot(nmat6[1, ] ~ species2, col=unique(colx))
```



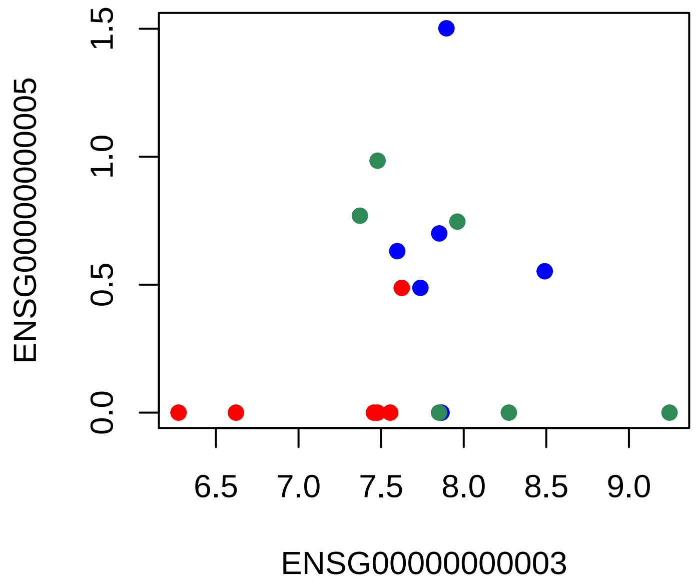
```
boxplot(nmat6[2, ] ~ species2, col=unique(colx))
```



scatterplot3d

As you see, one gene's expression levels don't suffice. But using 2 or 3 genes together might be more helpful:

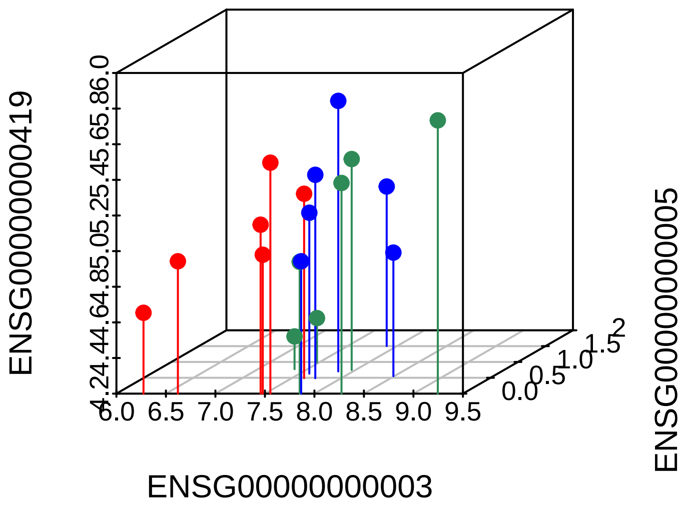
```
plot(t(nmat6[1:2, ]), col=colx, pch=19)
```



```
library(scatterplot3d)
```

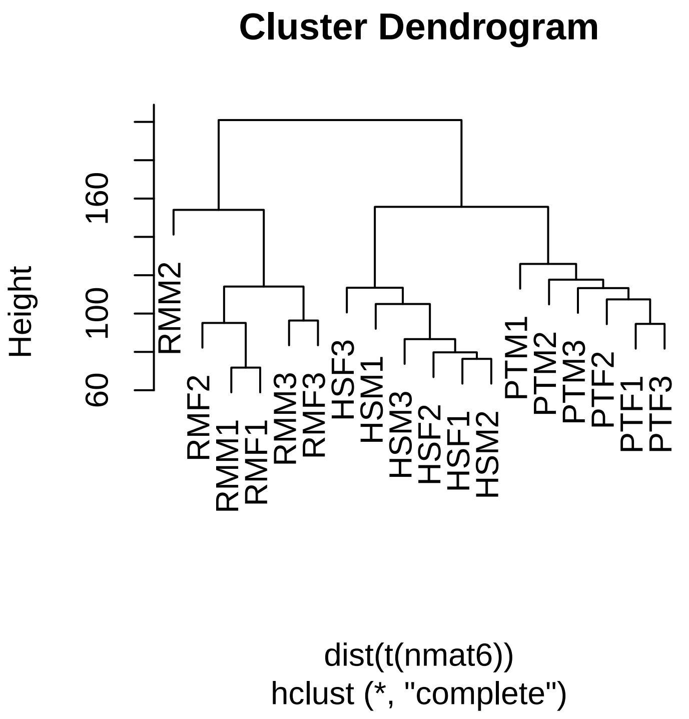
```
## Warning: package 'scatterplot3d' was built under R version 3.3.2
```

```
scatterplot3d( t(nmat6[1:3, ]), pch = 19, type="h", color = colx)
```



Obviously, with more data, we can separate species readily. This we can show using hieararchical clustering:

```r
plot( hclust ( dist( t( nmat6 ) ) ) )
```



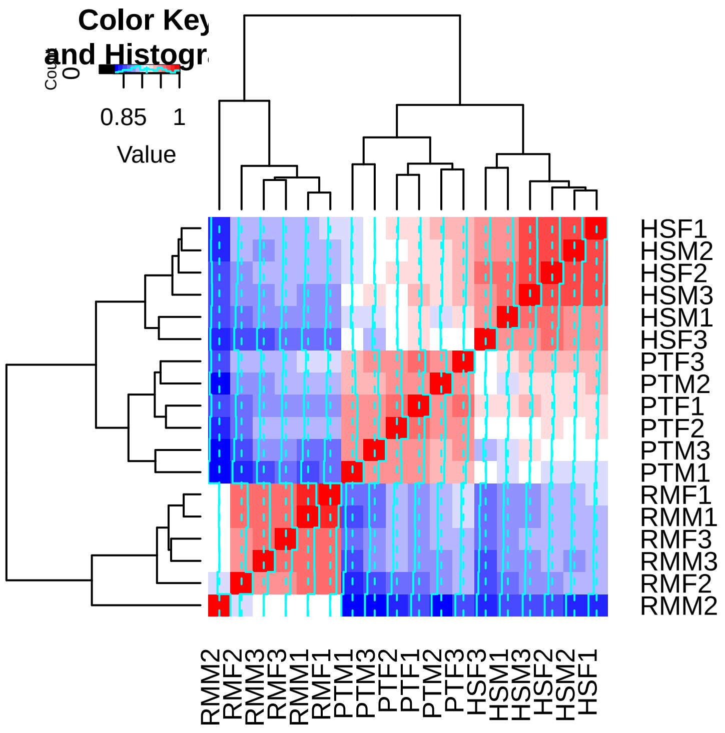
Another representation yet could be a correlation plot, this time representing grouping among individuals depending on the correlation level:

```r
library("gplots")
```

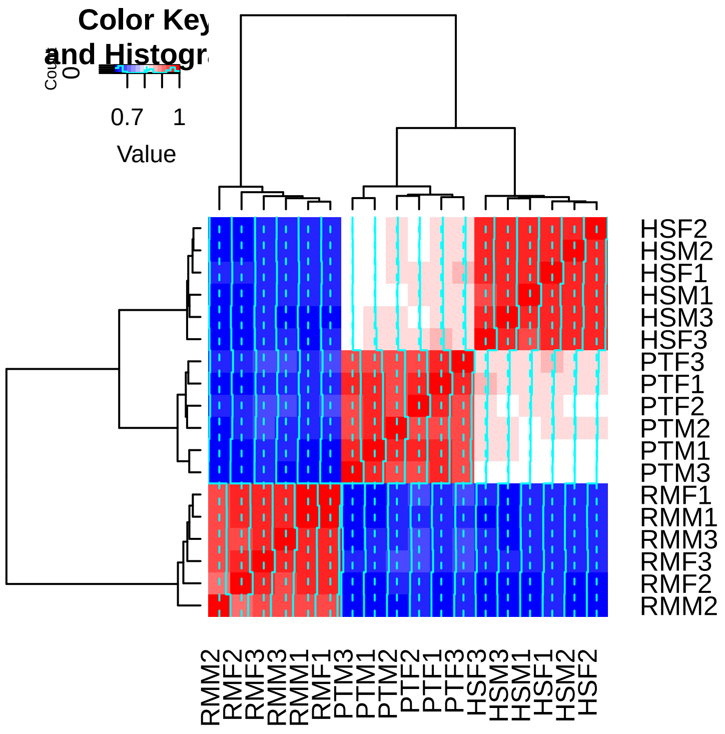
```
##
## Attaching package: 'gplots'

## The following object is masked from 'package:IRanges':
##
##     space

## The following object is masked from 'package:S4Vectors':
##
```

```
##     space

## The following object is masked from 'package:stats':
##
##     lowess
```

```
# with all genes
heatmap.2( cor(nmat6), col=bluered )
```



```
# with DE genes - more distinct, because we selected for such genes!
heatmap.2( cor(nmat6[spDEGenes,]), col=bluered )
```



If we had the expression profile of an individual of unknown species origin, we could correlate its profile with each of the 3 species. The individual should then belong to the species with the highest correlation levels.

Support vector machines (the `e1071` package)

A more elegant and powerful way to do the same is using support vector machines:

http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

This is a supervised learning algorithm, similar to logistic regression, discriminant analysis, neural networks, were you can predict the identity of a sample given enough predictor data. We can use the `e1071` package for this: https://www.rdocumentation.org/packages/e1071/versions/1.6-8

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.3.2
```

```
svm_model = svm(t(nmat6[1:3, ]), species2)
svm_model
```

```
##
## Call:
## svm.default(x = t(nmat6[1:3, ]), y = species2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.3333333
##
## Number of Support Vectors:  17
```

```r
pred = predict(svm_model, t(nmat6[1:3, ]))
pred
```

```
## HSM1 PTF1 RMM1 HSF1 PTM1 RMF1 RMF2 HSM2 PTF2 RMM2 HSF2 PTM2 RMM3 RMF3 HSM3
##   HS   PT   RM   HS   HS   HS   PT   PT   PT   RM   HS   PT   RM   RM   HS
## PTF3 PTM3 HSF3
##   PT   PT   HS
## Levels: HS PT RM
```

```r
# for only individual 1
predict(svm_model, t(nmat6[1:3, 1]))
```

```
##  1
## HS
## Levels: HS PT RM
```

```r
# and for all individuals
table(pred, species2)
```

```
##     species2
## pred HS PT RM
##   HS  5  1  1
##   PT  1  5  1
##   RM  0  0  4
```

Note that not all individuals are correctly predicted.

How about using the whole dataset?

```r
svm_model = svm(t(nmat6), species2)
svm_model
```

```
##
## Call:
## svm.default(x = t(nmat6), y = species2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  6.412312e-05
##
## Number of Support Vectors:  18
```

```r
pred = predict(svm_model, t(nmat6))
pred
```

```
## HSM1 PTF1 RMM1 HSF1 PTM1 RMF1 RMF2 HSM2 PTF2 RMM2 HSF2 PTM2 RMM3 RMF3 HSM3
##   HS   PT   RM   HS   PT   RM   RM   HS   PT   RM   HS   PT   RM   RM   HS
## PTF3 PTM3 HSF3
##   PT   PT   HS
## Levels: HS PT RM
```

```r
table(pred, species2)
```

```
##     species2
## pred HS PT RM
##   HS  6  0  0
##   PT  0  6  0
##   RM  0  0  6
```

Now we have much more power and all individuals are predicted correctly.

Leaving out

To fully demonstrate the predictive power of the `SVM`, we could also run this analyses with only part of the data (e.g. leaving out an individual), and then check if those individual(s) (which the model is naive about) can still be correctly identified.

```
species2[3]
```

```
## [1] RM
## Levels: HS PT RM
```

```
# SVM without a macaque
svm1 = svm(x = t(nmat6[1:3, -3]), species2[-3] )
# predict the species identity
predict( svm1, t(nmat6[1:3, 3]) )
```

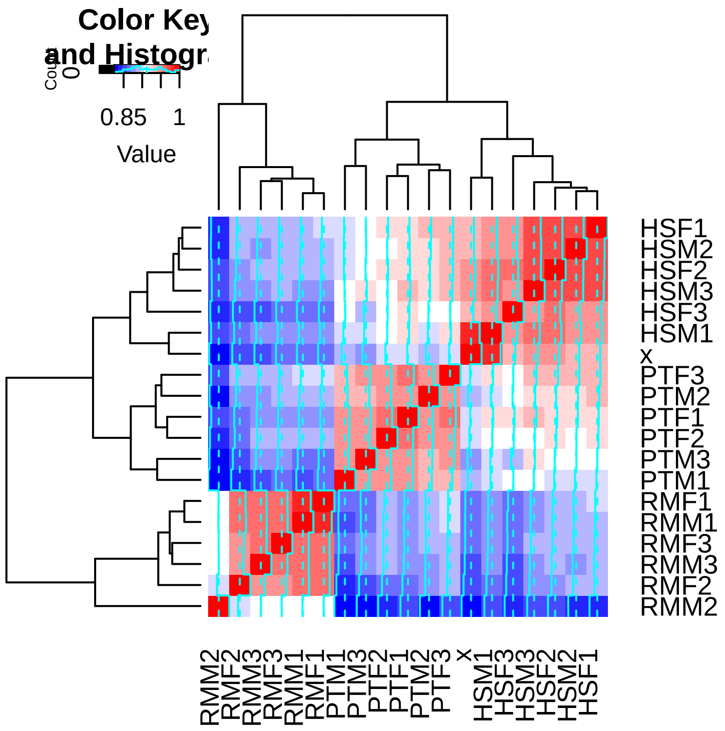
```
##  1
## RM
## Levels: HS PT RM
```

This also works nicely.

A hypothetical individual

How about a hypothetical individual with some noise?

```
# use the 1st human's profile and add some noise from the standard normal distribution
x = nmat6[,1] + rnorm(nrow(nmat6), mean=0, sd=0.5)
heatmap.2( cor(cbind(nmat6, x)), col=bluered )
```

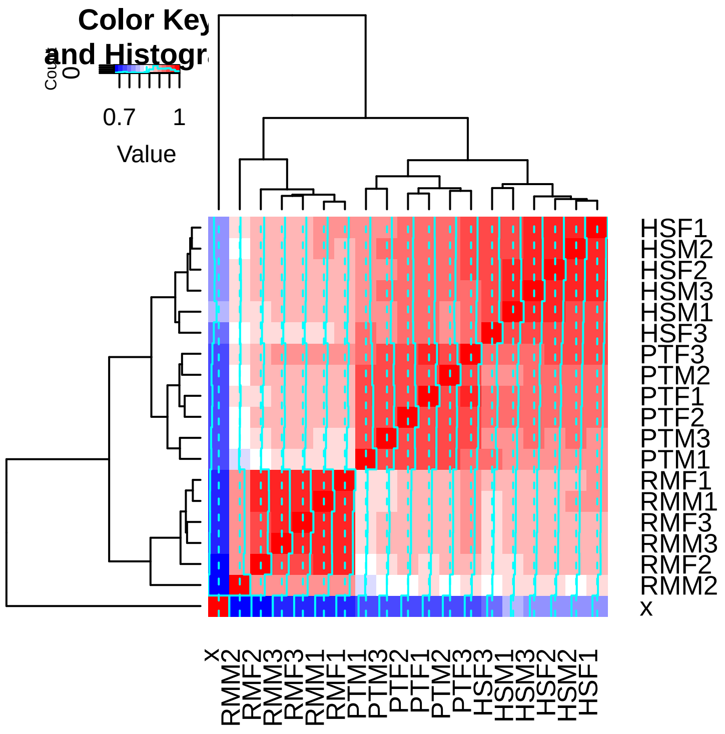


```
predict(svm_model, t(x))
```

```
##  1
## HS
## Levels: HS PT RM
```

This works. How about more noise?

```
x = nmat6[,1] + rnorm(nrow(nmat6), mean=0, sd=2)
heatmap.2( cor(cbind(nmat6, x)), col=bluered )
```



```
predict(svm_model, t(x))
```

```
##  1
## RM
## Levels: HS PT RM
```

This doesn't work: altough this made-up individual has relatively the highest correlation to other humans, it is predicted to be macaque, most likely because macaques are the outlier group, and the made-up profile is also an outlier.

Sensitivity and specificity of prediction

With only data from 3 genes, the accuracy is low. We can also talk about sensitivity and specificity, with respect to a certain outcome.

Sensitivity is the proportion of predicted positives out of all true positives (similar to **power**). E.g. how many real humans can we detect out of the 6?

`Sensitivity = true positives/(true positives + false negatives)`

`Sensitivity = 1 - false negative rate`

So this is 0.8333333 for humans. We can also calculate this for all species:

```
sum( diag(table(pred, species2))) / length(species2)
```

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

Specificity is the proportion of true negatives out of all predicted negatives.

`Specificity = true negatives/(true negatives + false positives)`

`Specificity = 1 - false positive rate`

E.g. what is the probability that you detect someone is *not* a macaque?

```
12/(12 + 2)
```

```
## [1] 0.8571429
```

Analysing the `iris` dataset

This part is courtesy of Dr. Tuba Bucak and other sources: https://cran.r-project.org/web/packages/dendextend/vignettes/Cluster_Analysis.html

The famous (Fisher???s or Anderson???s) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

Let us study the dataset using different classification methods:

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
dim(iris)
```

```
## [1] 150   5
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
##                 
```

```
str(iris)
```

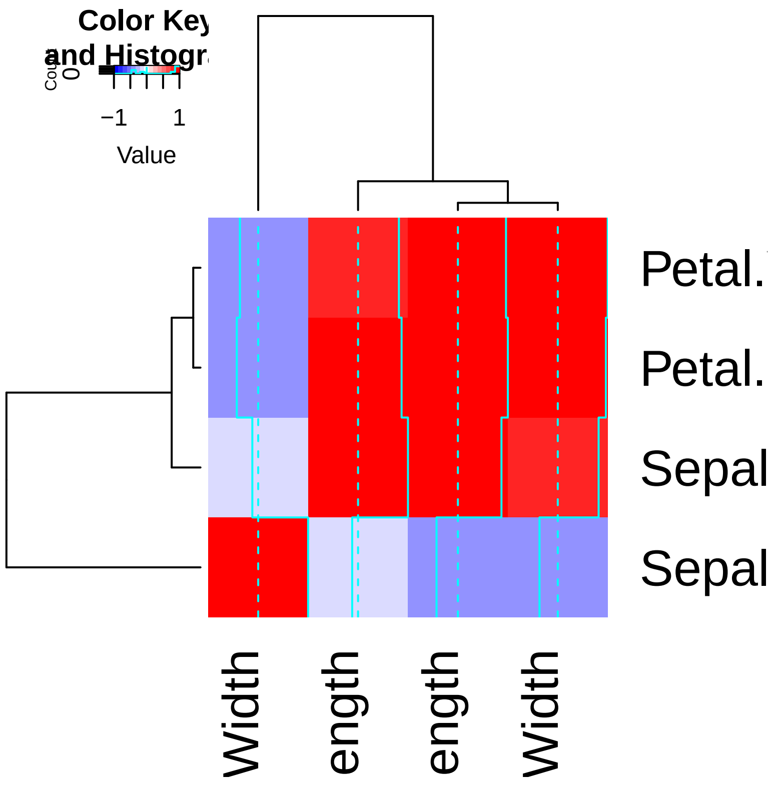
```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

One way to study this dataset is to study correlations among the numeric variables. Using `cor` on heatmap is one approach:

```r
cor(iris[,1:4])
```

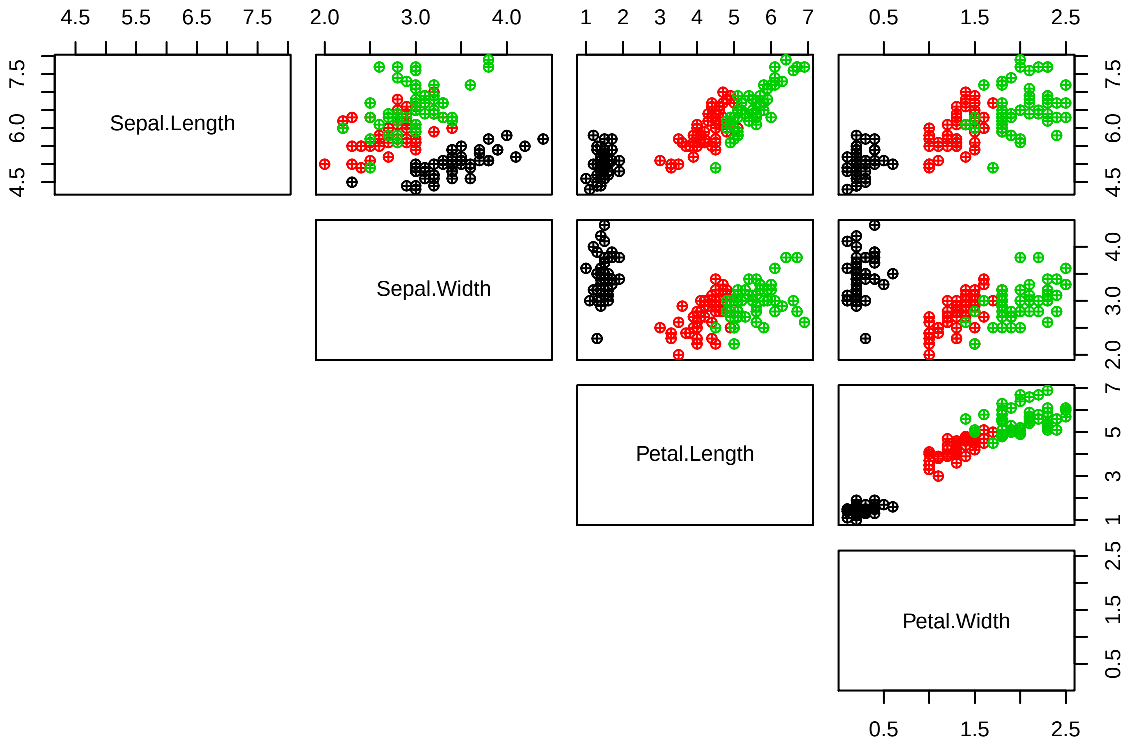
```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

```r
heatmap.2(cor(iris[,1:4]), col = "bluered")
```



But using the `pairs` function is even better:

```r
pairs(iris[,1:4], col = as.numeric(iris$Species),
      lower.panel = NULL,
      cex.labels=1, pch=10, cex = 1.0)
```



```r
levels(iris$Species)
```

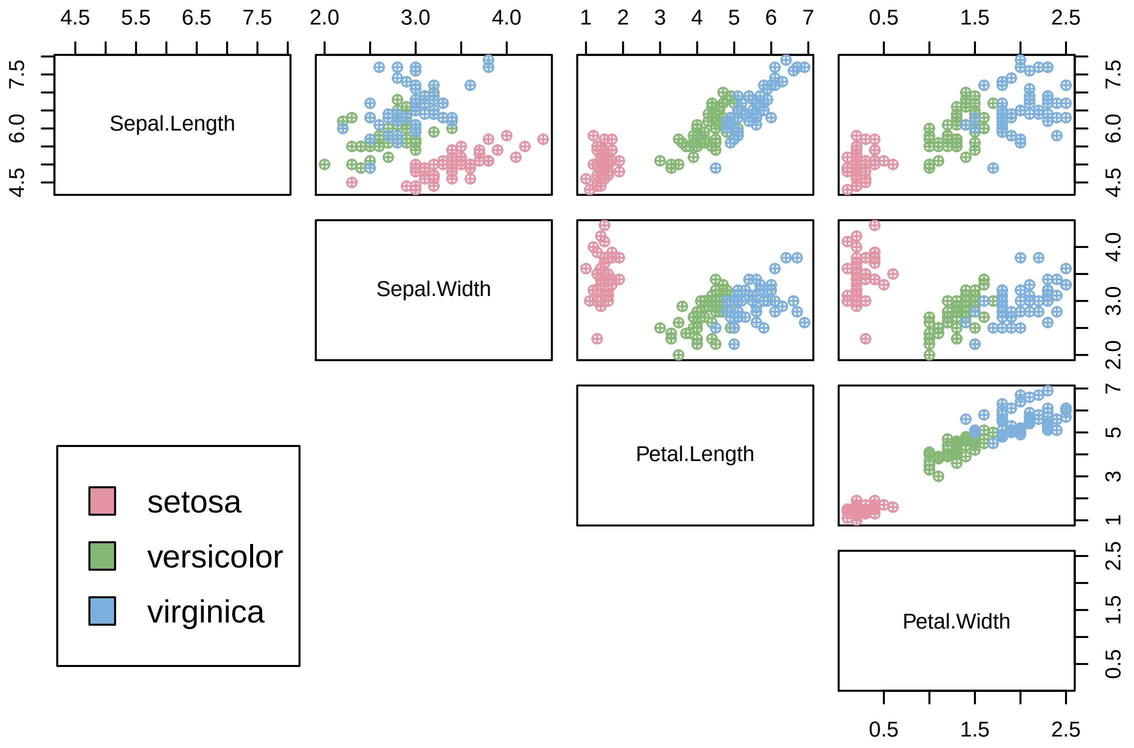
```
## [1] "setosa"     "versicolor" "virginica"
```

We can try a nicer version with colors:

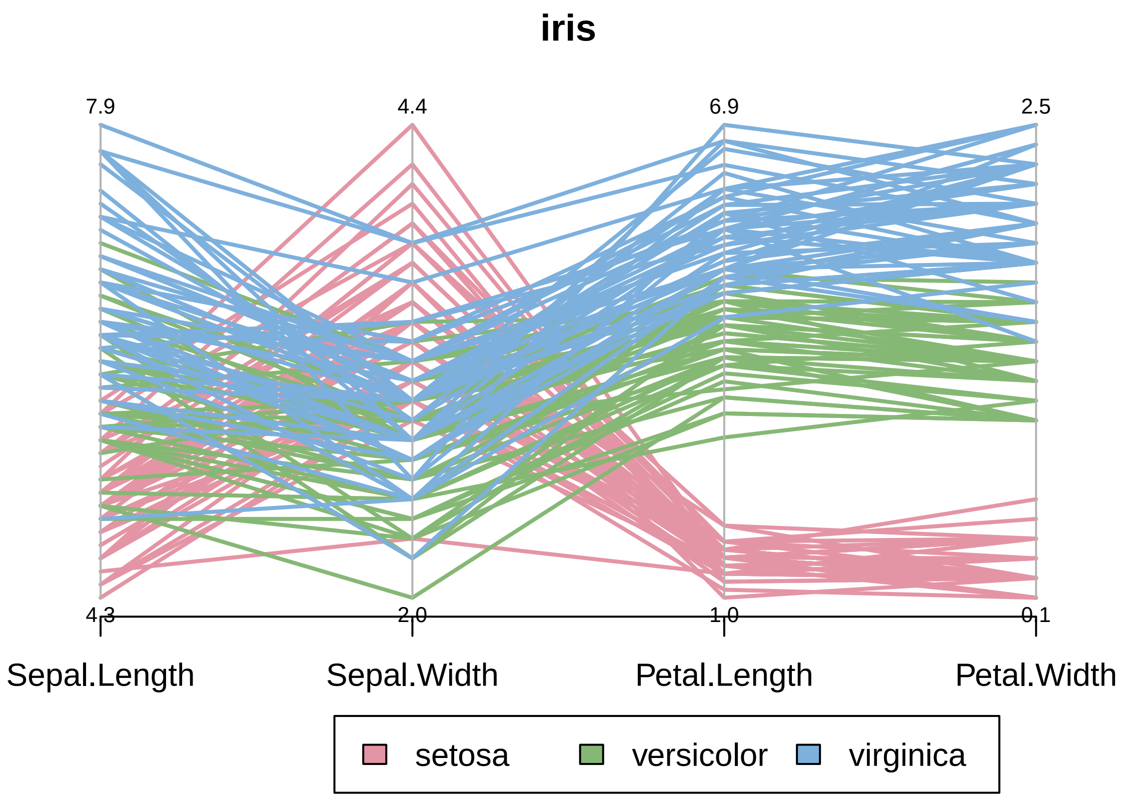
```r
library(colorspace)
```

```
## Warning: package 'colorspace' was built under R version 3.3.2
```

```r
colx = rainbow_hcl(3)[as.numeric(iris$Species)]
pairs(iris[,1:4], col = colx,
      lower.panel = NULL,
      cex.labels=1, pch=10, cex = 1.0)
par(xpd = TRUE)   # add the legend to the existing plot
legend(x = 0.05, y = 0.4, cex = 1,
       legend = as.character(levels(iris$Species)),
       fill = unique(colx))
```



```
MASS::parcoord(iris[,1:4], col = colx, var.label = TRUE, lwd = 2)

title("iris")
par(xpd = TRUE)
legend(x = 1.75, y = -.25, cex = 1,
   legend = as.character(levels(iris$Species)),
    fill = unique(colx), horiz = TRUE)
```

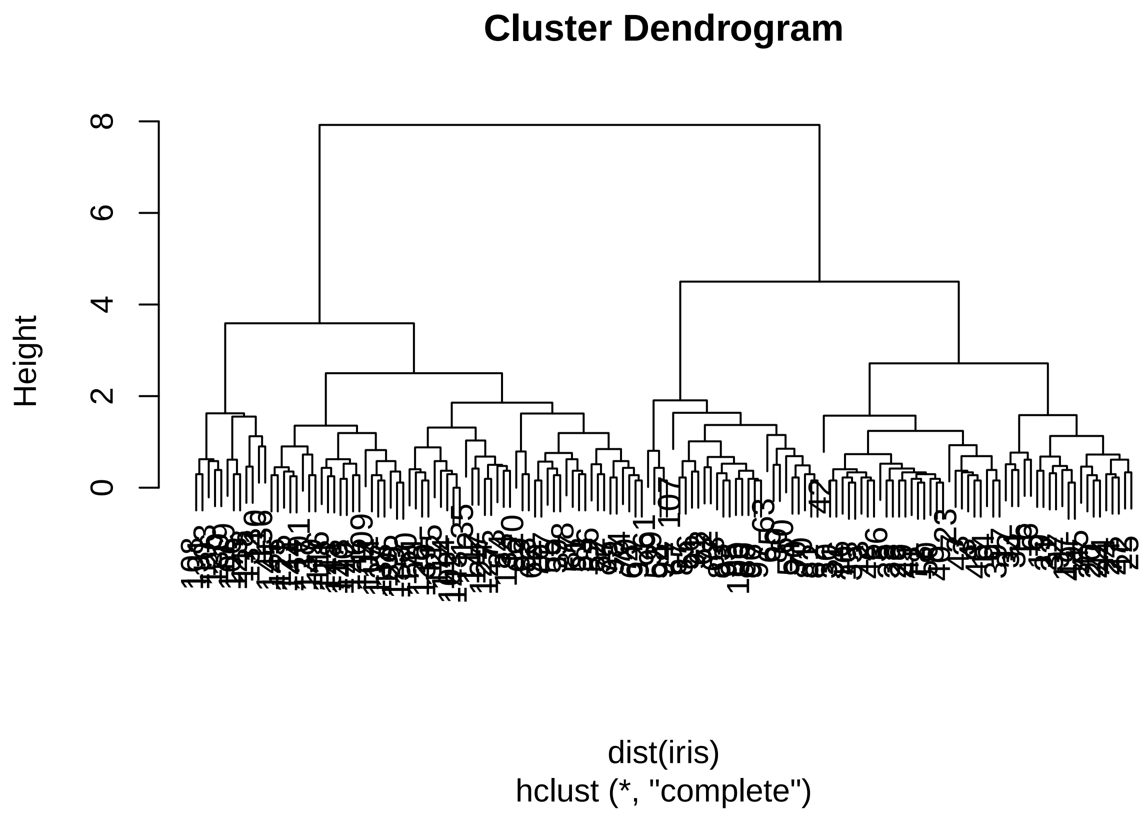


Unsupervised clustering (hierarchical)

```
hc_iris = hclust( dist(iris) )
```

```
## Warning in dist(iris): NAs introduced by coercion
```

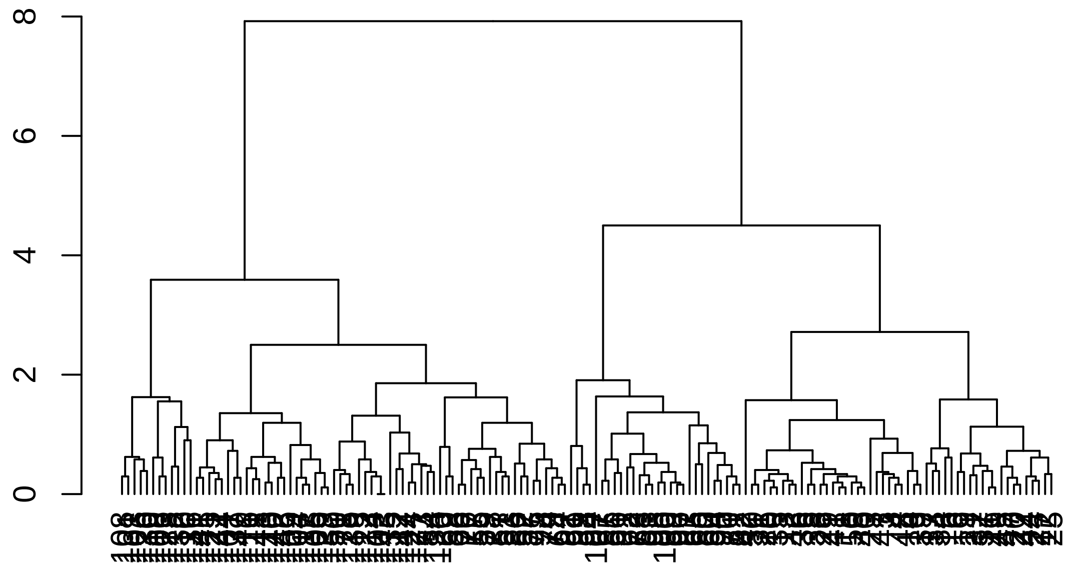
```
plot( hc_iris )
```



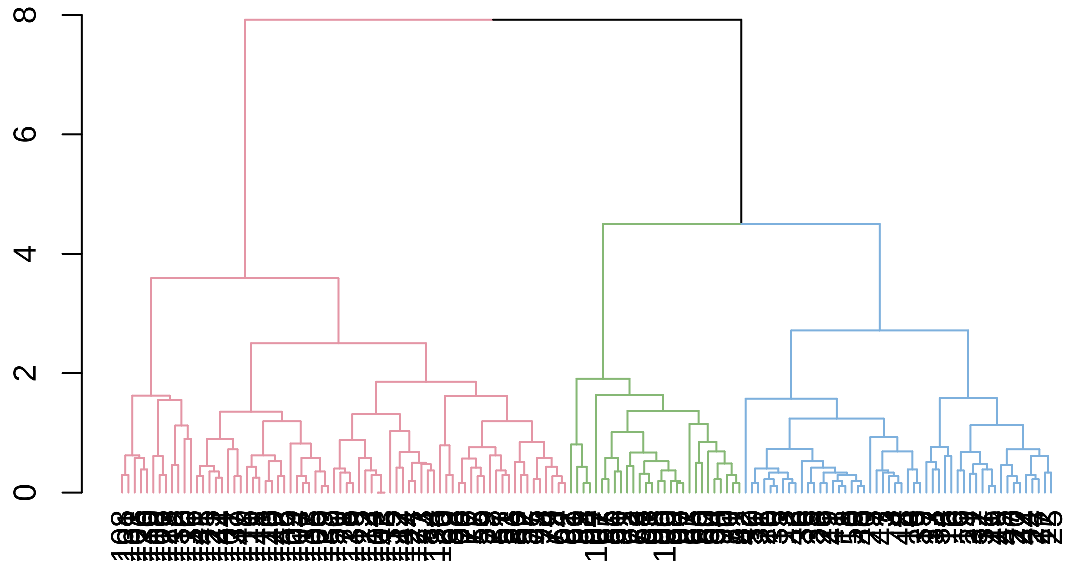
```
dend = as.dendrogram(hc_iris)
plot(dend)
library(dendextend)
```

```
## Warning: package 'dendextend' was built under R version 3.3.2

##
## ---------------------
## Welcome to dendextend version 1.5.2
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ---------------------

##
## Attaching package: 'dendextend'

## The following object is masked from 'package:stats':
##
##     cutree
```

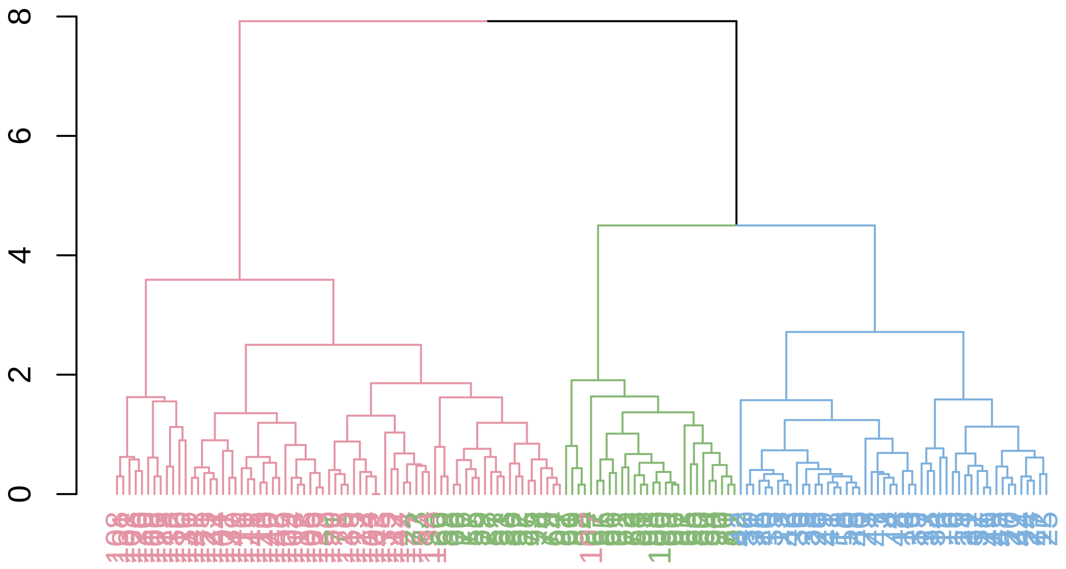
```
dend = rotate(dend, 1:150)
plot(dend)
```



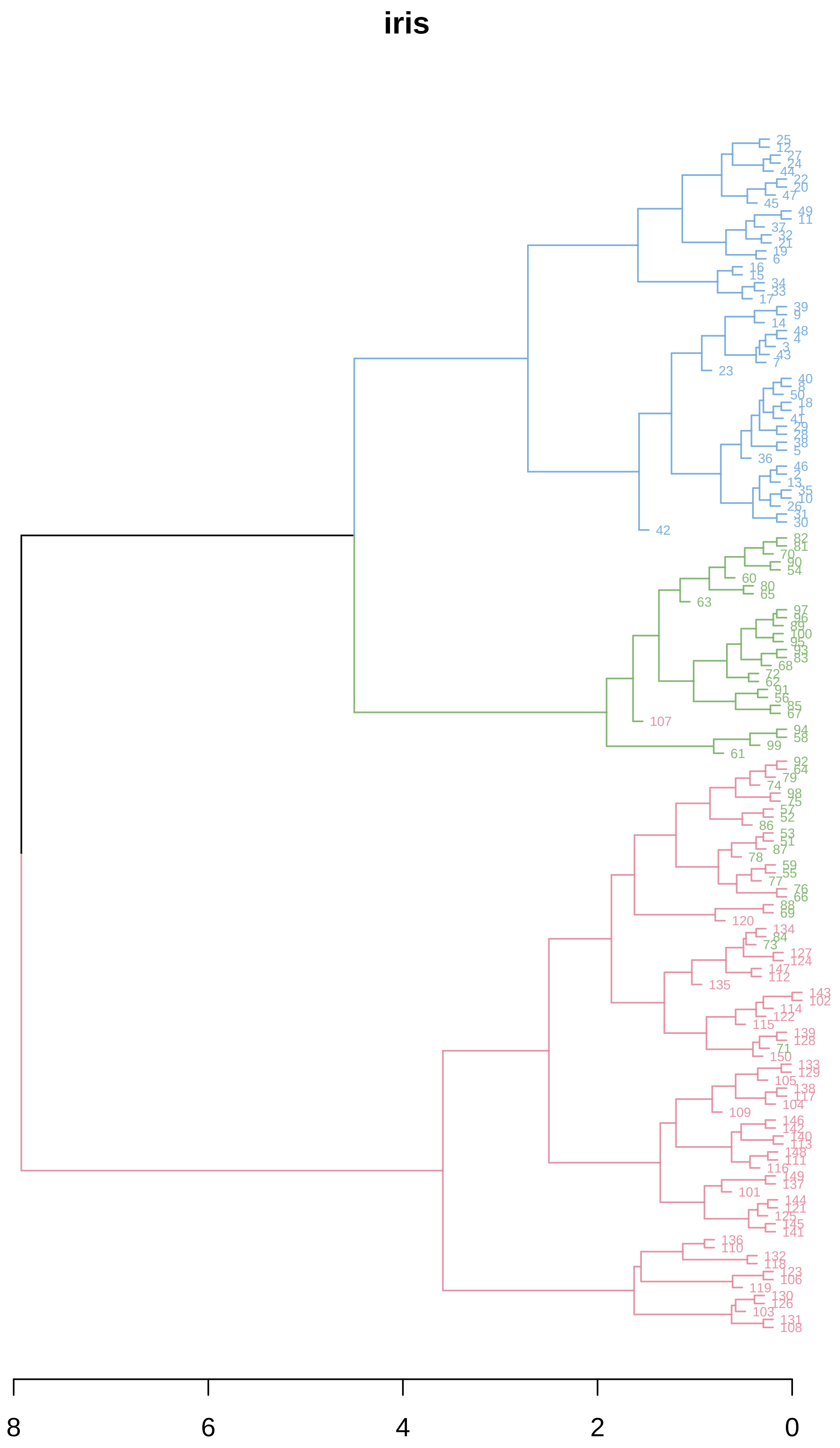
```r
dend = color_branches(dend, k=3, col=unique(colx))
plot(dend)
```



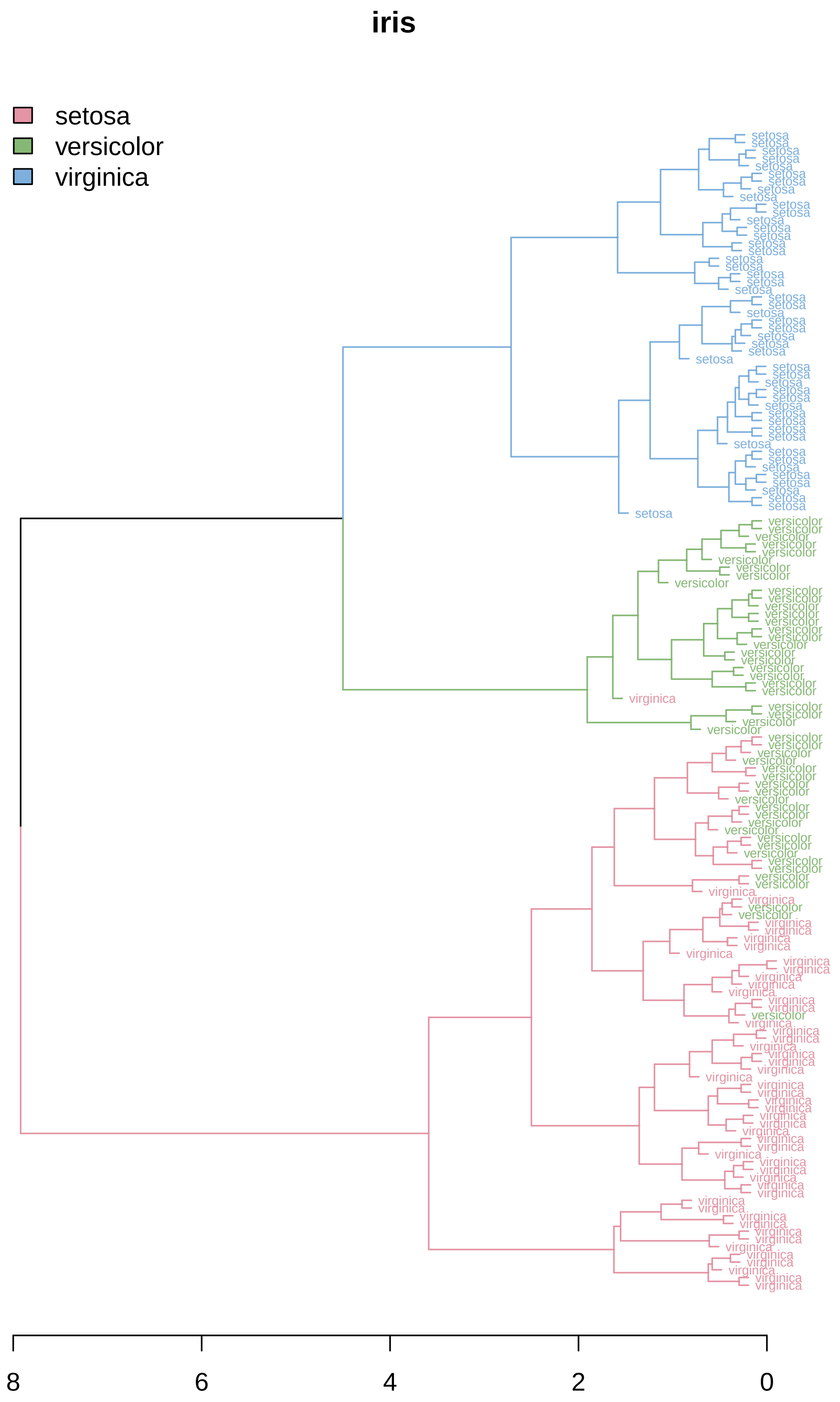
```r
orderx = order.dendrogram(dend)
# change color labels
labels_colors(dend) = unique(colx) [
  sort_levels_values( as.numeric(iris$Species) [ orderx ] )]
plot(dend)
```



```
dend = hang.dendrogram(dend,hang_height=0.1)
dend = set(dend, "labels_cex", 0.5)
plot(dend,
     main = "iris",
     horiz =  TRUE,  nodePar = list(cex = .007))
```



```r
labels(dend) = as.character(iris$Species)[order.dendrogram(dend)]
plot(dend,
     main = "iris",
     horiz =  TRUE,  nodePar = list(cex = .007))

legend("topleft", legend = unique(iris$Species), fill = rainbow_hcl(3), bty="n")
```


iris
setosa
versicolor
virginica
8
6
4
2
0

Supervised clustering (SVM)

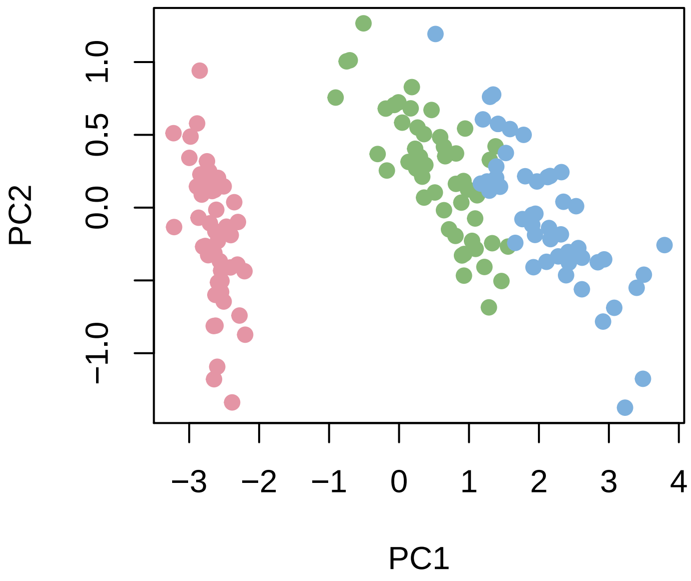
```r
svm_model = svm(iris[,1:4], iris$Species)
pred = predict(svm_model, iris[,1:4])
table(pred, iris$Species)
```

```
##
## pred         setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         2
##   virginica       0          2        48
```

`biplot` and PCA factors

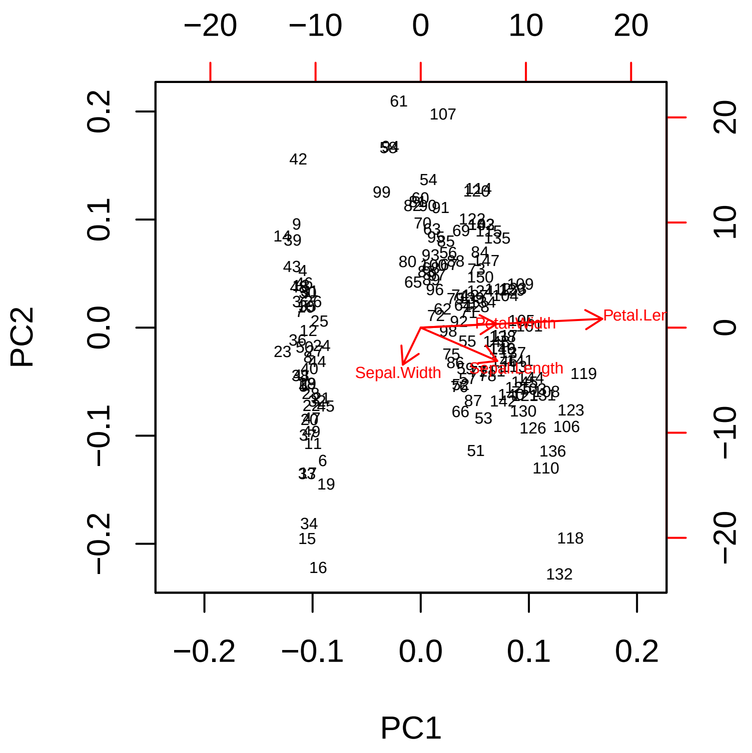
Now let's repeat our PCA analysis to check if the species separate, and which variable allows most separation:

```r
pc = prcomp(as.matrix(iris[,1:4]))
plot(pc$x, col=colx, pch=19)
```



When we have limited number of variables, we can use `biplot`, which aims to represent both the observations and variables of a matrix of multivariate data on the same plot:

```r
biplot(pc, cex=0.5)
```



We can see that petal length has strong influence on the data (explains large part of the variance). Petal width is correlated with petal length but explains less variance. And sepal width is uncorrelated.