Meta-data in Bioconductor

Educational Materials

Overview

- Closing the gap between knowledge of sequence and knowledge of function requires aggressive, integrative use of biological research databases of many different types.
- In this lecture we focus attention on resources that can help to make use of meta-data in different analyses.
- We make use of the following packages: annotate, GO, KEGG, hgu133a, GOstats, and cMAP.

Per chip annotation

- An early design decision was that we should provide meta-data on a per chip-type basis.

```
> library("hgu133a")
> ls("package:hgu133a")
 [1] "hgu133a"              "hgu133aACCNUM"
 [3] "hgu133aCHR"           "hgu133aCHRLENGTHS"
 [5] "hgu133aCHRLOC"        "hgu133aENZYME"
 [7] "hgu133aENZYME2PROBE" "hgu133aGENENAME"
 [9] "hgu133aGO"            "hgu133aGO2ALLPROBES"
[11] "hgu133aGO2PROBE"      "hgu133aGRIF"
[13] "hgu133aLOCUSID"       "hgu133aMAP"
[15] "hgu133aOMIM"          "hgu133aORGANISM"
[17] "hgu133aPATH"          "hgu133aPATH2PROBE"
[19] "hgu133aPMID"          "hgu133aPMID2PROBE"
[21] "hgu133aREFSEQ"        "hgu133aSUMFUNC"
[23] "hgu133aSYMBOL"        "hgu133aUNIGENE"
```

A brief description

- These packages contain R `environments`, which are used as hash tables.
- For each package there is quality control information available, use `hgu133a()`; which report how many of each of the different mappings were found.
- You can access the data directly using any of the standard subsetting or extraction tools for `environments`; `get`, `mget`, `$` and `[[`.

```
> hgu133aSYMBOL$"201473_at"
[1] "JUNB"
> hgu133aLOCUSID[["201476_s_at"]]
[1] 6240
> get("201475_x_at", hgu133aOMIM)
[1] "156560"
```

MetaData

LocusLink is a catalog of genetic loci that connects curated sequence information to official nomenclatur

UniGene defines sequence clusters. UniGene focuses on protein-coding genes of the nuclear genome (excluding rRNA and mitochondrial sequences).

RefSeq is a non-redundant dataset of transcripts and proteins of known genes for a variety of species, including human, mouse and rat.

Enzyme Commission (EC) numbers are assigned to different enzymes and linked to genes through their association with LocusLink identifiers.

Gene Ontology (GO) is a structured vocabulary of terms describing gene products according to relevant molecular function, biological process, or cellular component.

Meta-data

PubMed is a service of the U.S. National Library of Medicine. PubMed provides a very rich resource of data and tools for working with papers published in journals that are related to medicine and health. The data source, while large, is not comprehensive and not all papers have been abstracted.

LITDB The Protein Research Foundation curates LITDB, which covers all articles dealing with peptides from scientific journals accessible in Japan.

OMIM Online Mendelian Inheritance in Man is a catalog of human genes and genetic disorders.

NetAffx The NetAffx Analysis Center provides tools that correlate experimental data assayed using the Affymetrix GeneChip technology.

Meta-data

KEGG Kyoto Encyclopedia of Genes and Genomes; a collection of data resources including a rich collection of pathway data.

cMAP Pathway data from both KEGG and BioCarta, in a computable form.

Chromosomal Location Genes are identified with chromosomes, and where appropriate with strand.

Data Archives The NCBI coordinates the Gene Expression Omnibus (GEO); TIGR provides the Resourcerer database, and the EBI supports ArrayExpress.

Working with Meta-data

- Suppose for example, we are interested in the gene BAD.

```
> gsyms <- unlist(as.list(hgu95av2SYMBOL))
> whBAD <- grep("^BAD$", gsyms)
> gsyms[whBAD]
1861_at
  "BAD"
```

BAD Pathways

- Now find the pathways that BAD is associated with.

```
> BADpath <- hgu95av2PATH$"1861_at"
> mget(BADpath, KEGGPATHID2NAME)
$"04210"
[1] "Apoptosis"

$"05030"
[1] "Amyotrophic lateral sclerosis (ALS)"
> allProbes <- mget(BADpath, hgu95av2PATH2PROBE)
> sapply(allProbes, length)
04210 05030
  149    28
```

Annotating a Genome

- Bioconductor also provides some comprehensive annotations for whole genomes (e.g. *S. cerevisae*).
- These packages are like the chip annotation packages, except a different set of primary keys is used (e.g. for yeast we use the systematic names such as `YBL088C`)

```
> library("YEAST")
> ls("package:YEAST")
 [1] "YEAST"              "YEASTALIAS"
 [3] "YEASTCHR"           "YEASTCHRLENGTHS"
 [5] "YEASTCHRLOC"        "YEASTDESCRIPTION"
 [7] "YEASTENZYME"        "YEASTENZYME2PROBE"
 [9] "YEASTGENENAME"      "YEASTGO"
[11] "YEASTGO2ALLPROBES" "YEASTGO2PROBE"
[13] "YEASTORGANISM"      "YEASTPATH"
[15] "YEASTPATH2PROBE"    "YEASTPMID"
[17] "YEASTPMID2PROBE"
```

The annotate package

- Functions for harvesting of curated persistent data sources
- functions for simple HTTP queries to web service providers
- interface code that provides common calling sequences for the assay based meta-data packages such as `getGI` and `getSEQ` perform web queries to NCBI to extract the GI or nucleotide sequence corresponding to a GenBank accession number.

```
> ggi <- getGI("M22490")
> ggi
[1] "179503"
> gsq <- getSEQ("M22490")
> substring(gsq, 1, 40)
[1] "GGCAGAGGAGGAGGGAGGGAGGGAAGGAGCGCGGAGCCCG"
```

The annotate package

- other interface functions include `getGO`, `getSYMBOL`, `getPMID`, and `getLL`
- functions whose names start with `pm` work with lists of PubMed identifiers for journal articles.

```
> hoxa9 <- "37809_at"
> absts <- pm.getabst(hoxa9, "hgu95av2")
> substring(abstText(absts[[1]][[1]]), 1, 60)
[1] "In primary cells from acute leukemia patients, expression of"
```

Working with GO

- An ontology is a structured vocabulary that characterizes some conceptual domain.
- The Gene Ontology (GO) Consortium defines three ontologies characterizing aspects of knowledge about genes and gene products.
- These ontologies are
  - molecular function (MF),
  - biological process (BP)
  - cellular component (CC).
- for explicit descriptions of these categories you should consult the GO web page.

GO

- *molecular function* of a gene product is what it does at the biochemical level. This describes what the gene product can do, but without reference to where or when this activity actually occurs. Examples of functional terms include "enzyme," "transporter," or "ligand."
- *biological process* is a biological objective to which the gene product contributes. There is often a temporal aspect to a biological process. Biological processes usually involve the transformation of a physical thing. The terms "DNA replication" or "signal transduction" describe general biological processes.
- *cellular component* is a part of a cell that is a component of some larger object or structure. Examples of cellular components include "chromosome", "nucleus" and "ribosome".

GO Characteristics

	Number of Terms
BP	8578
CC	1335
MF	6891

Table 1: Number of GO terms per ontology.

Working with GO For precision and conciseness, all indexing of GO resources employs the 7 digit tags with prefix `GO:`. Three very basic tasks that are commonly performed in conjunction with GO are

- navigating the hierarchy, determining parents and children of selected terms, and deriving subgraphs of the overall DAG constituting GO;
- resolving the mapping from GO tag to natural language characterizations of function, location, or process;
- resolving the mapping between GO tags or terms and elements of catalogs of genes or gene products.

Navigating the hierarchy

- Finding parents and children of different terms is handled by using the `PARENT` and `CHILDREN` mappings.
- To find the children of `"GO:0008094"` we use:

```
> get("GO:0008094", GOMFCHILDREN)
[1] "GO:0004003" "GO:0008722" "GO:0015616" "GO:0043142"
```

- We use the term *offspring* to refer to all descendants (children, grandchildren, and so on) of a node.
- Similarly we use the term *ancestor* to refer to the parents, grandparents, and so on, of a node.

```
> get("GO:0008094", GOMFOFFSPRING)
[1] "GO:0004003" "GO:0008722" "GO:0015616" "GO:0043142"
[5] "GO:0017116" "GO:0043140" "GO:0043141"
```

Searching for terms

- All GO terms are provided in the `GOTERM` environment. It is relatively easy to search for a term with the word `chromosome` in it using `eapply` and `grep` or `agrep`.

```
> hasChr <- eapply(GOTERM, function(x) x[grep("chromosome",
+     Term(x))])
> lens <- sapply(hasChr, length)
> hasChr <- hasChr[lens > 0]
> length(hasChr)
[1] 64
```

Searching for terms

- We can write a function:

```
> GOTerm2Tag <- function(term) {
+     GTL <- eapply(GOTERM, function(x) {
+         grep(term, x@Term, value = TRUE)
+     })
+     Gl <- sapply(GTL, length)
+     names(GTL[Gl > 0])
+ }
```

- and then apply to find all terms with a specific phrase, for example, “transcription factor binding”:

```
> hasTFA <- GOTerm2Tag("transcription factor binding")
> hasTFA
[1] "GO:0003719" "GO:0008134"
```

Evidence Codes

- The mapping of genes to GO terms is carried out separately by GOA
- Four environments in the GO package address the association between LocusLink sequence entries and GO terms: `GOLOCUSID`, `GOALLOCUSID`, `GOLOCUSID2GO`, and `GOLOCUSID2ALLGO`

Term	Definition
IMP	inferred from mutant phenotype
ISS	inferred from sequence similarity
IEA	inferred from electronic annotation
TAS	traceable author statement

Table 2: Some GO Evidence Codes.

GO Evidence Codes

- find the GO identifier for "transcription factor binding" and use that to get all LocusLink identifiers that have that annotation.

```
> gg1 <- get(GOTerm2Tag("^transcription factor binding$"),
+     GOLOCUSID)
> table(names(gg1))

IDA IMP IPI ISS NAS TAS
  9   1   8  16   4  28
```

LocusLink ID

- consider the gene with LocusLink ID 7355, SLC35A2

```
> lll <- GOLOCUSID2GO[["7355"]]
> length(lll)
[1] 9
> sapply(lll, function(x) x$Ontology)
GO:0000139 GO:0015785 GO:0005459 GO:0008643 GO:0006012
      "CC"       "BP"       "MF"       "BP"       "BP"
GO:0016021 GO:0015780 GO:0005338 GO:0005351
      "CC"       "BP"       "MF"       "MF"
```

- there are 9 different GO terms.
- We get those from the `BP` ontology by using `getOntology`.

```
> getOntology(lll, "BP")
[1] "GO:0015785" "GO:0008643" "GO:0006012" "GO:0015780"
```

Evidence Codes

- then get the evidence codes using `getEvidence`
- we can drop codes using `dropEcode`

```
> getEvidence(ll1)
GO:0000139 GO:0015785 GO:0005459 GO:0008643 GO:0006012
     "IEA"      "TAS"      "TAS"      "IEA"      "TAS"
GO:0016021 GO:0015780 GO:0005338 GO:0005351
     "IEA"      "IEA"      "IEA"      "IEA"
> zz <- dropECode(ll1, code = "IEA")
> getEvidence(zz)
GO:0015785 GO:0005459 GO:0006012
     "TAS"      "TAS"      "TAS"
```

GO graphs

- For any set of selected genes, and any of the three GO ontologies the *induced GO graph* is the set of GO terms that the genes are associated with, together with all less specific terms
- the term "transcription factor activity" is in the molecular function (MF) ontology and has the GO label `GO:0003700`

```
> library("GO")
> library("GOstats")
> GOTERM$"GO:0003700"
GOID = GO:0003700
Term = transcription factor activity
Definition = Any protein required to initiate or
     regulate transcription; includes both gene
     regulatory proteins as well as the general
     transcription factors.
Ontology = MF
```

- The induced graph, based on the MF hierarchy, can be produced using the `GOGraph` function of the package GOstats

```
> tfG <- GOGraph("GO:0003700", GOMFPARENTS)
```

- we can plot the induced GO graph using Rgraphviz and the code below.

```
> plot(tfG, nodeAttrs = nattr)
```

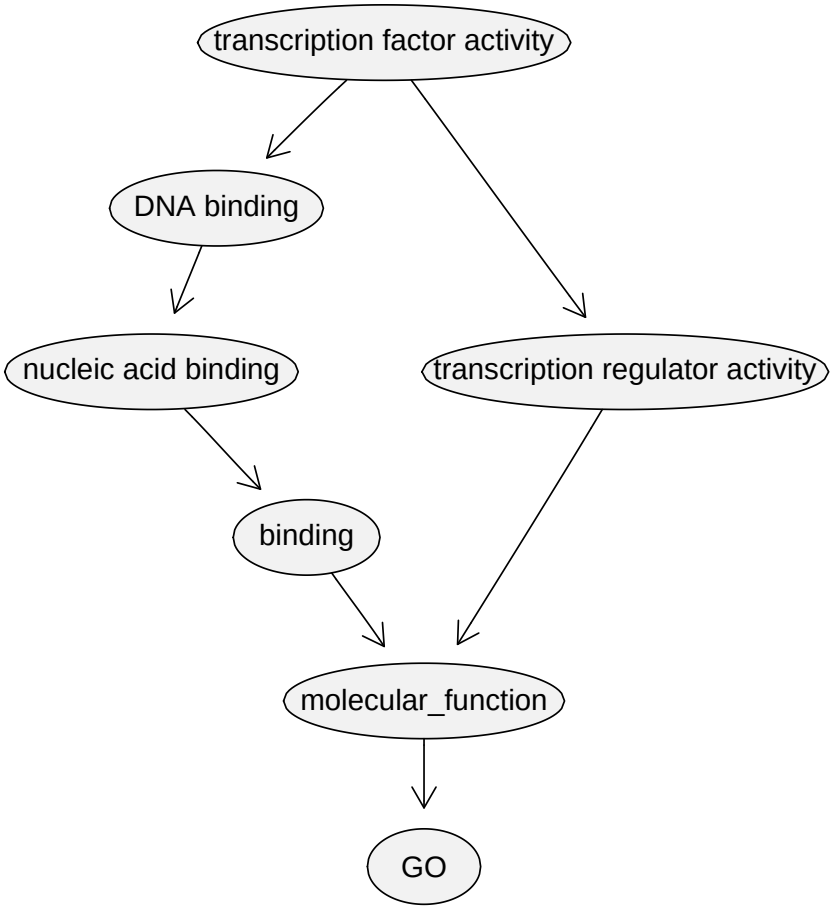


Figure 1: Graph of GO relationships for the term "transcription factor activity".

Induced GO graphs

```
> tfch <- GOMFCHILDREN$"GO:0003700"

[1] "GO:0003705"

> tfchild <- mget(tfch, GOTERM)

$"GO:0003705"
GOID = GO:0003705
Term = RNA polymerase II transcription factor
     activity, enhancer binding
Definition = Functions to initiate or regulate RNA
     polymerase II transcription by binding a
     promoter or enhancer region of DNA.
Ontology = MF
```

KEGG

- KEGG provides on set of mappings from genes to pathways.
- For each package we provide mappings, you can also query the site directly using KEGGSOAP or any other software.
- One problem with the KEGG is that the data is not in a form that is amenable to computation. The cMAP project provides data that is somewhat more useful for constructing networks.

KEGG

Data available in the KEGG package includes:

KEGGEXTID2PATHID which provides mappings from either LocusLink (for human, mouse and rat) or Open Reading Frame (yeast) to KEGG pathways, a second environment, `KEGGPATHID2EXTID` contains the mappings in the other direction.

KEGGPATHID2NAME which provides mappings from the KEGG path ID to a name (textual description of the pathway). Only the numeric part of the KEGG pathway identifiers is used (and not the three letter species codes).

Counts per species

	ath	dme	hsa	mmu	rno	sce
Counts	105	105	133	123	115	98

Table 3: Pathway Counts Per Species

Exploring KEGG

- Consider the pathway `00140`
- species specific mappings, from a pathway to the genes it contains, are indicated by gluing together a three letter species code, such as **`hsa`** for homo sapiens, to the numeric pathway code.

```
> KEGGPATHID2NAME$"00140"
[1] "C21-Steroid hormone metabolism"
> KEGGPATHID2EXTID$hsa00140
 [1] "1109" "1583" "1584" "1585" "1586" "1589" "3283" "3284"
 [9] "3290" "3291" "6718"
> KEGGPATHID2EXTID$sce00140
[1] "YGL001C"
```

- We look up PAK1, which has LocusLink ID `5058` in humans, and find that it is involved in two pathways, **`hsa04010`** and **`hsa04510`**.

- For mice, the LocusLink ID for PAK1 is 18479.

```
> KEGGEXTID2PATHID$"5058"
[1] "hsa04010" "hsa04510"
> KEGGEXTID2PATHID$"18479"
[1] "mmu04010"
```

cMAP

- The cancer Molecular Analysis Project (cMAP) is a project that provides software and data for the comprehensive exploration of data relevant to cancer.
- cMAP provides pathway data in a format that is amenable to computational manipulation.

```
> keggproc <- eapply(cMAPKEGGINTERACTION, function(x) x$process)
> table(unlist(keggproc))
character(0)
> cartaproc <- eapply(cMAPCARTAINTERACTION, function(x) x$process)
> length(table(unlist(cartaproc)))
[1] 0
```

cMAP

- We study the pathway labeled hsa00020 as an example.

```
> cMK <- ls(cMAPKEGGPATHWAY)
> spec <- substr(cMK, 1, 3)
> table(spec)

spec
hsa map
 81   4

> cMK[[2]]

[1] "hsa00020"

> pw2 <- cMAPKEGGPATHWAY[[cMK[2]]]
> names(pw2)

[1] "id"        "organism"  "source"    "name"
[5] "component"

> pw2$name

[1] "citrate cycle (tca cycle)"

> pw2$component

 [1]   63   71 3473   80   68   78   79   77   72   65   55
```

```
[12]   82   74   84   75   64   83   61   58   59   69   81
[23]   60   56   73   66   76 1367   62   54
```

- We select the first element of the `pw2$component`; it is an interaction so we first extract it, and then explore its components.

```
> getI1 <- get("63", cMAPKEGGINTERACTION)
> unlist(getI1[1:4])
    source    process reversible  condition
    "KEGG"         NA     "TRUE"         NA
> unlist(getI1[[5]][[2]])
      id     edge     role location activity
       2       NA       NA       NA       NA
```

- We find that ATP is an input to the citrate cycle:

```
> get("2", cMAPKEGGMOLECULE)[[2]][7:8]
                          AS                                  AS
"adenosine 5'-triphosphate"                              "ATP"
```

Homology

- Two genes are said to be homologous if they have descended from a common ancestral DNA sequence.
- there is some interest in using homologous genes when studying related phenomena across species
- there is one homology package for each species; a three letter species name (for homo sapiens `hsa`) and a suffix of `homology`
- the mappings provided are between HomoloGene's identifiers and a variety of other commonly used identifiers, namely LocusLink and UniGene.

```
> library("hsahomology")
> ls("package:hsahomology")
[1] "hsahomology"          "hsahomologyACC2HGID"
[3] "hsahomologyDATA"      "hsahomologyHGID"
[5] "hsahomologyHGID2ACC" "hsahomologyHGID2LL"
[7] "hsahomologyLL2HGID"  "hsahomologyORGCODE"
```

Homology

- the data linking genes is provided in `hsahomologyDATA`
- each element in the list represents a gene that is homologous to the key
- there are three different types of homology that are recorded and represented in the data.
- a single letter is used; the type can be:
  - `B` (reciprocal best best between three or more organisms),
  - `b` (reciprocal best match between two organisms), or
  - `c` (curated homology relationship between two organisms)

Homology Example

- the code for homo sapiens is `9606`
- the Homologene project uses its own set of gene identifiers,
- to find the homologs for estrogen receptor 1 (ESR1) we use its LocusLink ID (2099) and find the corresponding HomoloGene ID.

```
> esrHG <- hsahomologyLL2HGID$"2099"
> hesr <- get(as.character(esrHG), hsahomologyDATA)
> sapply(hesr, function(x) x$homoOrg)
10090 10090 10116 10090 10116  8022  8355  8364  9823  7955
"mmu" "mmu" "rno" "mmu" "rno" "omy" "xla" "xtr" "ssc" "dre"
 9913
"bta"
```

Homology Cross Species

- to find all potential homologs in one species, starting with the genes in a different species.
- to find all Xenopus Laevis homologs for human genes we use the following code

```
> hXp <- eapply(hsahomologyDATA, function(x) {
+     gd <- sapply(x, function(x) if (!is.na(x$homoOrg) &&
+         x$homoOrg == "xla")
+         TRUE
+     else FALSE)
+     x[gd]
+ })
> lh <- sapply(hXp, length)
> hXp2 <- hXp[lh > 0]
```

- we find 7021 human genes that have a potential homolog in Xenopus Laevis.

Visualizing Genomic Data

- High-density scatterplots using hexbin and
- Heatmap: rectangular false-color displays.
- Visualizing distances
- Special plots: genomic coordinates

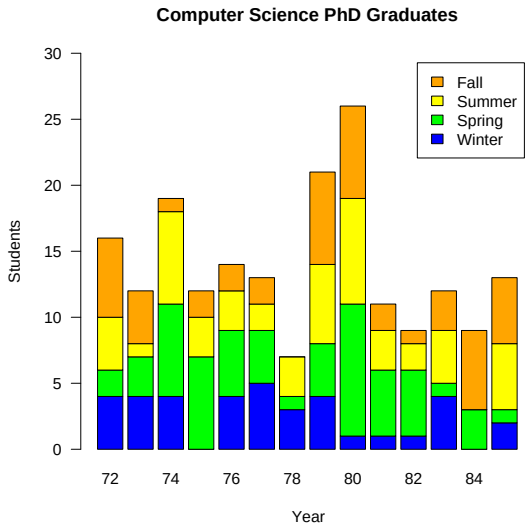
Visualization

- visualization is more than simply plotting data
- visualization is the process of plotting data so that the contents are easily and accurately perceived by the user
- irrelevant details should be suppressed, and important comparisons enhanced and put on an appropriate scale, or encoded using color etc.

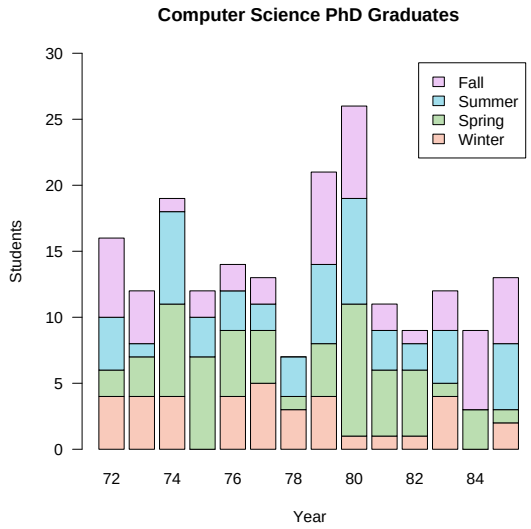
Use of Color

- color can greatly enhance perception
- the RColorBrewer package provides a number of different palettes to choose from
- There are 3 types of palettes, sequential, diverging, and qualitative.
- see the manual page for `brewer.pal` for more details
- the R function `hcl` provides another set of colors (which may be more appropriate for statistical graphics).
- both `colorRamp` and `colorRampPalette` provide tools for creating palettes of colors that map between two chosen colors.

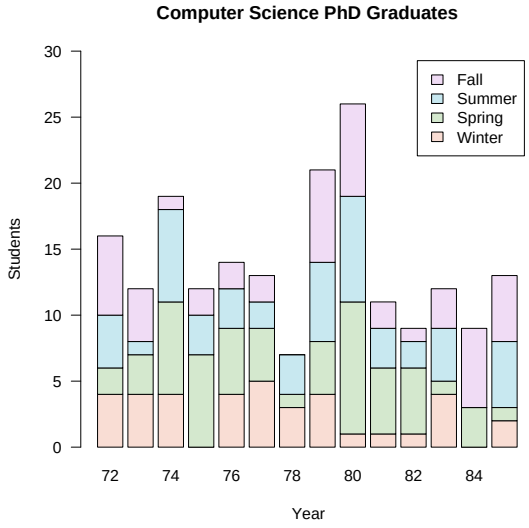
```
> csphd(c("blue", "green", "yellow", "orange"))

> csphd(hcl(h = c(30, 120, 210, 300)))

> csphd(hcl(h = c(30, 120, 210, 300), c = 20, l = 90,
+     fixup = FALSE))

> csphd(hcl(h = seq(60, 240, by = 60)))
```


Computer Science PhD Graduates
Students
Year
Fall
Summer
Spring
Winter
0
5
10
15
20
25
30
72
74
76
78
80
82
84

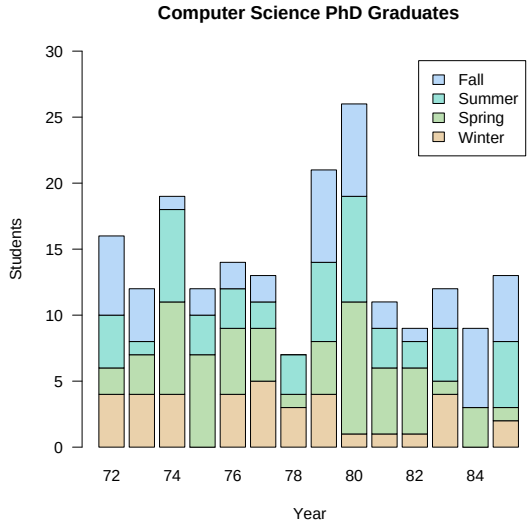
a)


Computer Science PhD Graduates
Students
Year
Fall
Summer
Spring
Winter
0
5
10
15
20
25
30
72
74
76
78
80
82
84

b)


Computer Science PhD Graduates
Students
Year
Fall
Summer
Spring
Winter
0
5
10
15
20
25
30
72
74
76
78
80
82
84

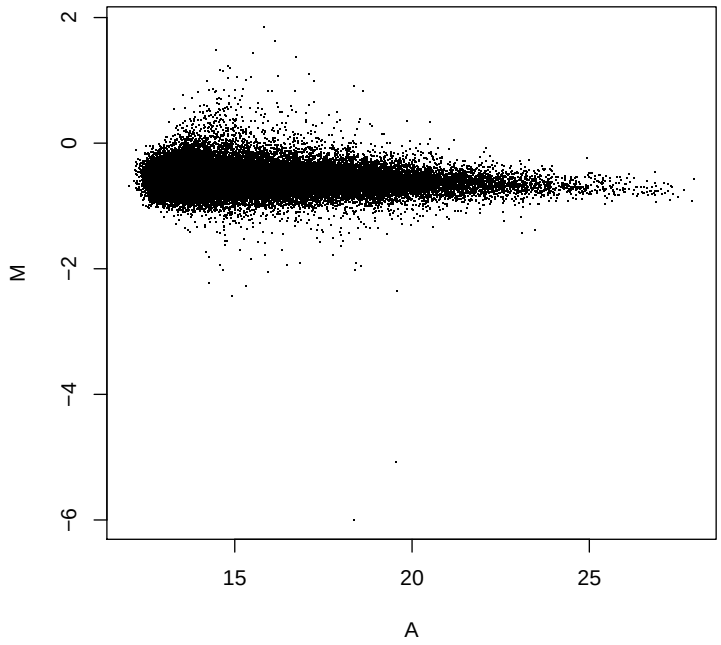
c)


Computer Science PhD Graduates
Students
Year
Fall
Summer
Spring
Winter
0
5
10
15
20
25
30
72
74
76
78
80
82
84

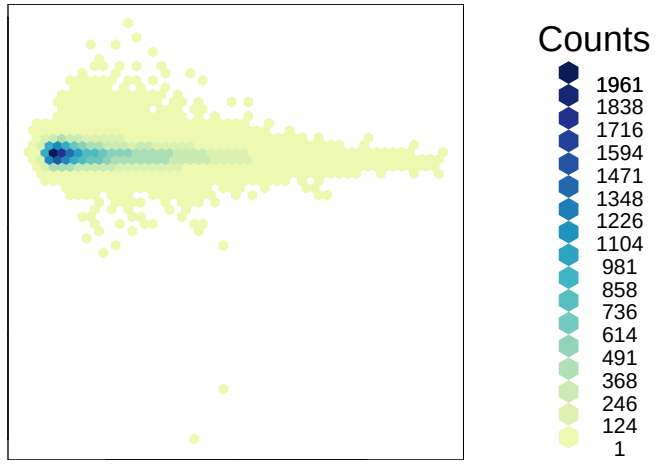
d)

High-density scatterplots

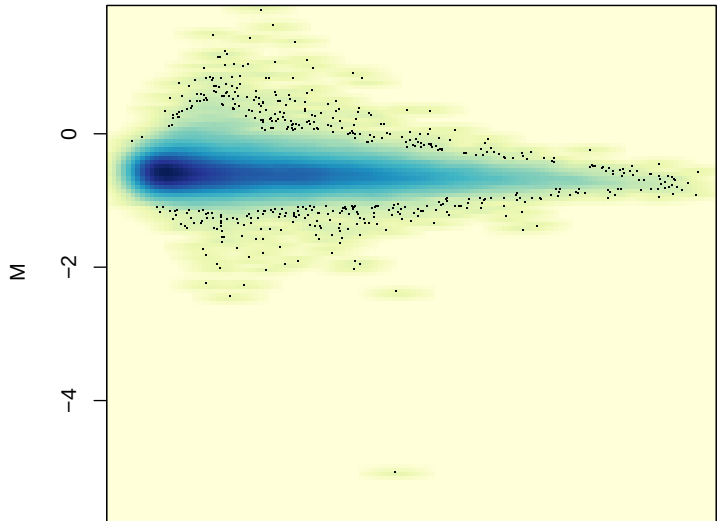
```
> library("affydata")
> data("Dilution")
> x <- log2(exprs(Dilution)[, 1:2])
> x <- x %*% cbind(A = c(1, 1), M = c(-1, 1))

> plot(x, pch = ".")

> library("hexbin")
> library("geneplotter")
> hb <- hexbin(x, xbins = 50)
> plot(hb, colramp = colorRampPalette(brewer.pal(9,
+     "YlGnBu")[-1]))

> library("prada")
> smoothScatter(x, nrpoints = 500)

> plot(x, col = densCols(x), pch = 20)
```

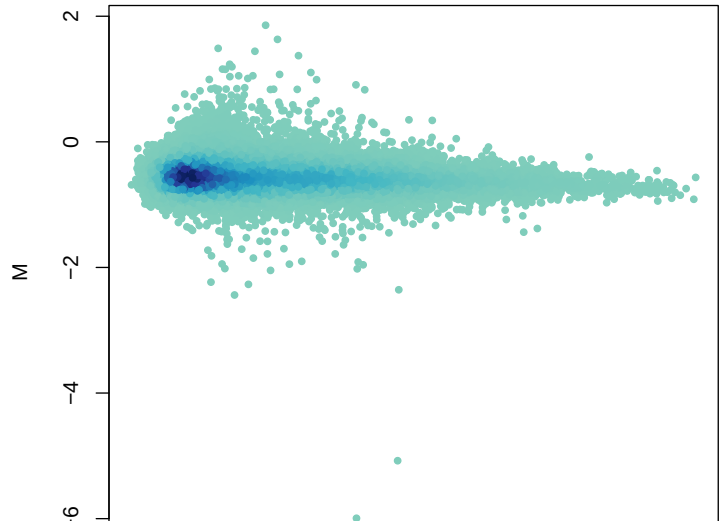

M
A
2
0
−2
−4
−6
15
20
25

a)


Counts
1961
1838
1716
1594
1471
1348
1226
1104
981
858
736
614
491
368
246
124
1

b)


M
2
0
−2
−4


M
2
0
−2
−4

Heatmaps

- A heatmap is a two-dimensional, rectangular, colored grid.
- It displays data that themselves come in the form of a rectangular matrix.
- The color of each rectangle is determined by the value of the corresponding entry in the matrix.
- The rows and columns of the matrix can be rearranged independently.
- Usually they are reordered so that similar rows are placed next to each other, and the same for columns.

Heatmaps

- the function `heatmap` is an implementation with many options
- users can control the ordering of rows and columns independently of each other
- they can use row and column labels of their own choosing or select their own color scheme
- they can also add a colored bar to annotate either the row or the column data

Heatmaps

```
> library("ALL")
> data("ALL")
> selSamples <- ALL$mol.biol %in% c("ALL1/AF4",
+     "E2A/PBX1")
> ALLs <- ALL[, selSamples]
> ALLs$mol.biol <- factor(ALLs$mol.biol)
> colnames(exprs(ALLs)) <- paste(ALLs$mol.biol,
+     colnames(exprs(ALLs)))

> library("genefilter")
> g <- split(1:length(ALLs$mol.biol), ALLs$mol.biol)
> meanThr <- log2(100)
> tThr <- qt(0.9999, df = sum(listLen(g)) - 2)
> s1 <- rowMeans(exprs(ALLs)[, g[[1]]]) > meanThr
> s2 <- rowMeans(exprs(ALLs)[, g[[2]]]) > meanThr
> s3 <- abs(fastT(exprs(ALLs), g[[1]], g[[2]], var.equal = TRUE)$z) >
+     tThr
> selProbes <- (s1 | s2) & s3
> ALLhm <- ALLs[selProbes, ]
```

```
> hmcol <- colorRampPalette(brewer.pal(10, "RdBu"))(256)
> spcol <- ifelse(ALLhm$mol.biol == "ALL1/AF4",
+     "goldenrod", "skyblue")
> heatmap(exprs(ALLhm), col = hmcol, ColSideColors = spcol)
```

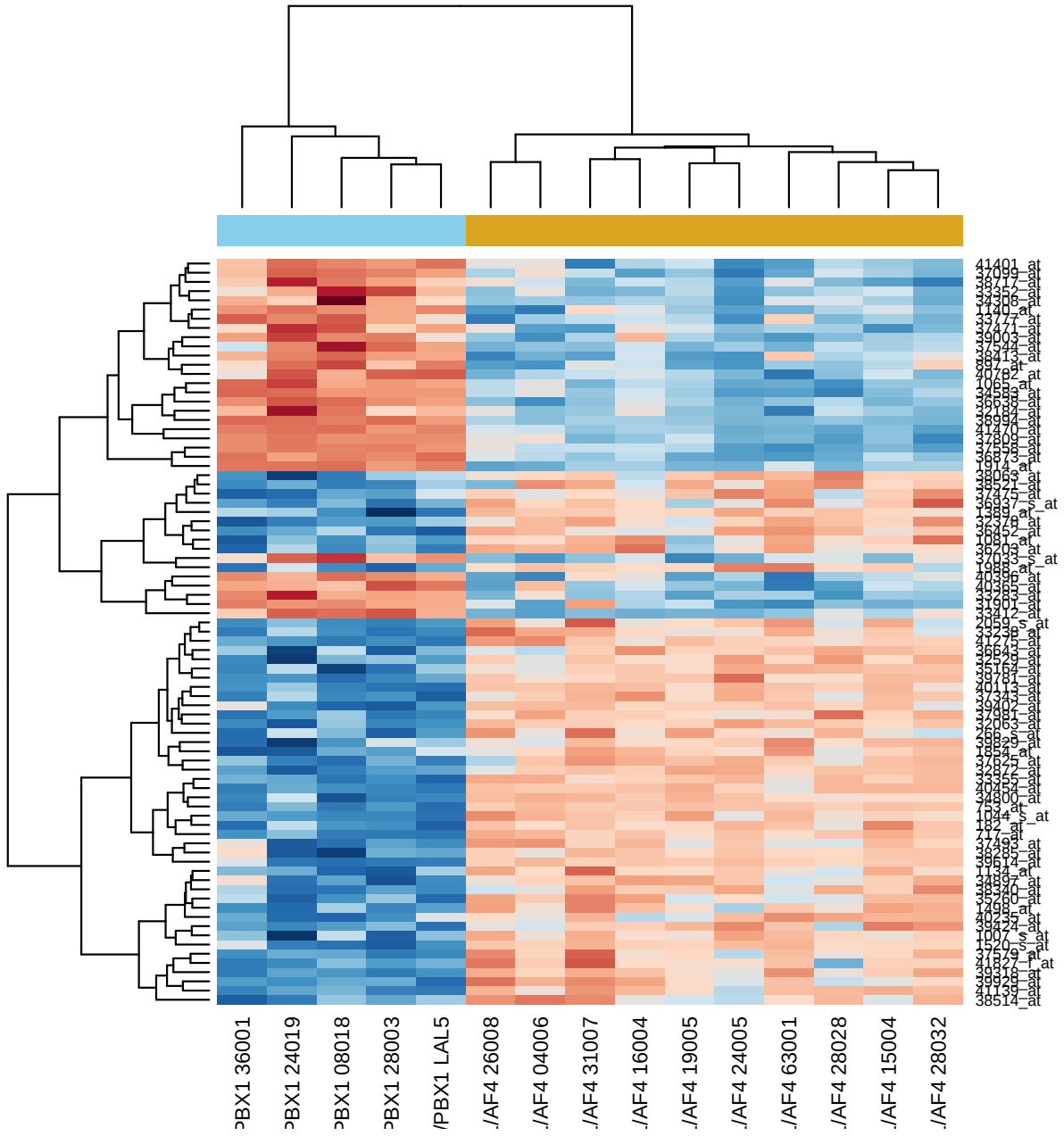


Figure 4: A heatmap comparing the ALL1/AF4 group (brown) to the E2A/PBX1 group (light blue)

Heatmaps of residuals

- in most cases you can find residuals from any model fit to gene expression data
- for example, even fitting t-tests to each row (gene) yields residuals
- the residuals are the same size as the original data and a heatmap can reveal structure in them
- often peculiar arrays (those with too many high or low values) can be detected
- as can sets of genes that the model does not fit, in the same way

```
> predict.MArrayLM <- function(f, design = f$design) {
+     return(f$coefficients %*% t(design))
+ }
> esFit <- predict(fit)
> res <- exprs(esEset) - esFit

> sel <- order(fit$coefficients[, "ES:T48"], decreasing = TRUE)[1:50]
> four.groups <- as.integer(factor(colnames(exprs(esEset))))
> csc <- brewer.pal(4, "Paired")[four.groups]
> heatmap(exprs(esEset)[sel, ], col = hmcol, ColSideColors = csc)

> heatmap(res[sel, ], col = hmcol, ColSideColors = csc)
```

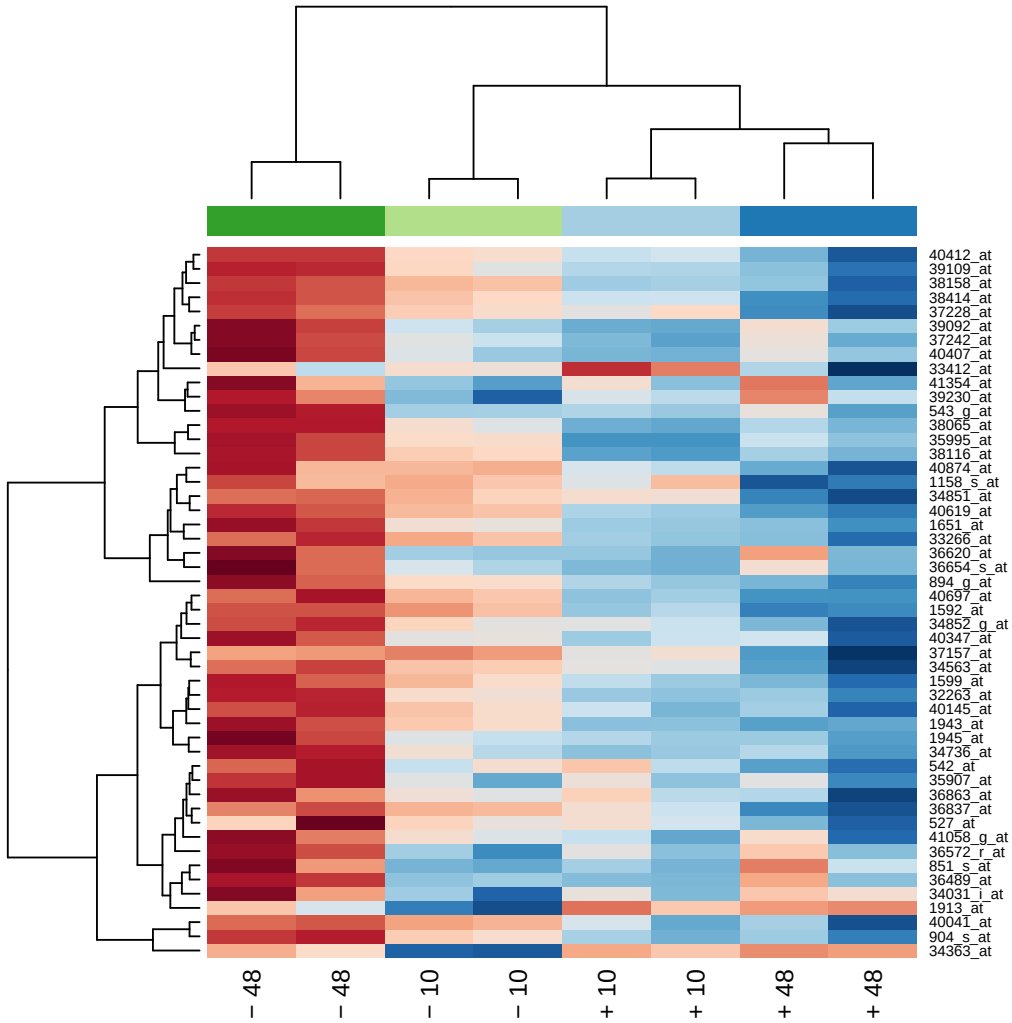


Figure 5: Heatmap of the estrogen data for the 50 probesets with the highest treatment–time interaction. The horizontal color bar corresponds to the 2×2 factor levels for treatment and time.

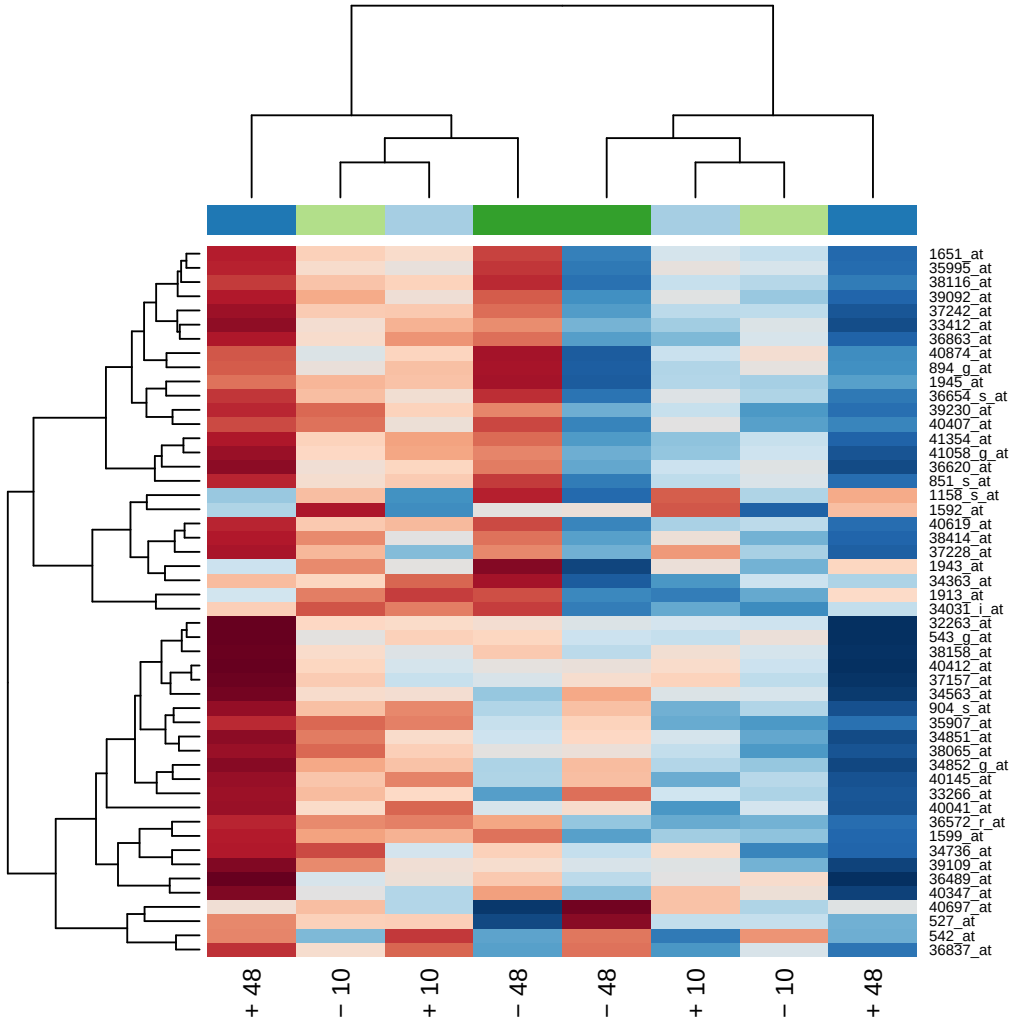
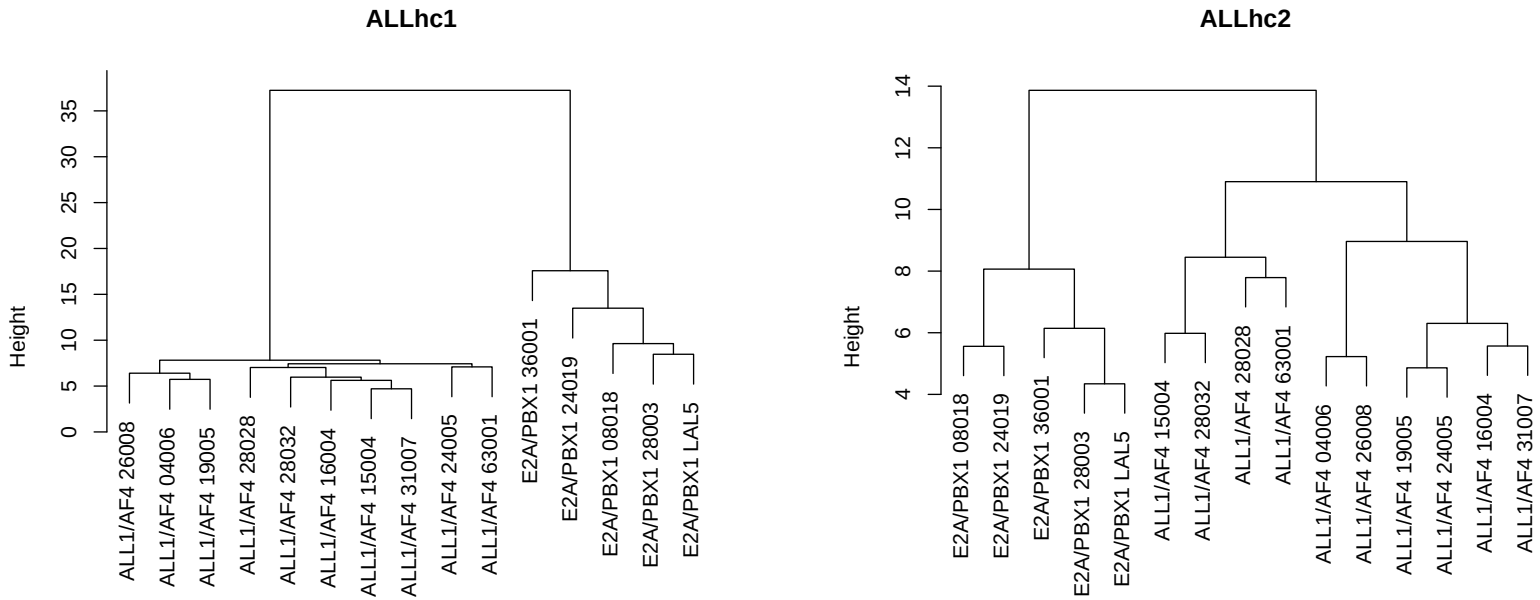


Figure 6: Heatmap of the residuals of the linear model for the estrogen data.

Visualizing Distances

- All machine learning involves the use and comparison of distances.
- the selection of an appropriate distance is important as are the development and use of methods to visualize distances
- unfortunately one of the most widely used tools (the dendrogram) is the least suited to this objective and should be avoided where possible.

```
> standardize <- function(z) {
+     rowmed <- apply(z, 1, median)
+     rowmad <- apply(z, 1, mad)
+     rv <- sweep(z, 1, rowmed)
+     rv <- sweep(rv, 1, rowmad, "/")
+     return(rv)
+ }
> ALLhme <- exprs(ALLhm)
> ALLdist1 <- dist(t(standardize(ALLhme)))
> ALLhc1 <- hclust(ALLdist1)

> plot(ALLhc1, xlab = "", sub = "", main = "ALLhc1")

> ALLsub2 <- exprs(ALLs[(s1 | s2), ])
> rowMads <- apply(ALLsub2, 1, mad)
> ALLsub2 <- ALLsub2[rowMads > 1.4, ]
> ALLdist2 <- dist(t(standardize(ALLsub2)))
> ALLhc2 <- hclust(ALLdist2)

> plot(ALLhc2, xlab = "", sub = "", main = "ALLhc2")
```



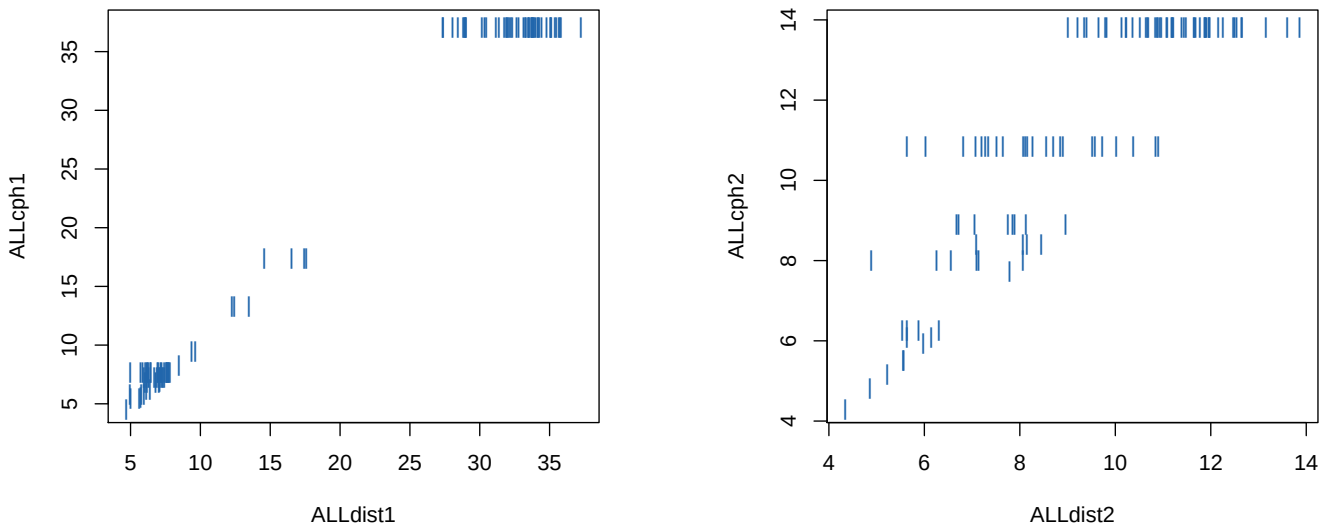
a) b)

Figure 7: Dendrograms for the ALL1/AF4 and E2A/PBX1 samples. The clustering was obtained a) using the 81 probes in `ALLhme` that were selected in Section **??** by the t-statistic, b) using the 58 probes in `ALLsub2` that were filtered for their overall variability.

Cophentic Distance

- to compute the dendrogram you must have some distance between the objects
- the dendrogram induces a second distance between the objects
- the cophenetic correlation is the correlation between these two distances
- the larger the cophenetic correlation, the more the dendrogram reflects the original distances, and the better the dendrogram is as a visual representation of the data

```
> mypal <- brewer.pal(7, "RdBu")
> blue <- mypal[7]
> red <- mypal[1]
> ALLcph1 <- cophenetic(ALLhc1)
> cor(ALLdist1, ALLcph1)

[1] 0.99

> plot(ALLdist1, ALLcph1, pch = "|", col = blue)

> ALLcph2 <- cophenetic(ALLhc2)
> cor(ALLdist2, ALLcph2)

[1] 0.877

> plot(ALLdist2, ALLcph2, pch = "|", col = blue)
```



a) b)

Figure 8: Scatterplots of actual distances versus cophenetic distances. a) distances calculated with t-test selected probes, b) variability-selected probes. Each pair of distances is shown by a vertical bar. Note that the cophenetic distances only take on a discrete set of values.

Multidimensional scaling

- MDS starts from a matrix of all pairwise distances or dissimilarities between n objects
- it tries to arrange n points in a k-dimensional Euclidean space such that the distances between the points are as much like the given distances as possible
- this can be done in a variety of ways and each leads to slightly different solutions
- R functions for carrying out MDS include: `cmdscale` in the `stats` package, and `isoMDS` and `sammon` in `MASS`
- like all dimension reduction methods, it is essential that the user check to see if the reduction is suitable (you can always compute it, it just does not have to be a good representation of the data)

Goodness of fit

- classical MDS solution, `cmdscale` returns two statistics.
- one is the sum of the eigenvalues for the components S divided by the sum of the absolute value of all eigenvalues
- the other is S divided by the sum of all positive eigenvalues
- to decide how many dimensions are necessary to adequately represent your data, it is useful to look at the *scree plot*, that is the plot of the goodness-of-fit statistic as a function of k
- a criterion for the choice of k is to pick a solution for which adding more dimensions does not significantly improve the goodness-of-fit.

Other MDS solutions

- `isoMDS` provides one form of non-metric MDS. It chooses a k-dimensional configuration to minimize the stress

$$s^2 = \frac{\sum_{i \neq j} (f(p_{ij}) - d_{ij})^2}{\sum_{i \neq j} d_{ij}^2}, \tag{1}$$

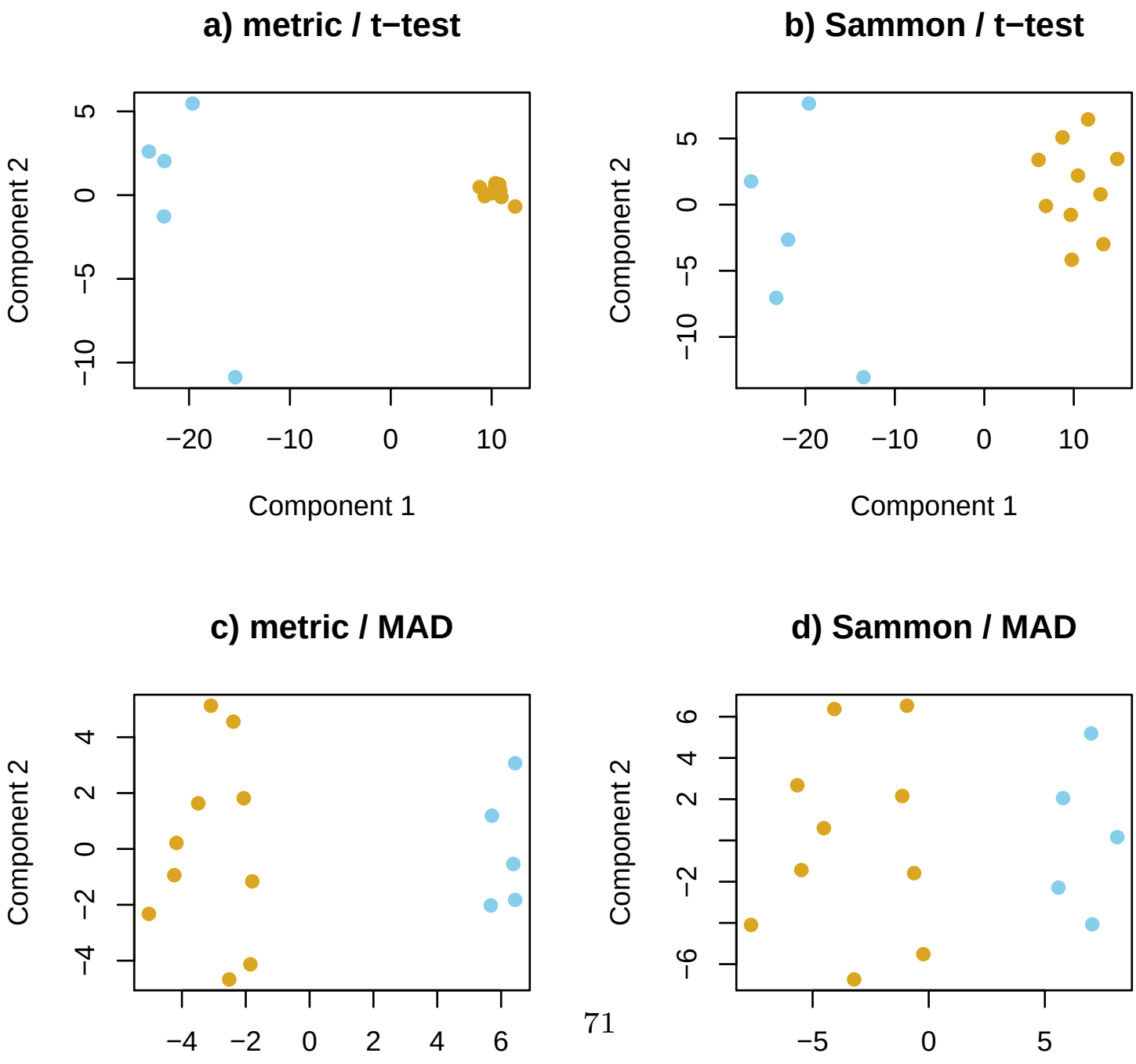
  where p_{ij} is the original distance matrix, f is a monotonic transformation, and d_{ij} are the distances between the MDS points.

- `sammon` uses a different loss-function

- The different variants of MDS lead to different relative importances of large versus small distances to the fitted MDS solution.

MDS Example Code

```
> cm1 <- cmdscale(ALLdist1, eig = TRUE)
> cm1$GOF

[1] 0.908 0.908

> samm1 <- sammon(ALLdist1, trace = FALSE)
> cm2 <- cmdscale(ALLdist2, eig = TRUE)
> cm2$GOF

[1] 0.646 0.646

> samm2 <- sammon(ALLdist2, trace = FALSE)

> ALLscol <- c("goldenrod", "skyblue")[as.integer(ALLs$mol.biol)]
> plot(cm1$points, col = ALLscol, ...)
```


a) metric / t−test
b) Sammon / t−test
c) metric / MAD
d) Sammon / MAD
Component 1
Component 2
−20
−10
0
10
−4
−2
2
4
6
−5
5
−6
−10

Using Heatmaps with Distances

- we can make use of the `heatmap` function for showing distances
- we call `heatmap` with both `sym=TRUE` and specify our own distance function
- the resulting plot is symmetric and there is a strong indication that there are two (or possibly three) groups of samples.

```
> heatmap(as.matrix(ALLdist2), sym = TRUE, col = hmcol,
+     distfun = function(x) as.dist(x))
```

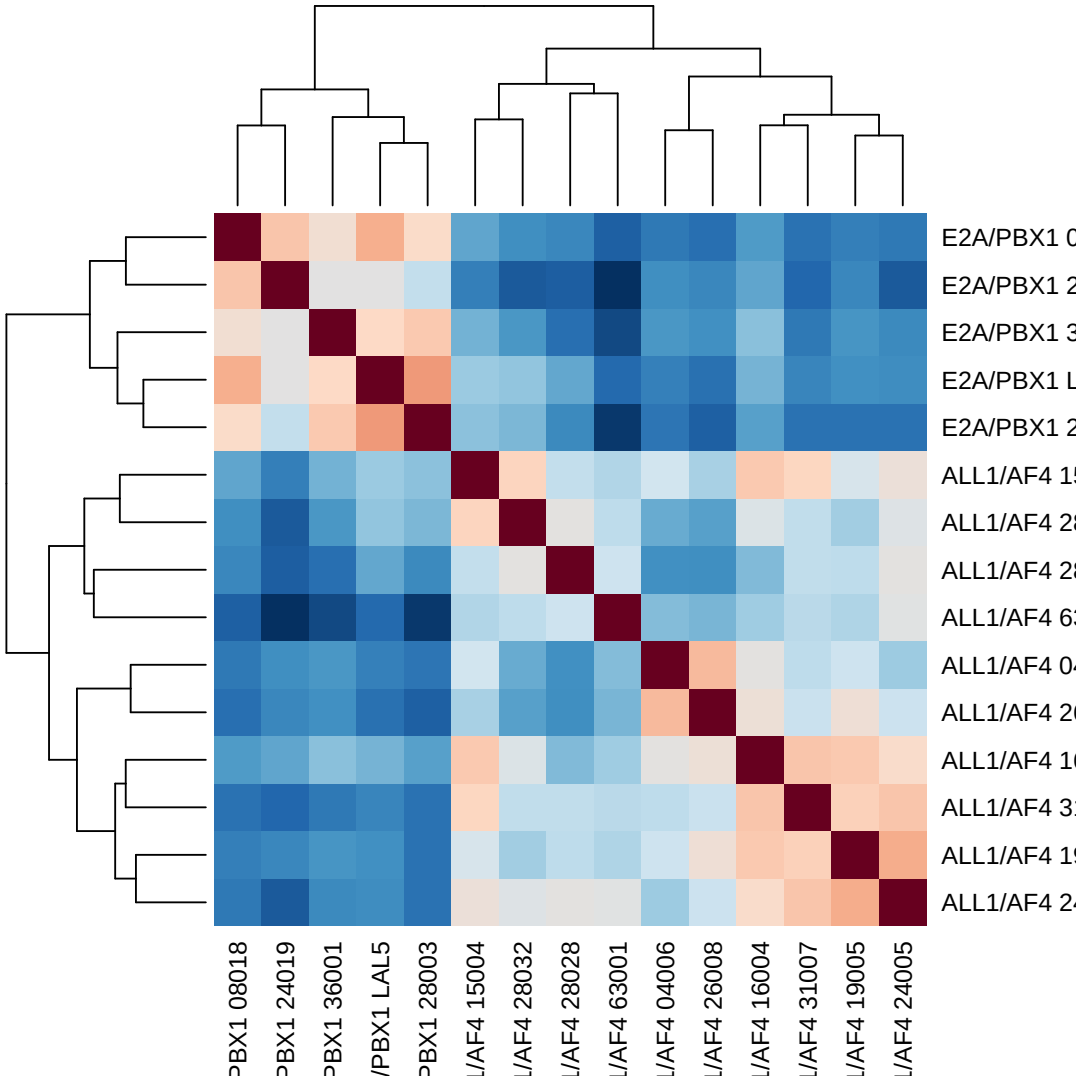


Figure 10: A heatmap of the between sample distances.

Plotting in Genomic Coordinates

- some of the genetic defects that are associated with cancer such as deletions and amplifications induce correlations in expression that are related to chromosomal proximity.
- Genomic regions of correlated transcription have also been identified in normal tissues.
- This motivates the development of tools that relate gene expression to chromosomal location.
- Genomic DNA is double stranded, one strand is called the *sense strand*, the other the *antisense strand*
- Both strands can contain coding sequences for genes, and the visualization methods we consider reflect this.

The geneplotter package

- We can build the object `chrLoc` using the code below

```
> library("geneplotter")
> chrLoc <- buildChromLocation("hgu95av2")
```

- this creates an object of class *chromLocation* which contains the location of all genes that were assayed on the HG-U95Av2 chip (for different experiments you would use a different chip).

Plotting chromosomal location

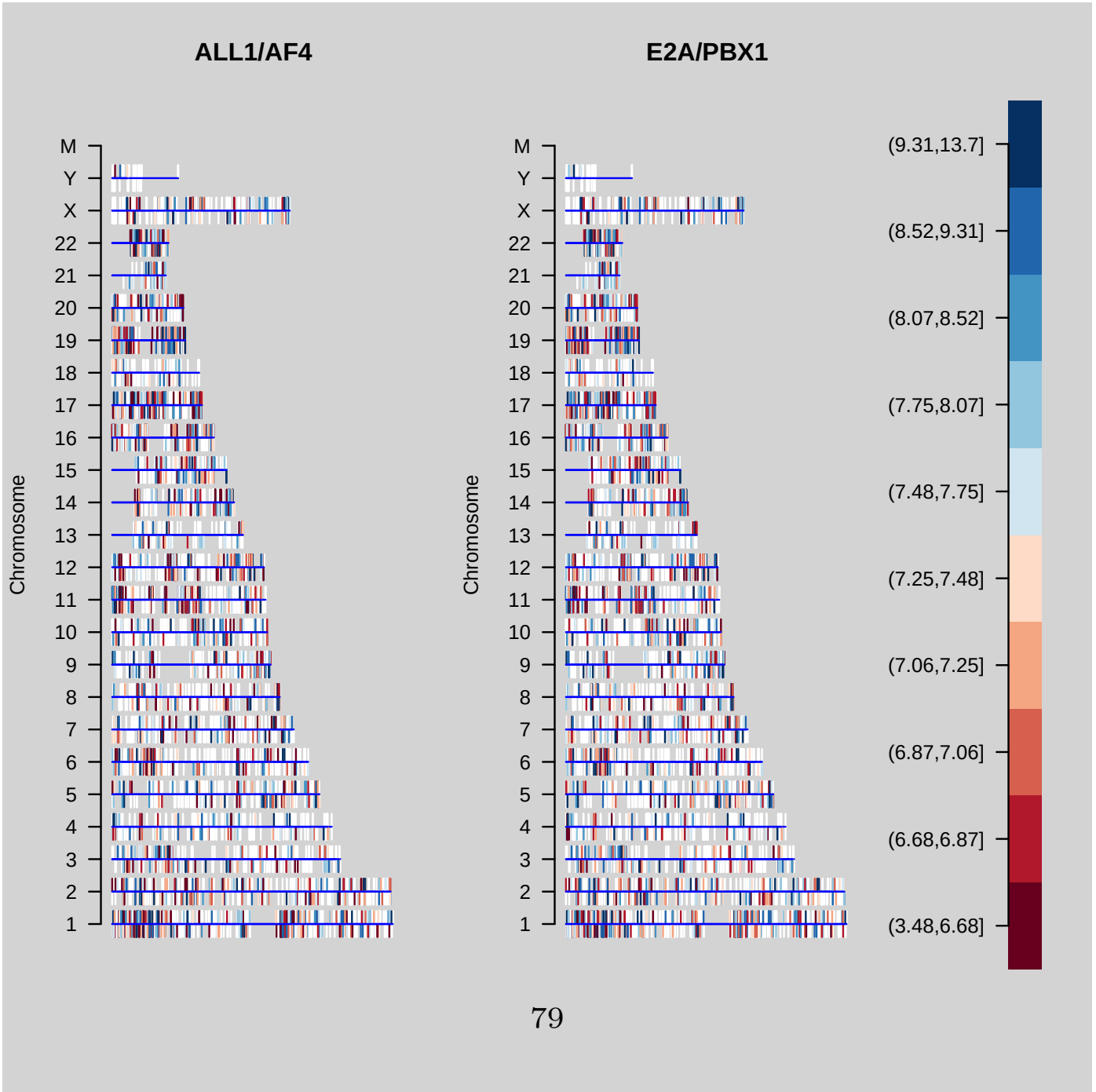
- select the highly expressing genes using `s1` and `s2` from above and compute the mean expression for each probe separately for the two groups of patient samples, ALL1/AF4 and E2A/PBX1.

```
> ALLch <- ALLs[s1 | s2, ]
> m1 <- rowMeans(exprs(ALLch)[, ALLch$mol.biol ==
+     "ALL1/AF4"])
> m2 <- rowMeans(exprs(ALLch)[, ALLch$mol.biol ==
+     "E2A/PBX1"])
```

- next compute the deciles of the combined data so the genes in each decile can be colored differently.

```
> deciles <- quantile(c(m1, m2), probs = seq(0,
+     1, 0.1))
> s1dec <- cut(m1, deciles)
> s2dec <- cut(m2, deciles)
> gN <- names(s1dec) <- names(s2dec) <- geneNames(ALLch)
```

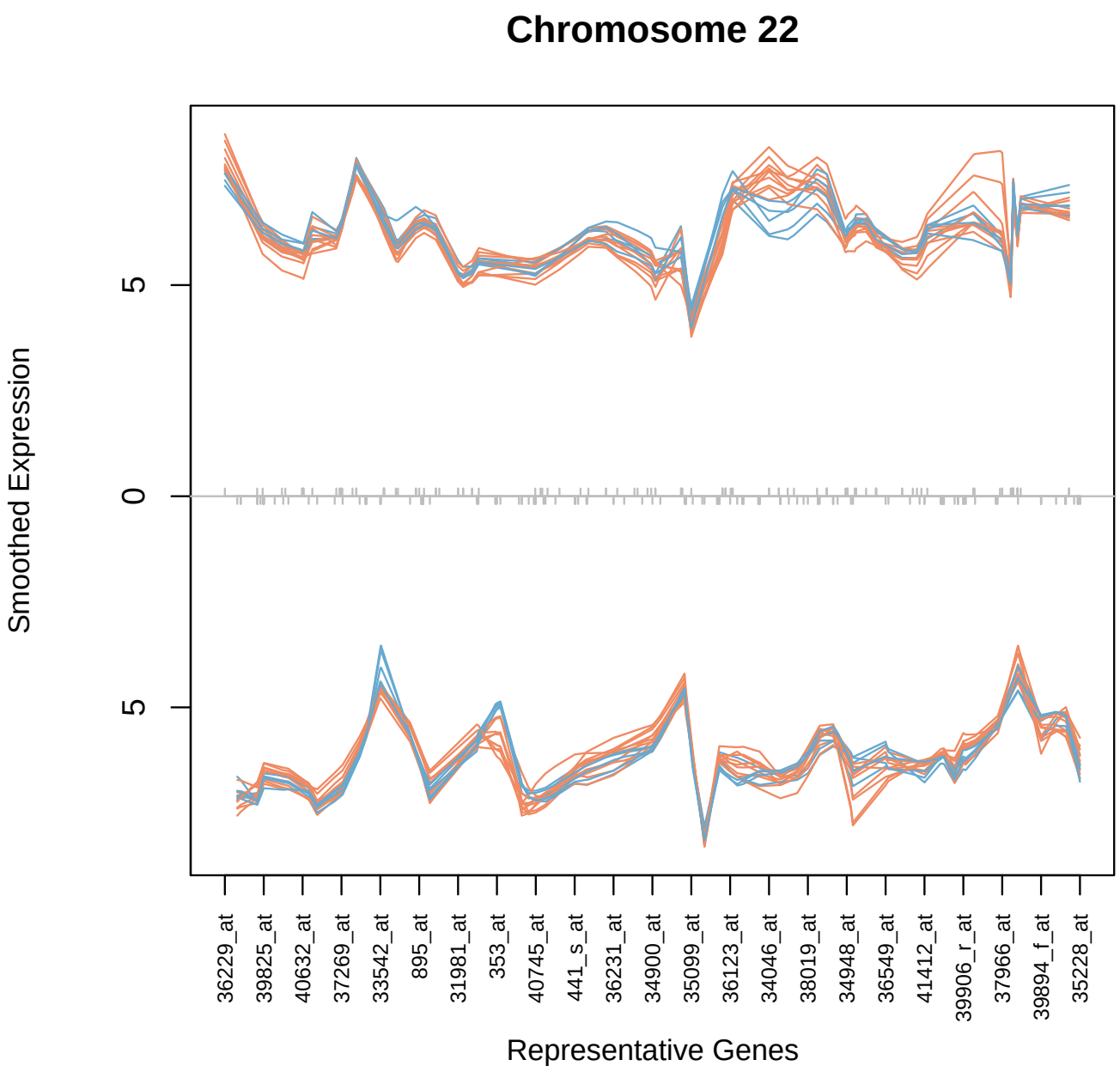
```
> colors <- brewer.pal(10, "RdBu")
> layout(matrix(1:3, nr = 1), widths = c(5, 5, 2))
> cPlot(chrLoc, main = "ALL1/AF4")
> cColor(gN, colors[s1dec], chrLoc)
> cPlot(chrLoc, main = "E2A/PBX1")
> cColor(gN, colors[s2dec], chrLoc)
> image(1, 1:10, matrix(1:10, nc = 10), col = colors,
+     axes = FALSE, xlab = "", ylab = "")
> axis(2, at = (1:10), labels = levels(s1dec), las = 1)
```


ALL1/AF4
E2A/PBX1
Chromosome
M
Y
X
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
(9.31,13.7]
(8.52,9.31]
(8.07,8.52]
(7.75,8.07]
(7.48,7.75]
(7.25,7.48]
(7.06,7.25]
(6.87,7.06]
(6.68,6.87]
(3.48,6.68]

Plotting Single Chromosomes

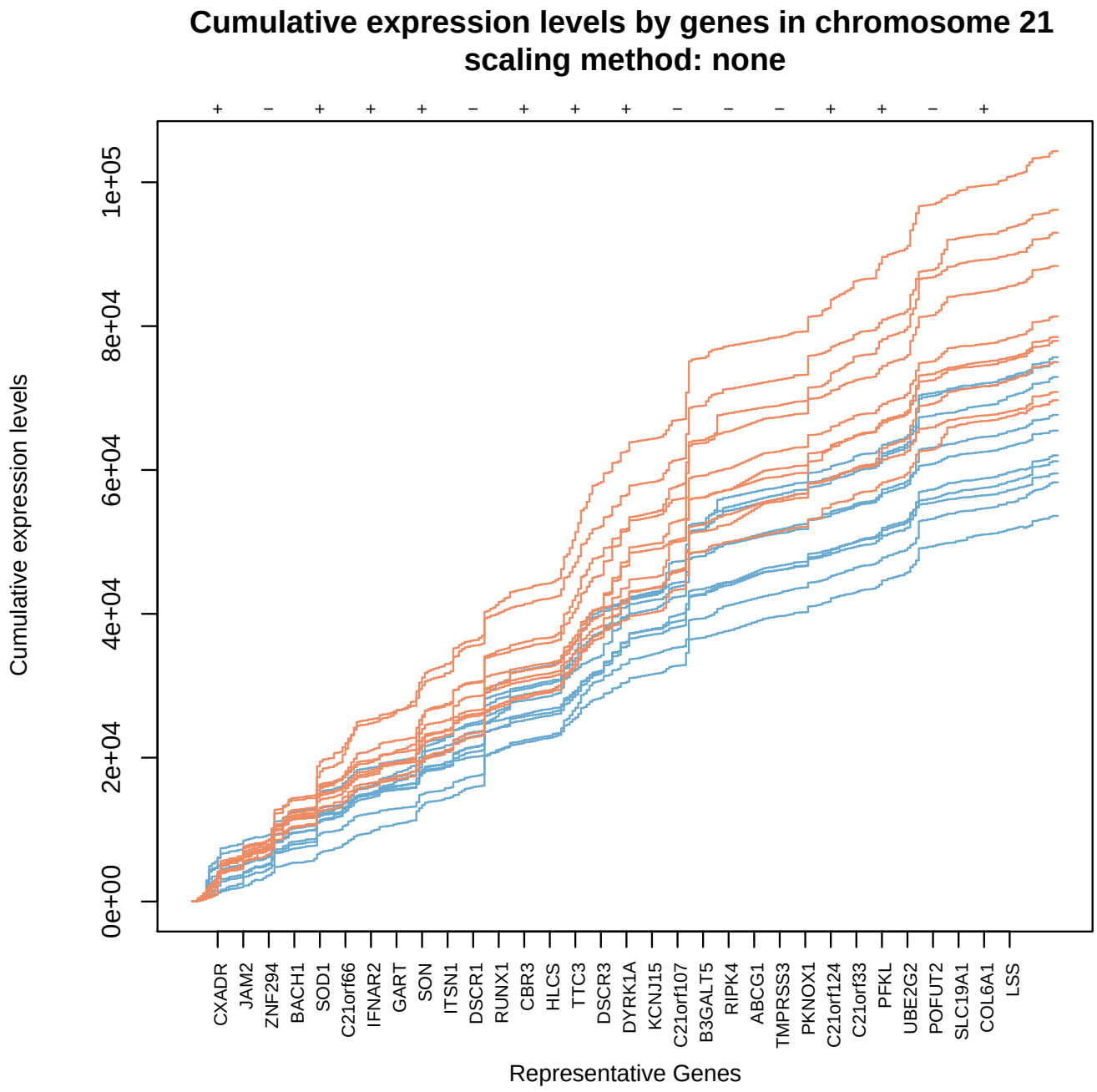
- `plotChr` produces one plot per chromosome.
- Each sample has two smooth lines; the one in the top half of the plot represents genes on the sense strand and the line in the bottom half of the plot represents expression for genes encoded on the antisense strand.
- Low expression values are near the center line and high expression values are towards the edge of the plot.

```
> par(mfrow = c(1, 1))
> msobj <- Makesense(ALLs, "hgu95av2")
> plotChr("22", msobj, col = ifelse(ALLs$mol.biol ==
+     "ALL1/AF4", "#EF8A62", "#67A9CF"), log = FALSE)
```


Chromosome 22
Smoothed Expression
5
0
5
36229_at
39825_at
40632_at
37269_at
33542_at
895_at
31981_at
353_at
40745_at
441_s_at
36231_at
34900_at
35099_at
36123_at
34046_at
38019_at
34948_at
36549_at
41412_at
39906_r_at
37966_at
39894_f_at
35228_at
Representative Genes

Cumulative Expression

- for some genomic aberrations looking at cummulative expression can be helpful
- it is hypothesized that losing one copy of a chromosome may only slightly alter gene expression and that the amount by which it changes is less than the variability in the population
- the function `alongChrom` plots gene expression with the genes ordered by their chromosomal location.
- the motivation for this is that on the level of individual loci, the technical and biological variability between samples can be large enough to obscur systematic differences due to copy number changes
- J.-P. Bourquin compared gene expression profiles between children with Down's syndrome (trisomy 21) and a transient myeloid disorder to children with different subtypes of AML.


Cumulative expression levels by genes in chromosome 21
scaling method: none
Cumulative expression levels
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
+ − + + + − + + + − − − + + − +
CXADR
JAM2
ZNF294
BACH1
SOD1
C21orf66
IFNAR2
GART
SON
ITSN1
DSCR1
RUNX1
CBR3
HLCS
TTC3
DSCR3
DYRK1A
KCNJ15
C21orf107
B3GALT5
RIPK4
ABCG1
TMPRSS3
PKNOX1
C21orf124
C21orf33
PFKL
UBE2G2
POFUT2
SLC19A1
COL6A1
LSS
Representative Genes