GS01 0163
Analysis of Microarray Data

Keith Baggerly and Kevin Coombes
Department of Bioinformatics and Computational Biology
UT M. D. Anderson Cancer Center
kabagg@mdanderson.org
kcoombes@mdanderson.org

25 October 2007

Lecture 17: Applied Clustering

- Clustering Methods
  - K-Means Clustering
  - Partitioning Around Medoids
  - Silhouette Widths
  - Principal Components
  - Principal Coordinates
- Project Normal
  - Project Normal Clustering
  - Abnormal Behavior
  - Problems and Solution

Review of hierarchical clustering

Last time: we looked at hierarchical clustering of the samples.

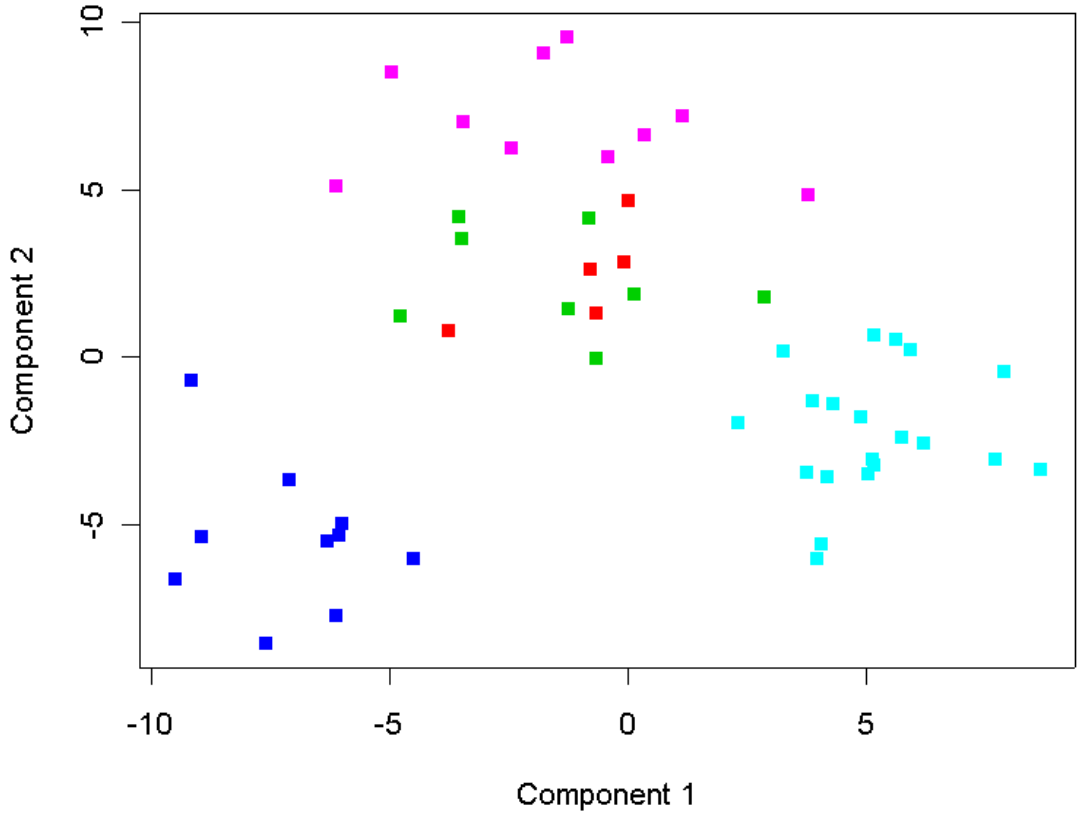
Key issues:

- What distance metric (euclidean, correlation, manhattan, canberra, minkowski) should we use?
- What linkage rule (average, complete, single, ward) should we use?
- Which clusters should we believe? (bootstrap resampling)
- How many clusters should we believe? (bootstrap)

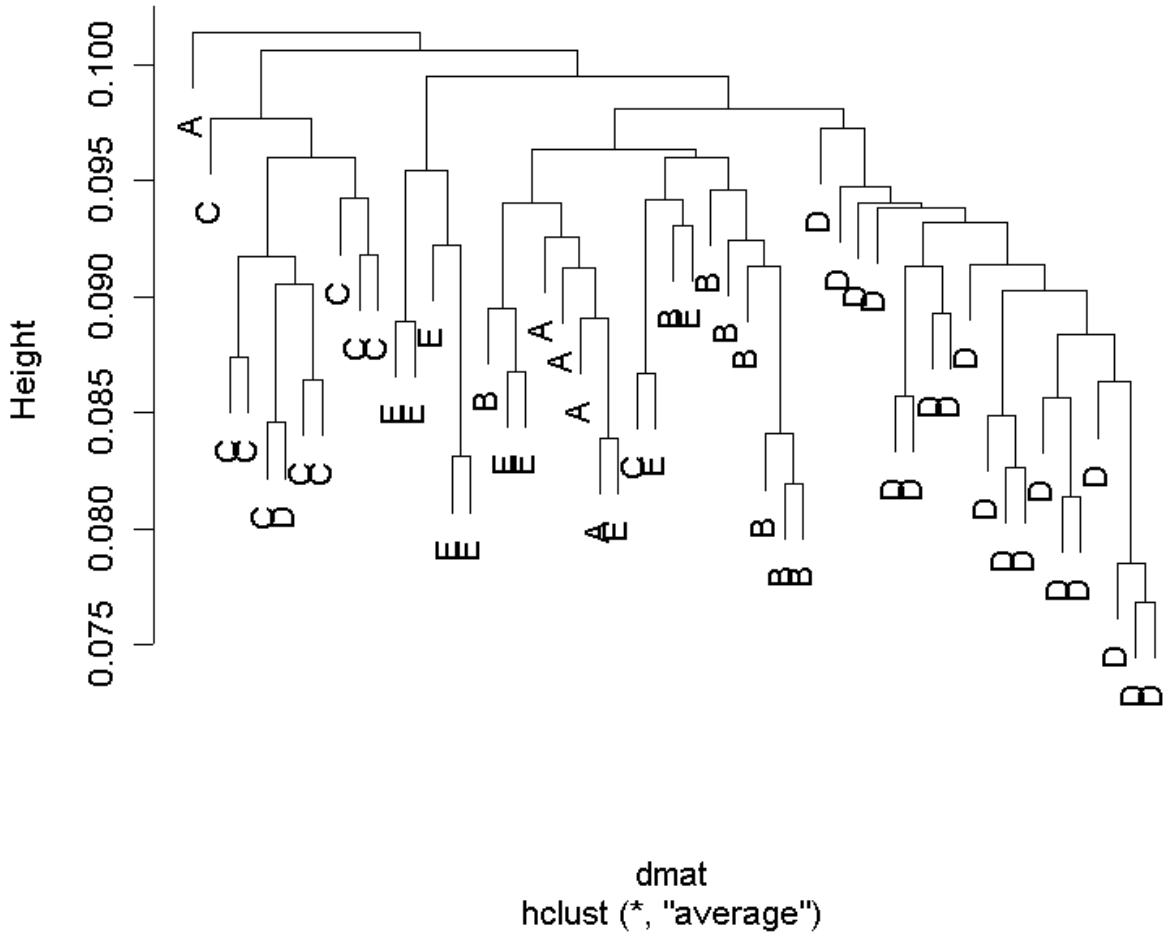
Is there any reason to believe that a hierarchical structure makes sense? Today, we'll look at some other clustering techniques.

Simulated data

To test some algorithms, we simulated data with 1000 genes and 5 different sample classes containing different numbers of samples. Here's a two-dimensional picture of the truth:



Hierarchical clusters (correlation; average)



Three of the classes (B, C, D) are mostly correct. The other two classes are less concentrated.

K-Means Clustering

Input: A data matrix, X, and the desired number of clusters, K.

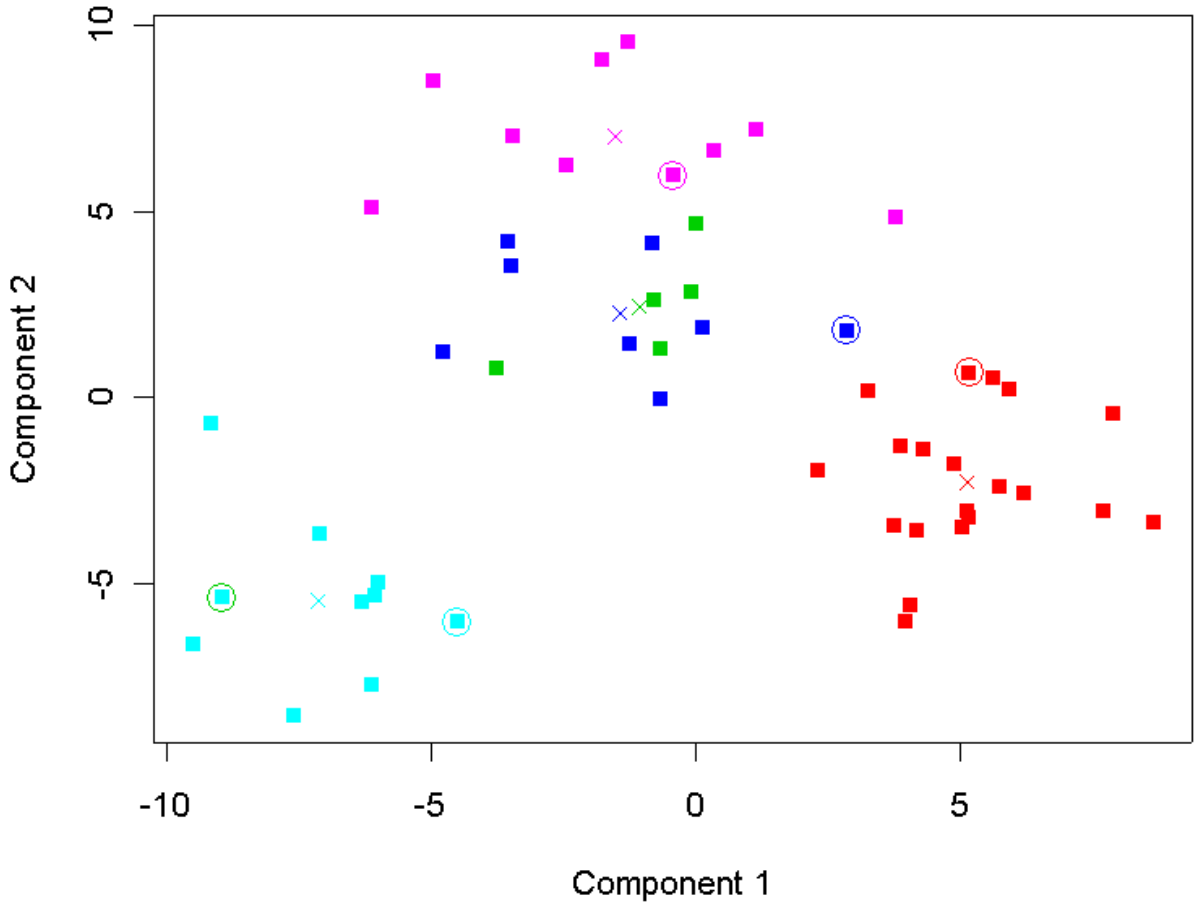
Output: For each sample i, a cluster assignment $C(i) \leq K$.

Idea: Minimize the within-cluster sum of squares

$$\sum_{c=1}^{K} \sum_{C(i)=c, C(j)=c} \sum_{\ell=1}^{N} (x_{i\ell} - x_{j\ell})^2$$

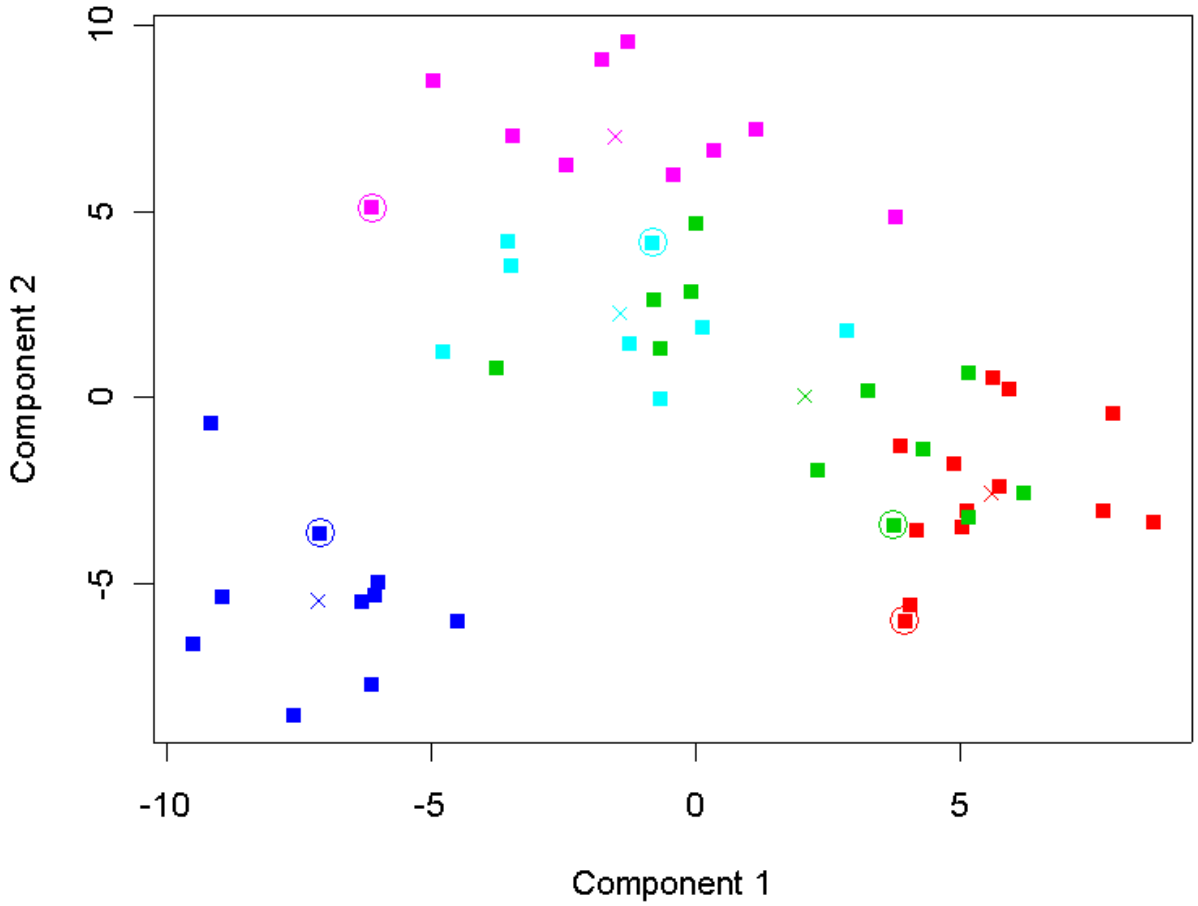
- Algorithm:
  1. Make an initial guess at the centers of the clusters.
  2. For each data point, find the closest cluster (Euclidean).
  3. Replace each cluster center by averaging data points that are closest to it.
  4. Repeat until the assignments stop changing.

K-Means, Take 1



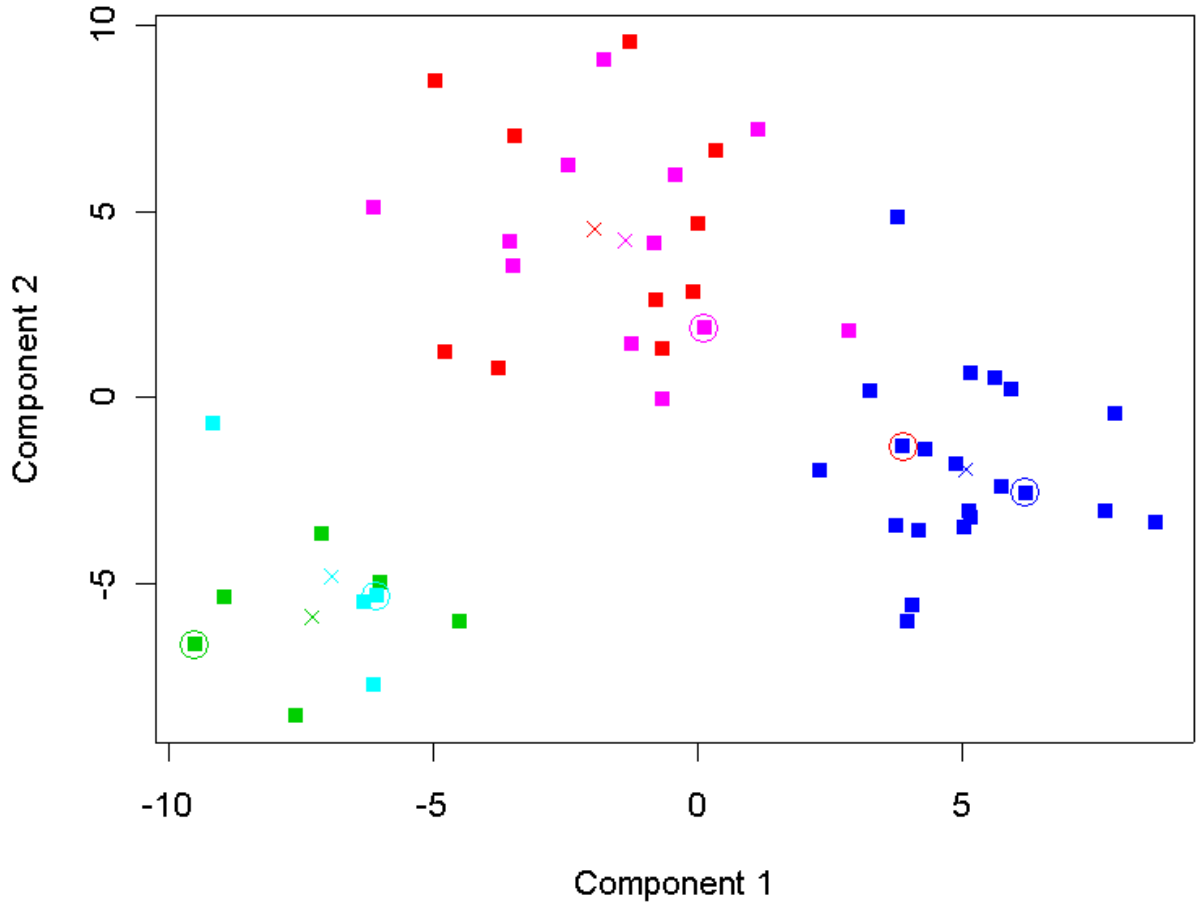
Perfect clustering! (Circles = starting group centers, X = final group centers)

K-Means, Take 2



Oops: bad starting points may mean bad clusters!

K-Means, Take 3



Local minima may not be global

K-means can be very sensitive to the choice of centers used as seeds for the algorithm. The problem is that the algorithm only converges to a local minimum for the within-cluster sum of squares, and different runs with randomly chosen centers (which is the default in the `kmeans` function in R) can converge to different local optima. You can see which of these three runs is better:

```
> sum(kres1$withinss)
[1] 25706.57
> sum(kres2$withinss)
[1] 25736.84
> sum(kres3$withinss)
[1] 25926.12
```

Local minima may not be global

There are two ways around the fact that local minima need not be global.

One is to find better starting seeds for the algorithm. For example, start with hierarchical clustering. Then cut the tree into five branches, and use the average of each branch as the starting points.

Alternatively, you can run the algorithm with many random seeds, keeping track of the within-cluster sum of squares:

```
require(ClassDiscovery)
repeatedKmeans(t(ldata), k=5, nTimes=100)
```

Multiple runs of the K-means algorithm

```
kcent <- sample(n.samples, 5)
kres <- kmeans(t(ldata), t(ldata[,kcent])
withinss <- sum(kres$withinss)
for (i in 1:100) {
  tcent <- sample(n.samples, 5)
  tres <- kmeans(t(ldata), t(ldata[,tcent]))
  print(sum(tres$withinss))
  if (sum(tres$withinss) < withinss) {
    kres <- tres
    kcent <- tcent
    withinss <- sum(kres$withinss)
  }
}
```

Can we use other measures of distance?

The K-means clustering algorithm has another limitation. (This is not the last one we will consider).

As described, it always uses Euclidean distance as the measure of dissimilarities between sample vectors. As we saw last time with hierarchical clustering, there are a large number of possible distances that we might want to use. Fortunately, a simple adjustment to the algorithm lets us work with any distance measure.

Partitioning Around Medoids (PAM)

Input: Data matrix, X, distance d, number of clusters, K.

Output: For each sample i, a cluster assignment $C(i) \leq K$.

Idea: Minimize the within-cluster distance

$$\sum_{c=1}^{K} \sum_{C(i)=c, C(j)=c} d(x_i, x_j)$$

- Algorithm:
  1. Make an initial guess at the centers of the clusters.
  2. For each data point, find the closest cluster.
  3. Replace each cluster center by the data point minimizing the total distance to other members in its cluster.
  4. Repeat until the assignments stop changing.

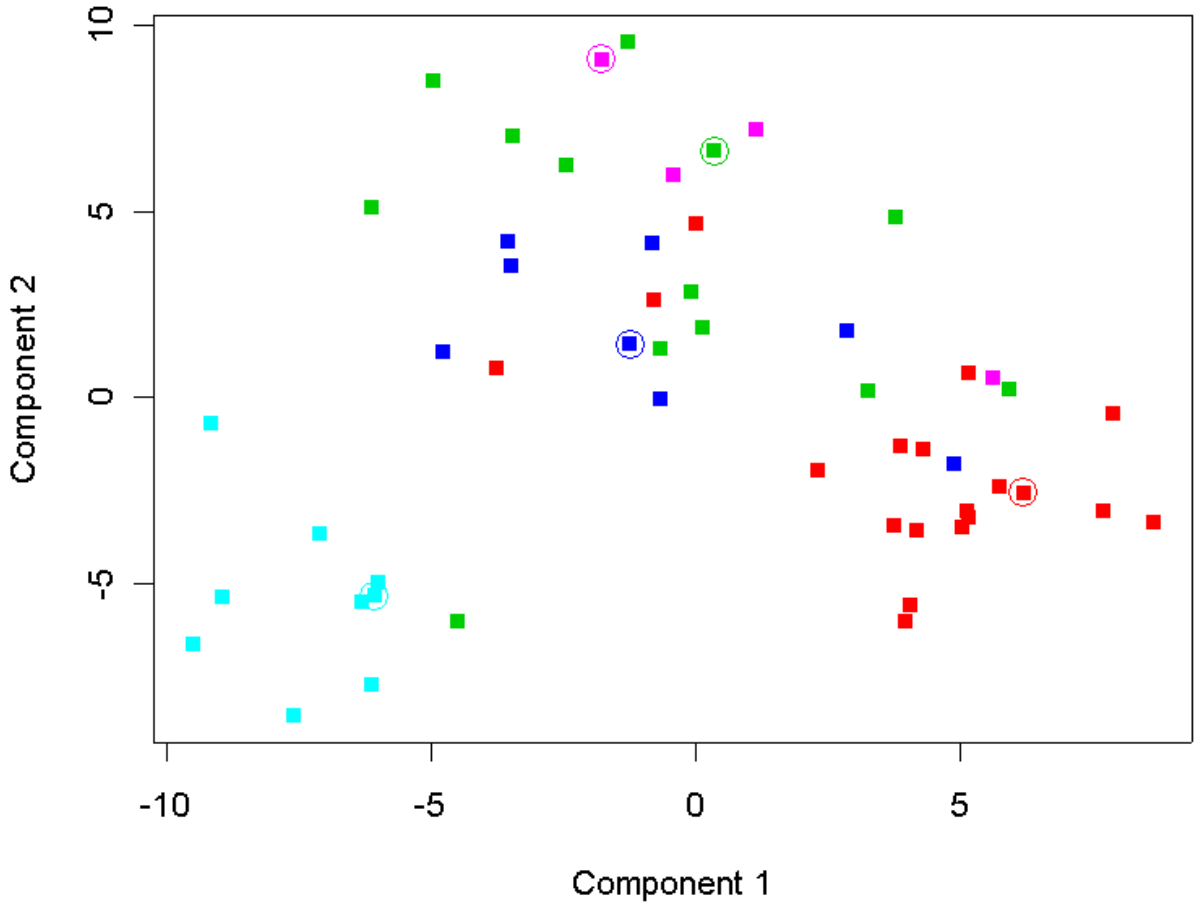
PAM in R

To use PAM in R, you must load another package:

```
> require(cluster)
> dist.matrix <- as.dist(1-cor(ldata)/2)
> pamres <- pam(dist.matrix, 5)
```

Unlike `kmeans`, the implementation of `pam` only lets you specify the number of clusters you want, not the starting point. It also apparently always uses the same method to choose the starting point, so it does not help to run the algorithm multiple times. If their heuristic chooses a poor starting configuration, there is no way to fix it.

PAM results



Not very good on our example data...

How many clusters are there?

Both `kmeans` and `pam` require you to specify the number of clusters before running the algorithm. In our example, we knew before we stated that there were five clusters. In real life, we rarely (if ever) know the number of real clusters before we start. How do we figure out the correct number of clusters?

One way is to run the algorithm with different values of K, and then try to decide which methods gives the best results. The problem that remains is how we measure "best".

Silhouette Widths

Kaufman and Rousseeuw (who wrote a book on clustering that describes pam along with quite a few other methods) recommend using the silhouette width as a measure of how much individual elements belong to the cluster where they are assigned. To compute the silhouette width of the i^{th} object, define

$$a(i) = \text{average distance to other elements in the cluster}$$

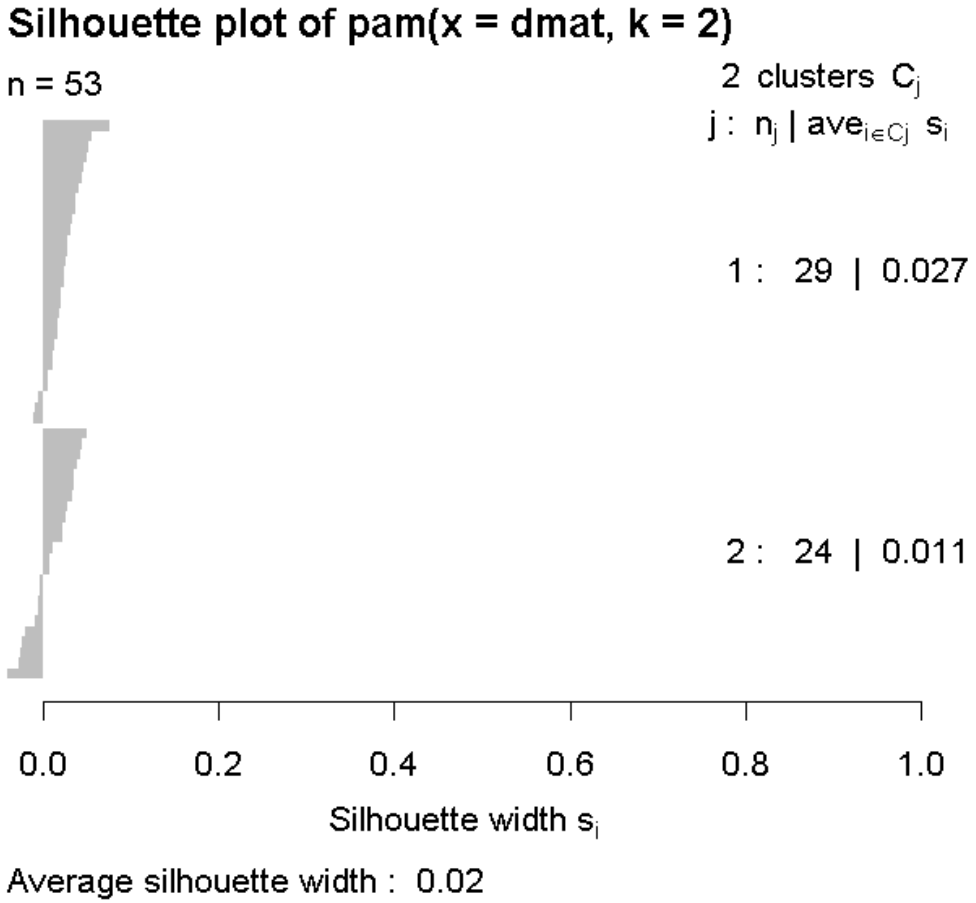
$$b(i) = \text{smallest average distance to other clusters}$$

$$sil(i) = (b(i) - a(i))/max(a(i), b(i)).$$

Interpretation: If $sil(i)$ is near 1, then the object is well clustered. If $sil(i) < 0$, then the object is probably in the wrong cluster. If $sil(i)$ is near 0, then it's on the border between two clusters.

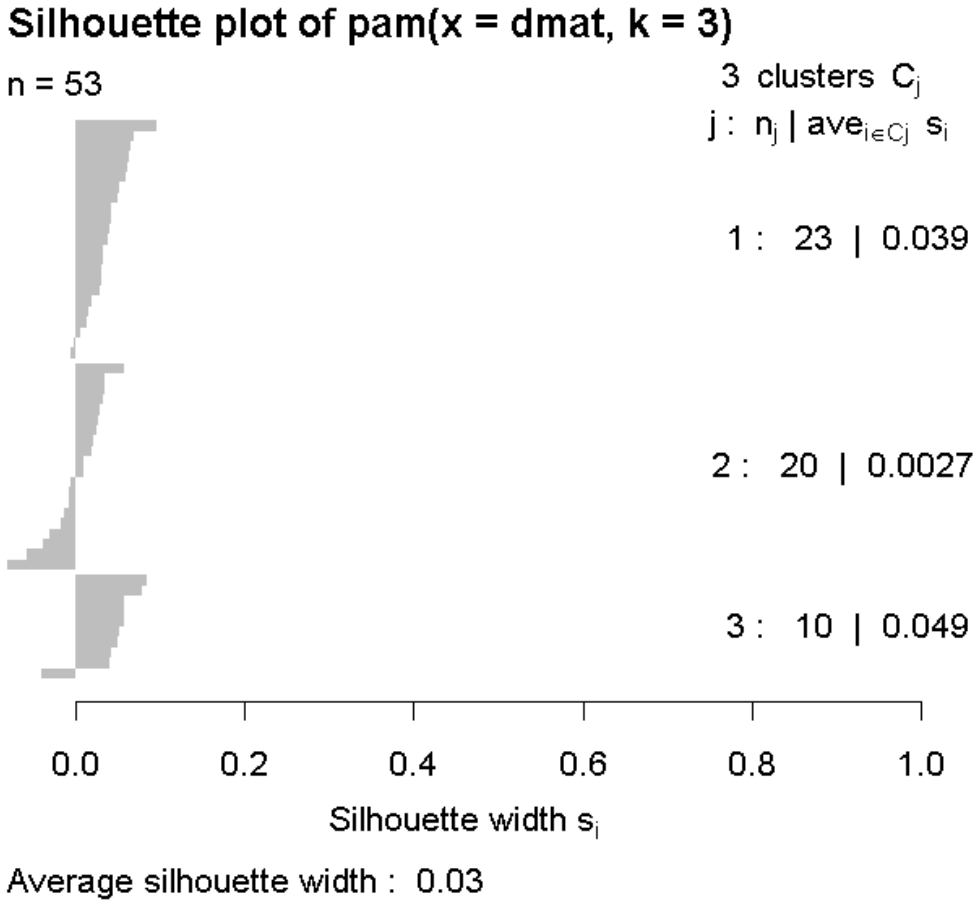
PAM : two clusters

```
> pam2 <- pam(dmat, 2)
> plot(pam2)
```



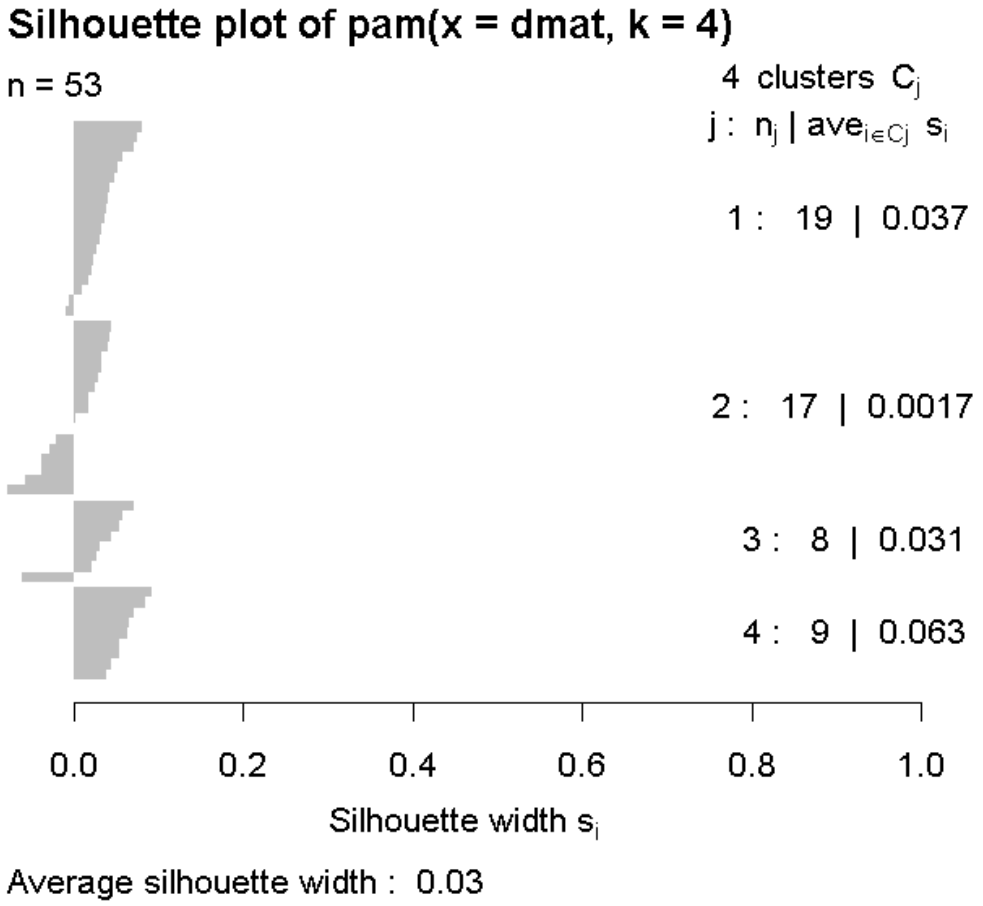
PAM : three clusters

```
> pam3 <- pam(dmat, 3)
> plot(pam3)
```



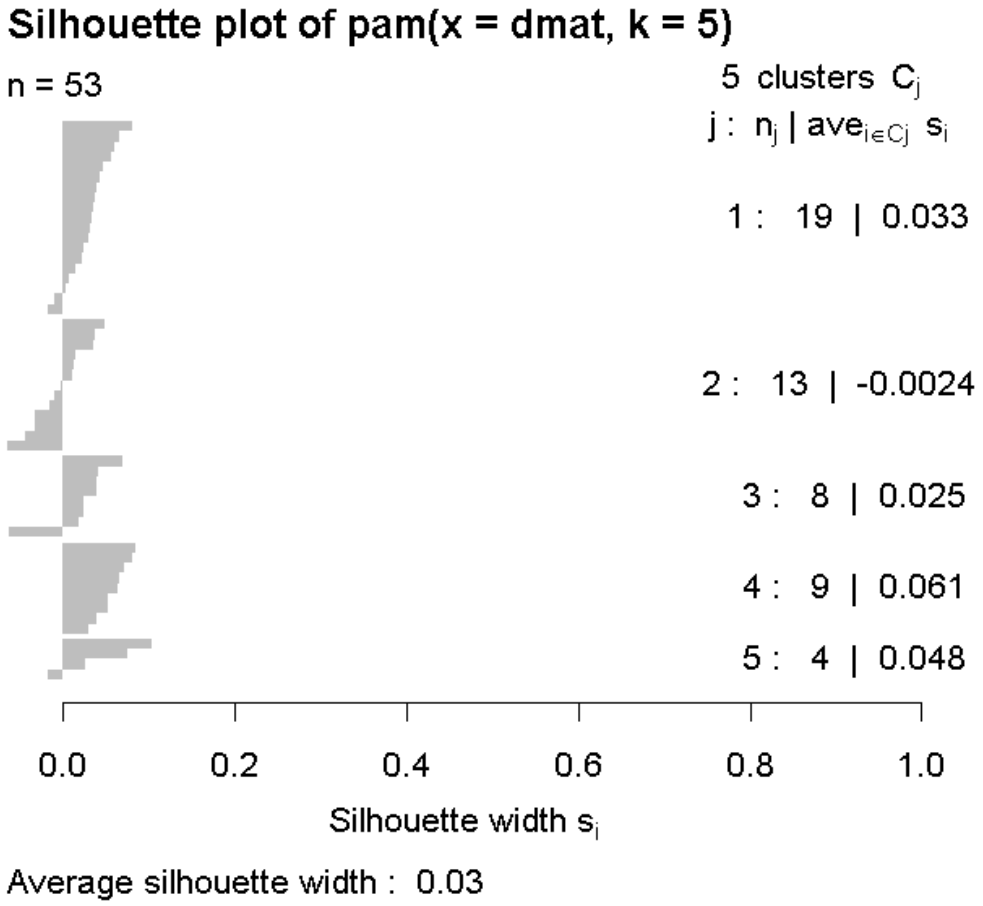
PAM : four clusters

```
> pam4 <- pam(dmat, 4)
> plot(pam4)
```



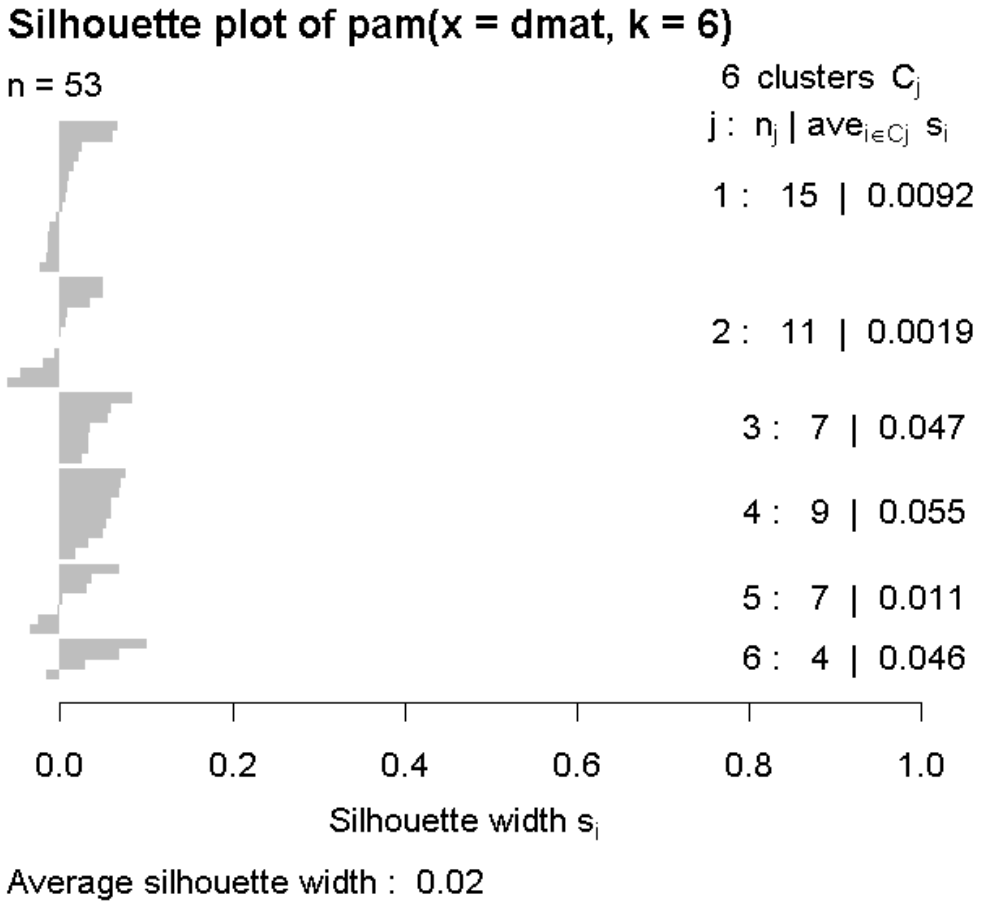
PAM : five clusters

```
> pam5 <- pam(dmat, 5)
> plot(pam5)
```



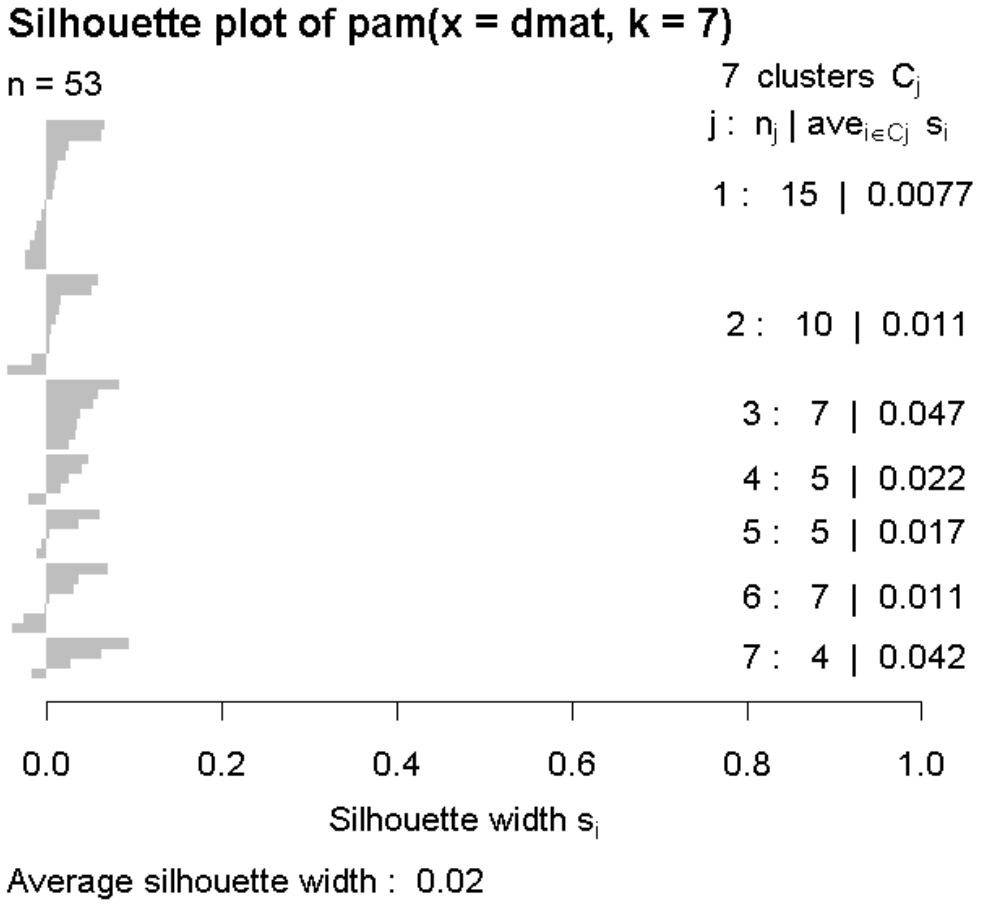
PAM : six clusters

```
> pam6 <- pam(dmat, 6)
> plot(pam6)
```



PAM : seven clusters

```
> pam7 <- pam(dmat, 7)
> plot(pam7)
```



```
> summary(silhouette(pam2))$avg.width
[1] 0.01938651
> summary(silhouette(pam3))$avg.width
[1] 0.02713564
> summary(silhouette(pam4))$avg.width
[1] 0.02904244
> summary(silhouette(pam5))$avg.width
[1] 0.02875473
> summary(silhouette(pam6))$avg.width
[1] 0.02342055
> summary(silhouette(pam7))$avg.width
[1] 0.01862886
```

In general, we want to choose the number of clusters that maximizes the average silhouette width. In this case, that means 4 or 5 clusters.

Using silhouettes with K-means

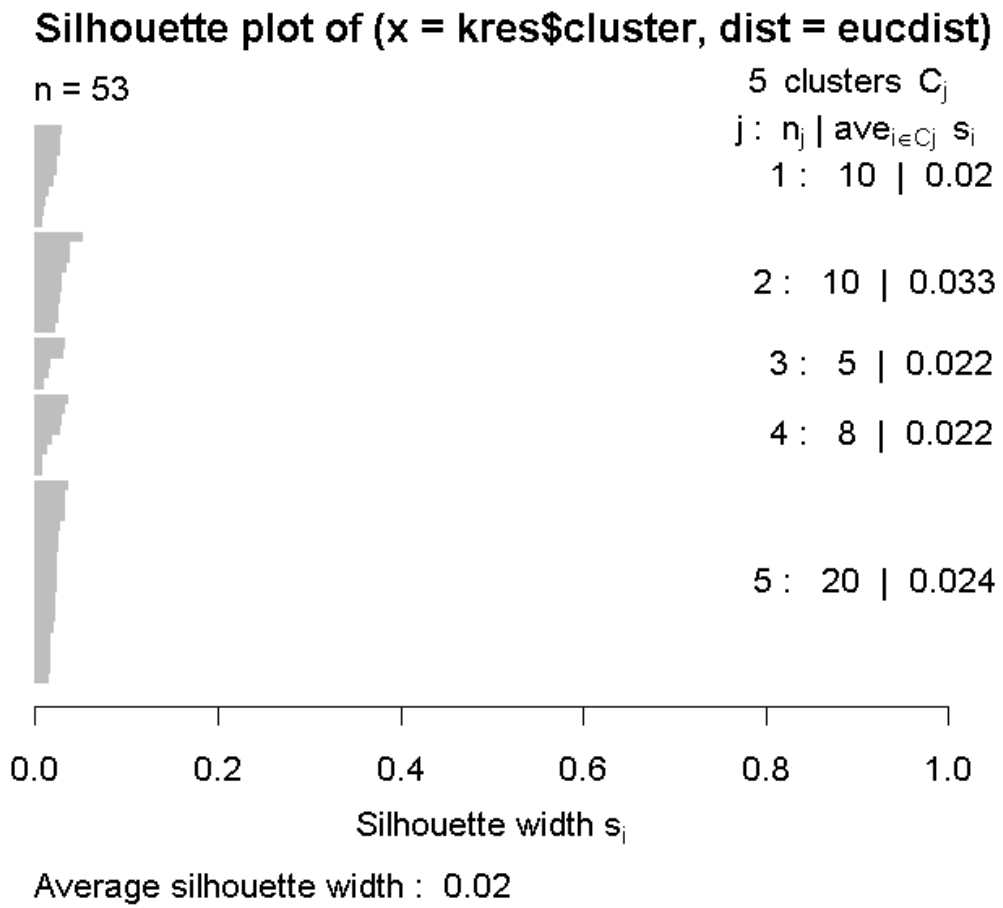
The `silhouette` function knows about `pam` objects, making it relatively easy to use. You can use it with other clustering routines, but you have to supply the clustering vector and the distance matrix. For example, here are the results for the best K-means clustering that we found (as measured by the within-cluster sum of squares).

```
> euc.distance <- dist(t(ldata))
> ksil <- silhouette(kres$cluster, euc.distance)
> summary(ksil)$avg.width
[1] 0.02453796
```

Note that the silhouette width is smaller than the one from PAM using correlation. However, the silhouette plot suggests that everything is classified correctly:

K-means: five cluster silhouette

```
> plot(ksil)
```



Silhouettes with hclust

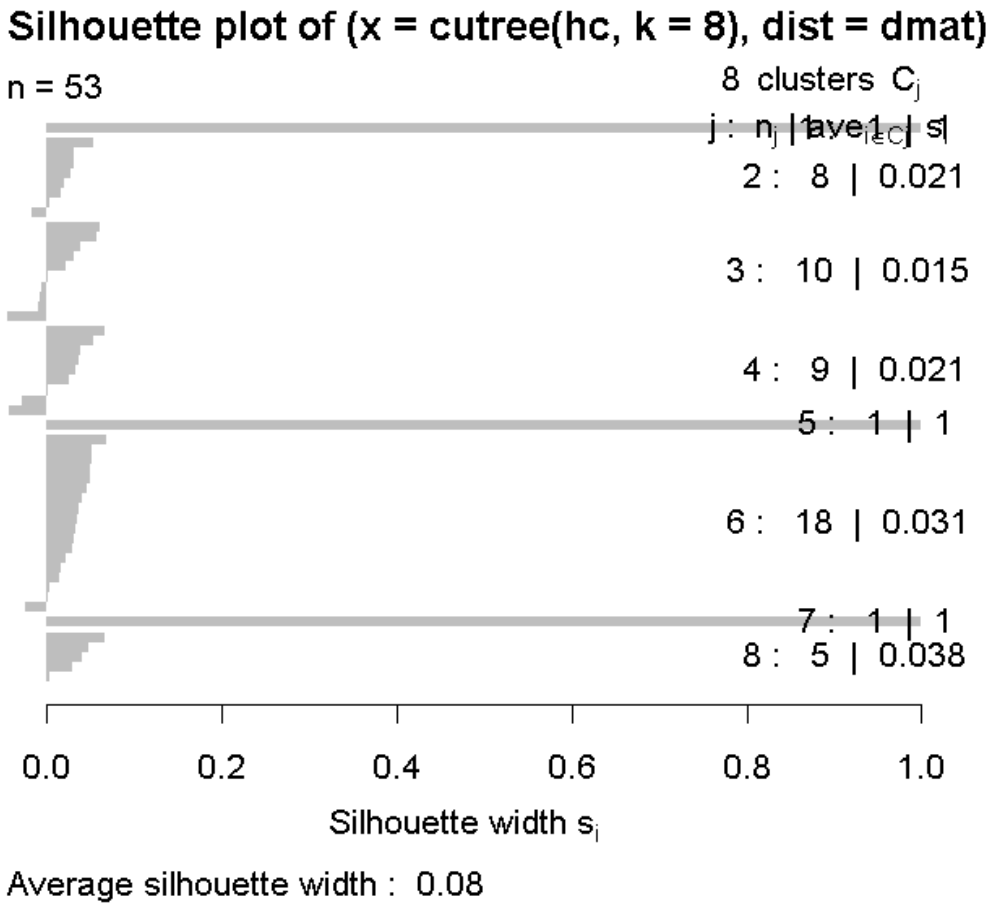
Looking back at the hierarchical clustering, we need to cut eight branches off (including some singletons) to get down to the clusters that look real.

```
> dmat <- as.dist((1-cor(ldata))/2)
> hc <- hclust(dmat, 'average')
> hsil <- silhouette(cutree(hc, k=8), dmat)
> summary(hsil)$avg.width
[1] 0.08024319
```

Why does the average silhouette width look so much better for this method?

Hierarchical clustering: silhouette with singletons

```
> plot(hsil)
```



The silhouette width

General conclusions: The average silhouette width is only a crude guide to the number of clusters present in the data. The silhouette plot seems to be more useful, but requires human intervention (in the form of “cortical filtering”).

The Gap statistic

An alternative method for determining the number of clusters relies on the gap statistic. The idea is to run a clustering algorithm for different values of the number K of clusters. Let $W(K)$ be the within-cluster error. In the case of Euclidean distance, $W(K)$ is just the within-cluster sum of squares. For other distance measures, it is the term we minimized in the description of PAM. Because adding more clusters will always reduce this error term, $W(K)$ is a decreasing function of K. However, it should decrease faster when K is less than the true number and slower when K is greater than the true number.

The gap statistic measures the difference (on the log scale, for each K) between the observed $W(K)$ and the expected value if the data were uniformly distributed. One then selects the K with the largest gap between the observed and expected values.

Principal Components

You may have wondered how we produced two-dimensional plots of the simulated data that involved 1000 genes and 53 samples. The short answer is: we used principal components analysis (PCA).

As we have been doing throughout our discussion of clustering, we view each sample as a vector $x = (x_1, \ldots, x_G)$ in G-dimensional "gene space". The idea behind PCA is to look for a direction (represented as a linear combination $u_1 = \sum_{i=1}^{G} w_i x_i$) that maximizes the variability across the samples. Next, we find a second direction u_2 at right angles to the first that maximizes what remains of the variability. We keep repeating this process. The u_i vectors are the principal components, and we can rewrite each sample vector as a sum of principal components instead of as a sum of separate gene expression values.

Data reduction

PCA can be used as a data reduction method. Changing from the original x-coordinates to the new u-coordinate system doesn't change the underlying structure of the sample vectors. However, it does let us focus on the directions where the data changes most rapidly. If we just use the first two or three principal components, we can produce plots that show us as much of the intrinsic variability in the data as possible.

Warning: even though there is a function in R called `princomp` that is supposed to compute principal components, it will not work in the context of microarrays. The problem is that `princomp` wants to decompose the covariance matrix, which is a square matrix with size given by the number of genes. That's simply too big to manipulate.

Sample PCA

We have written a version of PCA in R using Singular Value Decomposition. The first plot of our simulated data was produced using the following commands:

```
> spca <- SamplePCA(ldata)
> plot(spca, split=group.factor)
```

The plots that added the X's to mark the K-means centers were produced with:

```
> plot(spca, split=factor(kres$cluster))
> x1 <- spca@scores[kcent,1] # start circles
> x2 <- spca@scores[kcent,2]
> points(x1, x2, col=2:6, pch=1, cex=2)
> pcak <- predict(spca, t(kres$centers)) # finish X
> points(pcak[,1], pcak[,2], col=2:6, pch=4)
```

Principal Coordinates

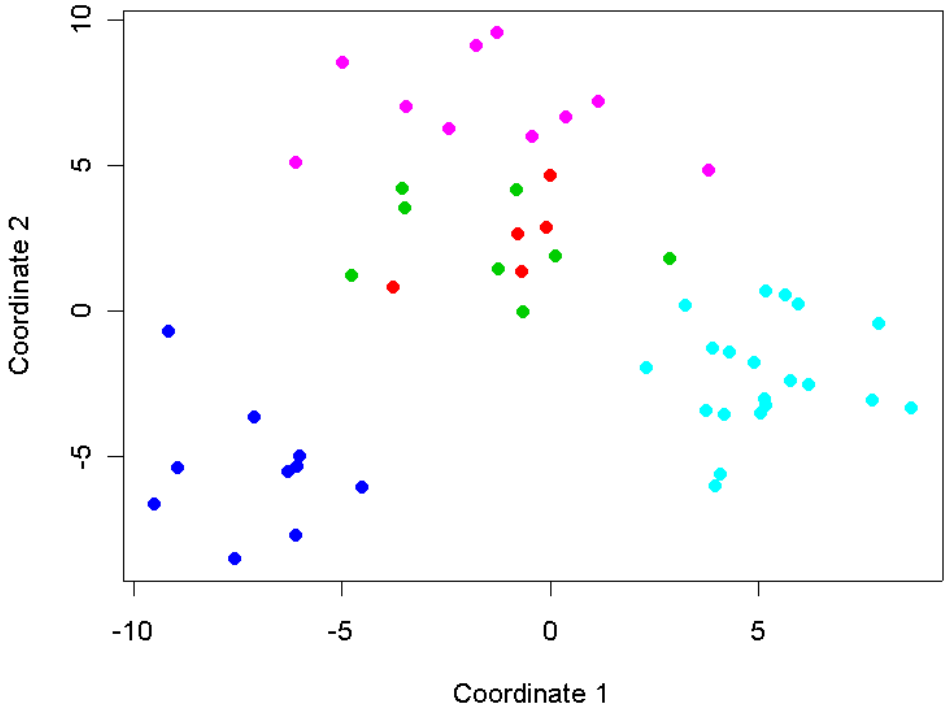
Using the first few principal components provides a view of the data that allows us to see as much of the variability as possible. Sometimes we have a different goal: we'd like to be able to visualize the samples in a way that does as good a job as possible of preserving the distances between samples. In general, this method is called multidimensional scaling (MDS).

The classical form of MDS is also known as principal coordinate analysis, and is implemented in R by the function `cmdscale`.

If you look at the resulting graph carefully, you'll discover that it is identical to the principal components plot: classical MDS using Euclidean distance is equivalent to plotting the first few principal components.

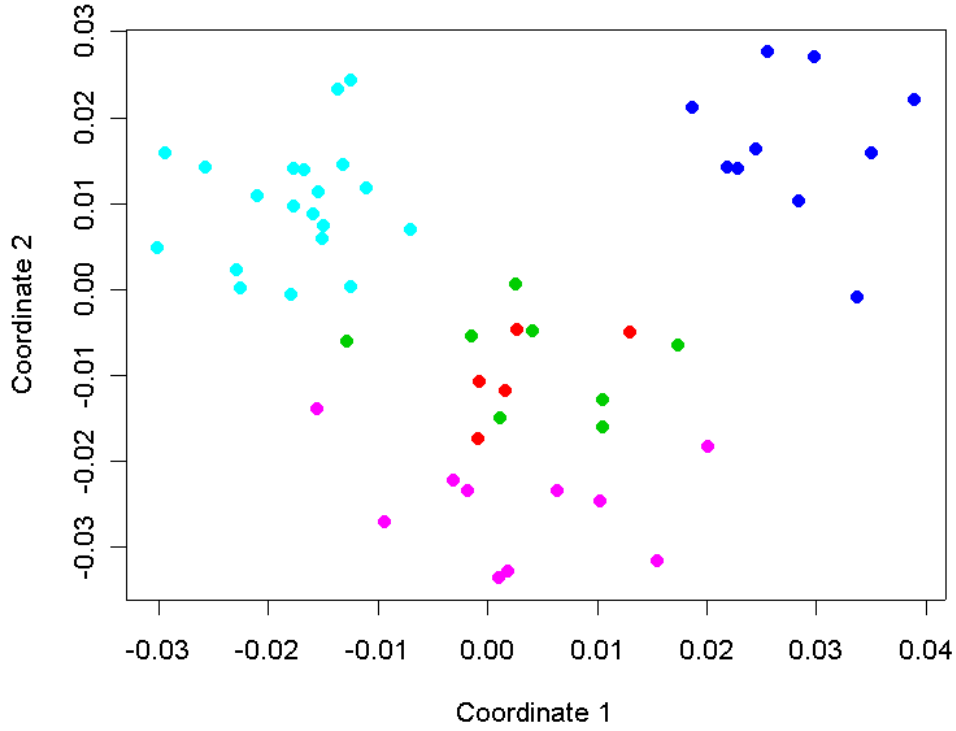
Euclidean classical MDS = PCA

```
> euloc <- cmdscale(euc.distance)
> plot(euloc[,1], euloc[,2], pch=16,
+      col=as.numeric(group.factor),
+      xlab='Coordinate 1', ylab='Coordinate 2')
```



Classical MDS with correlation

```
> loc <- cmdscale(dmat)
> plot(loc[,1], loc[,2], pch=16,
+      col=1+as.numeric(group.factor),
+      xlab='Coordinate 1', ylab='Coordinate 2')
```

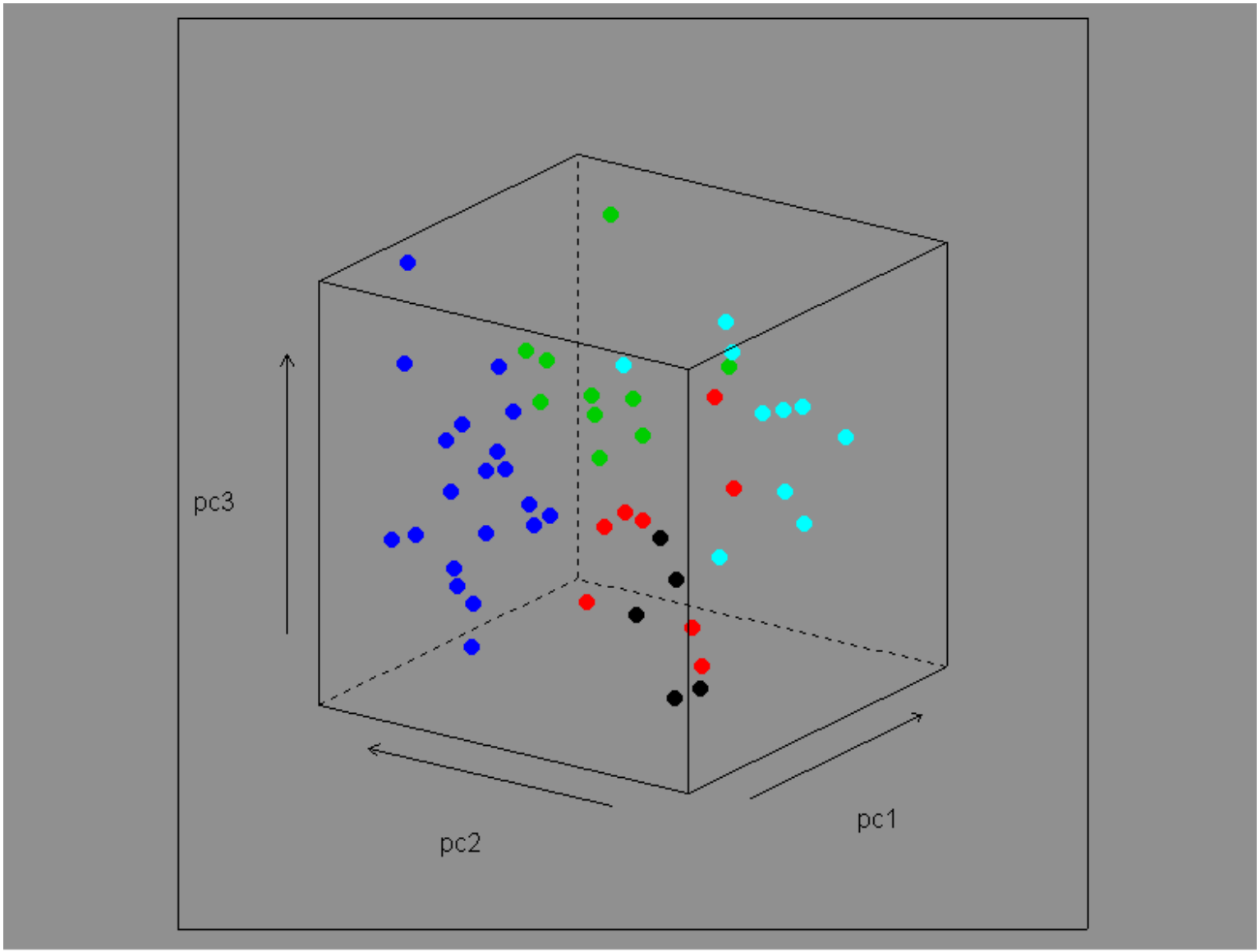


Classical MDS with correlation

Using the `cloud` function in the `lattice` package, we can also plot the first three principal coordinates:

```
> require(lattice)
> loc3d <- cmdscale(dmat, k=3)
> pc1 <- loc3d[,1]
> pc2 <- loc3d[,2]
>  pc3 <- loc3d[,3]
> cloud(pc3 ~ pc1*pc2, pch=16, cex=1.2,
+       col=1+as.numeric(group.factor),
+        screen=list(z=55, x=-70),
+       perspective=FALSE)
```

Three-D



Project Normal

- Eighteen samples
  - Six C57BL6 male mice
  - Three organs: kidney, liver, testis
- Reference material
  - Pool RNA from all eighteen mouse organs
- Replicate experiments on two-color arrays with common reference
  - Four experiments per mouse organ
  - Dye swaps: two red samples, two green samples

Original analysis of Project Normal

Reference: Pritchard, Hsu, Delrow, and Nelson. (2001) *Project normal: defining normal variance in mouse gene expression*. PNAS **98**: 13266–13271.

- Print-tip specific intensity dependent loess normalization
- Scale adjusted (using MAD)
- Work with log ratios (experimental/reference)
- Perform F-test for each gene to see if mouse-to-mouse variance exceeds the array-to-array variance.

First steps

We chose to process the data using a simple global normalization (to the 75^{th} percentile) instead of loess normalization, since we believed that the mixed reference RNA should have a different distribution of intensities than RNA from a single organ. We then transformed the intensities in each channel by computing their base-two logarithm.

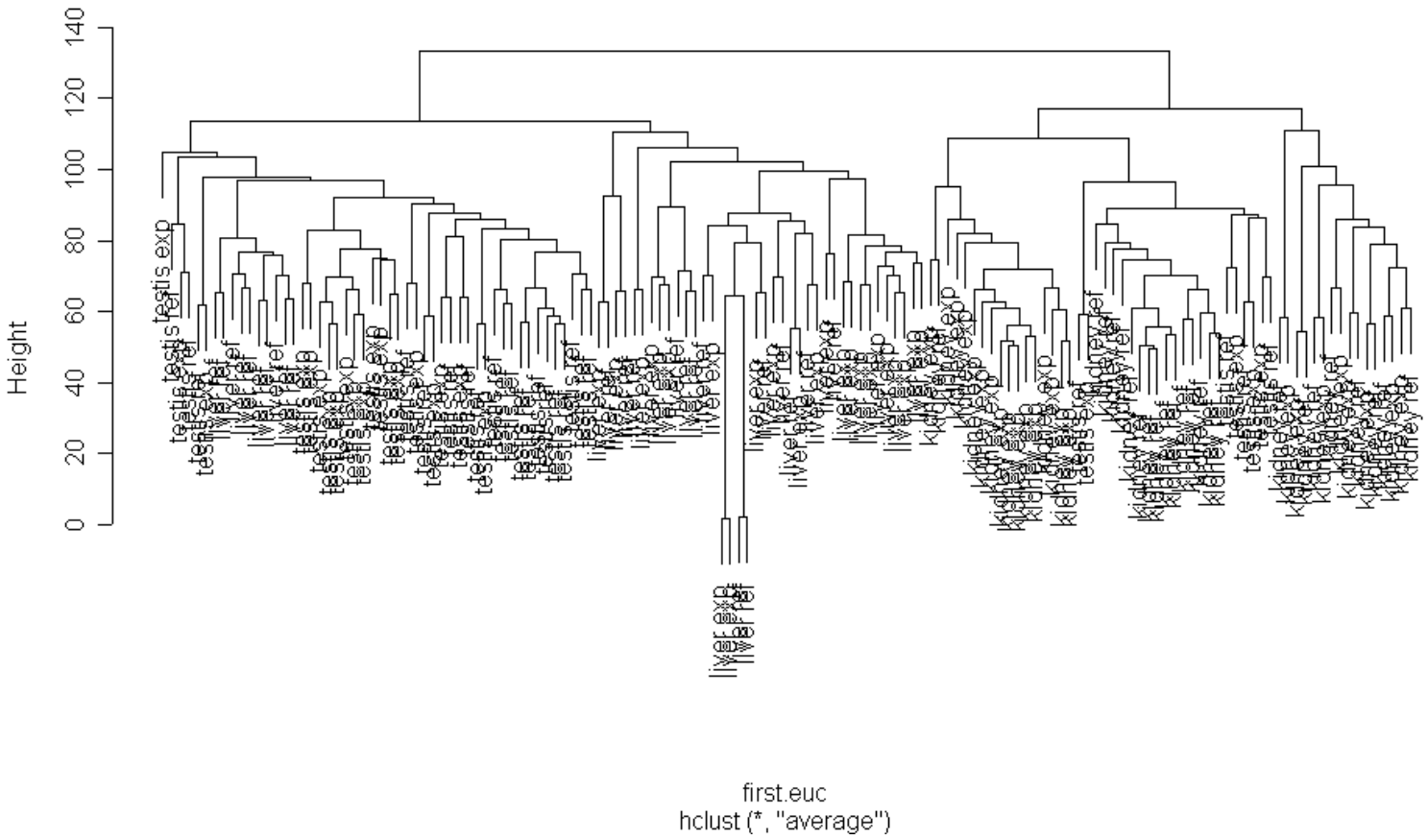
Main Question: Can we determine from the project normal data set which genes are specifically expressed in each of the three organs?

Clustering Methods

If we cluster the data, what should we expect to see? Which clustering method would be most appropriate for a first look at the data?

- Hierarchical clustering
- Partitioning around medoids
- K-means
- Multidimensional scaling
- Principal components analysis

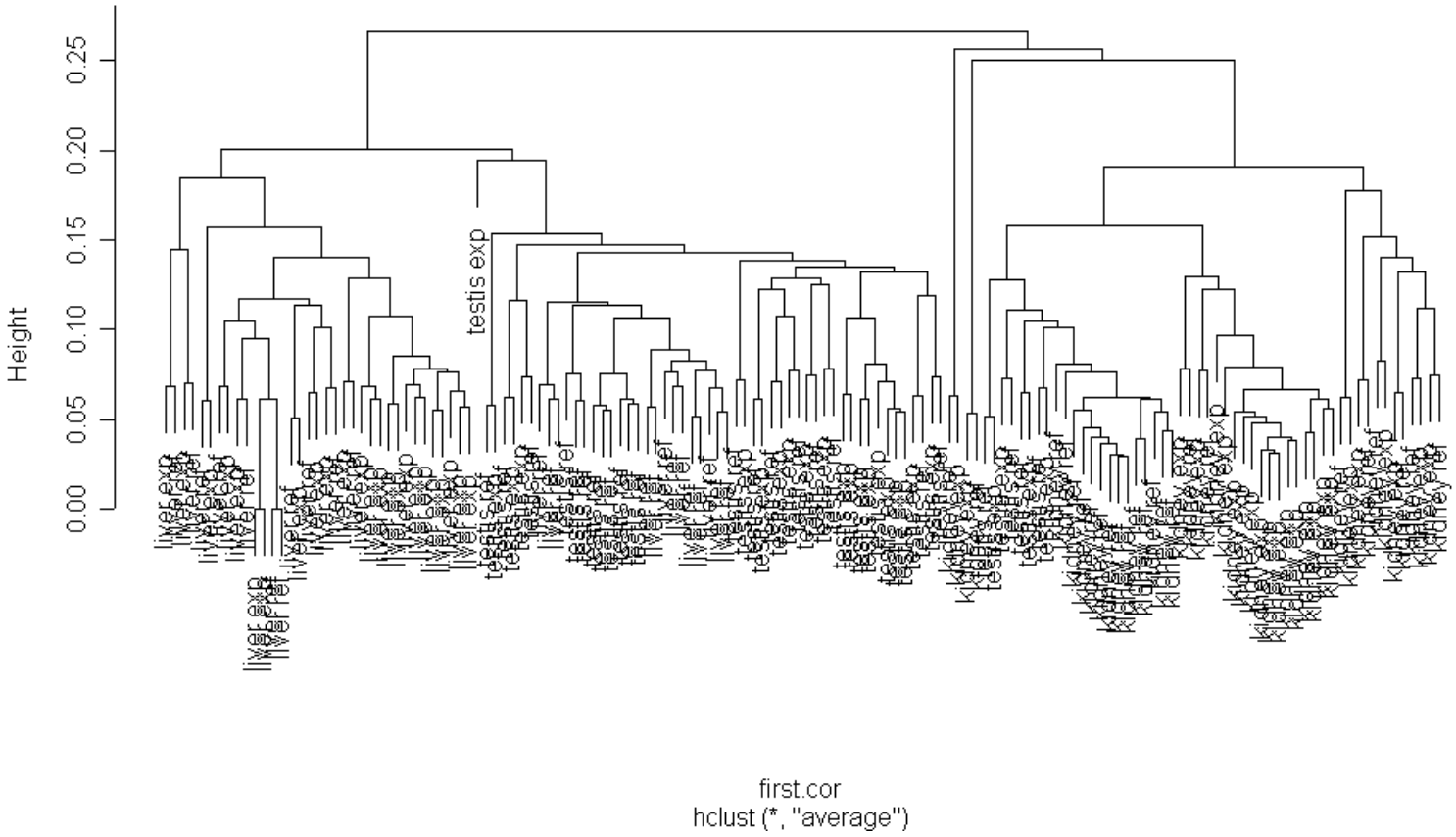
Hierarchical clustering



Euclidean distance, average linkage

Back to clustering methods

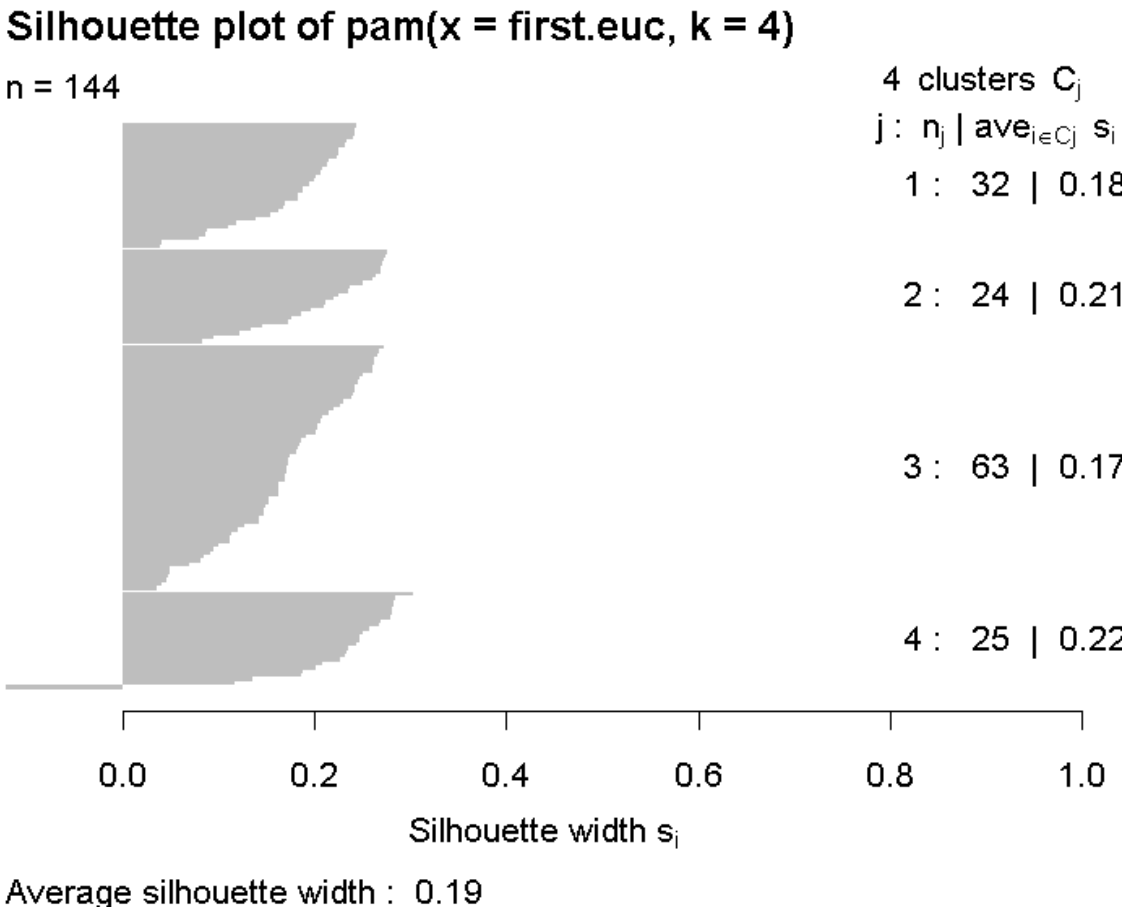
Hierarchical clustering



Correlation distance, average linkage

Back to clustering methods

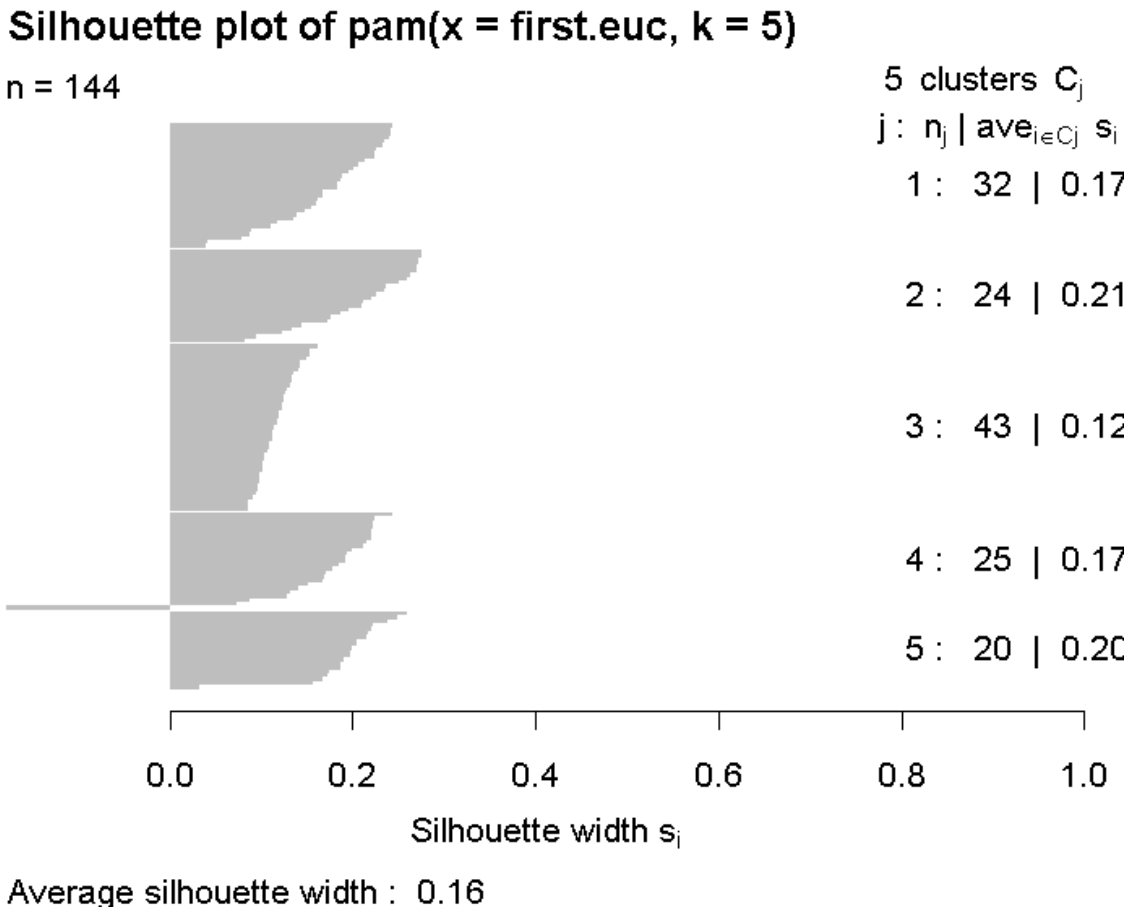
Partitioning Around Medoids



Euclidean distance, four clusters

Back to clustering methods

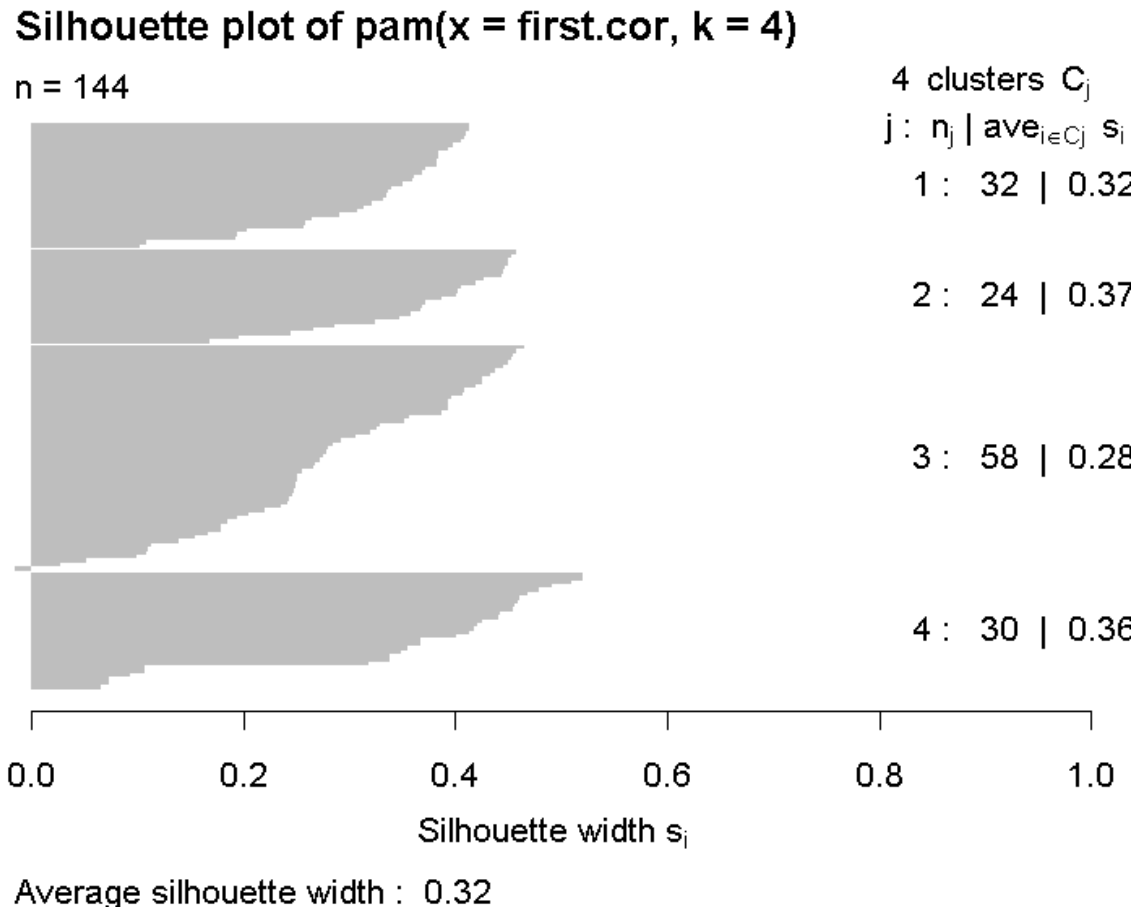
Partitioning Around Medoids



Euclidean distance, five clusters

Back to clustering methods

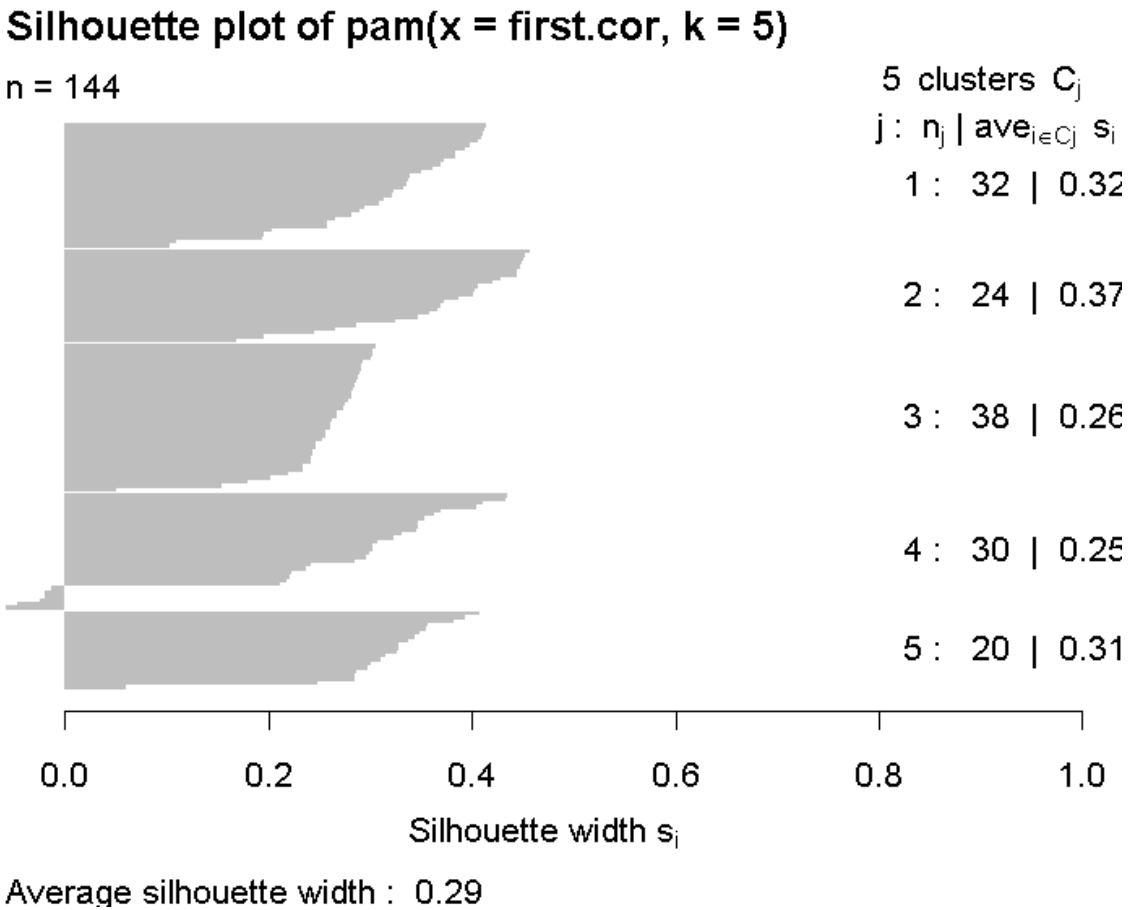
Partitioning Around Medoids



Correlation distance, four clusters

Back to clustering methods

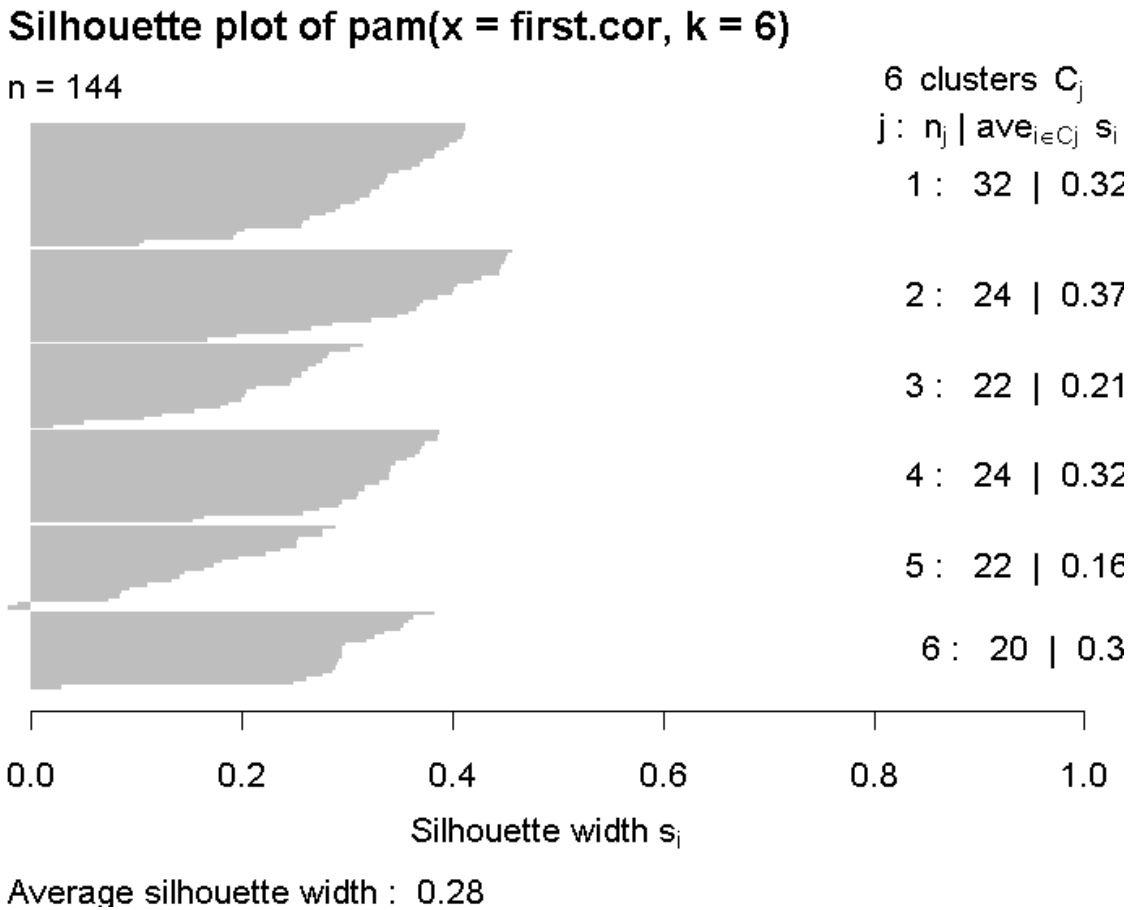
Partitioning Around Medoids



Correlation distance, five clusters

Back to clustering methods

Partitioning Around Medoids



Correlation distance, six clusters

Back to clustering methods

K-means

Number of channels in each cluster:

Channel	C1	C2	C3	C4
Experiment	4	24	20	24
Reference	28	0	44	0

Organ	C1	C2	C3	C4
Kidney	24	24	0	0
Liver	0	0	24	24
Testis	8	0	40	0

Best of 50 runs with four clusters

Back to clustering methods

K-means

Number of channels in each cluster:

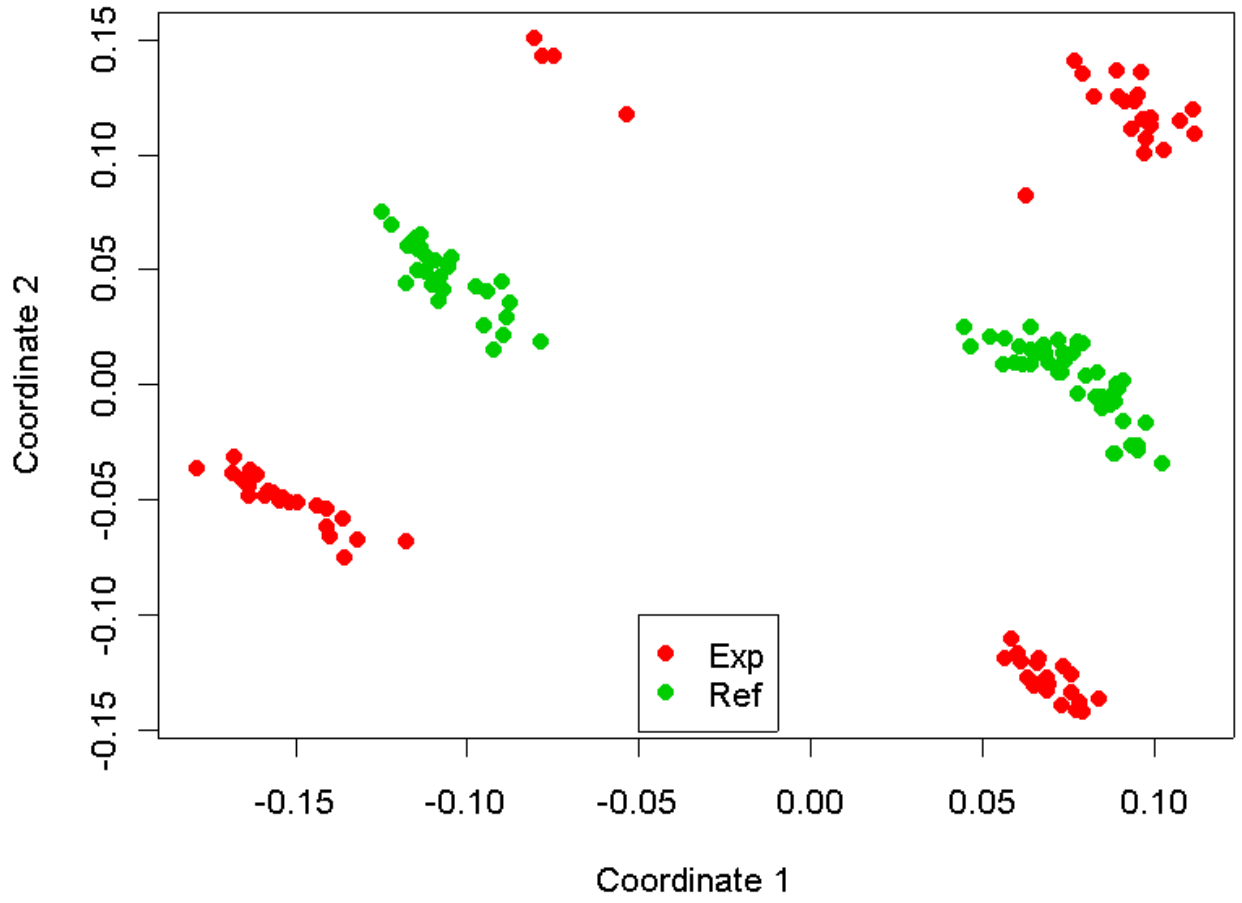
Channel	C1	C2	C3	C4	C5
Experiment	4	16	20	8	24
Reference	20	0	44	8	0

Organ	C1	C2	C3	C4	C5
Kidney	16	16	0	16	0
Liver	0	0	24	0	24
Testis	8	0	40	0	0

Best of 50 runs with five clusters

Back to clustering methods

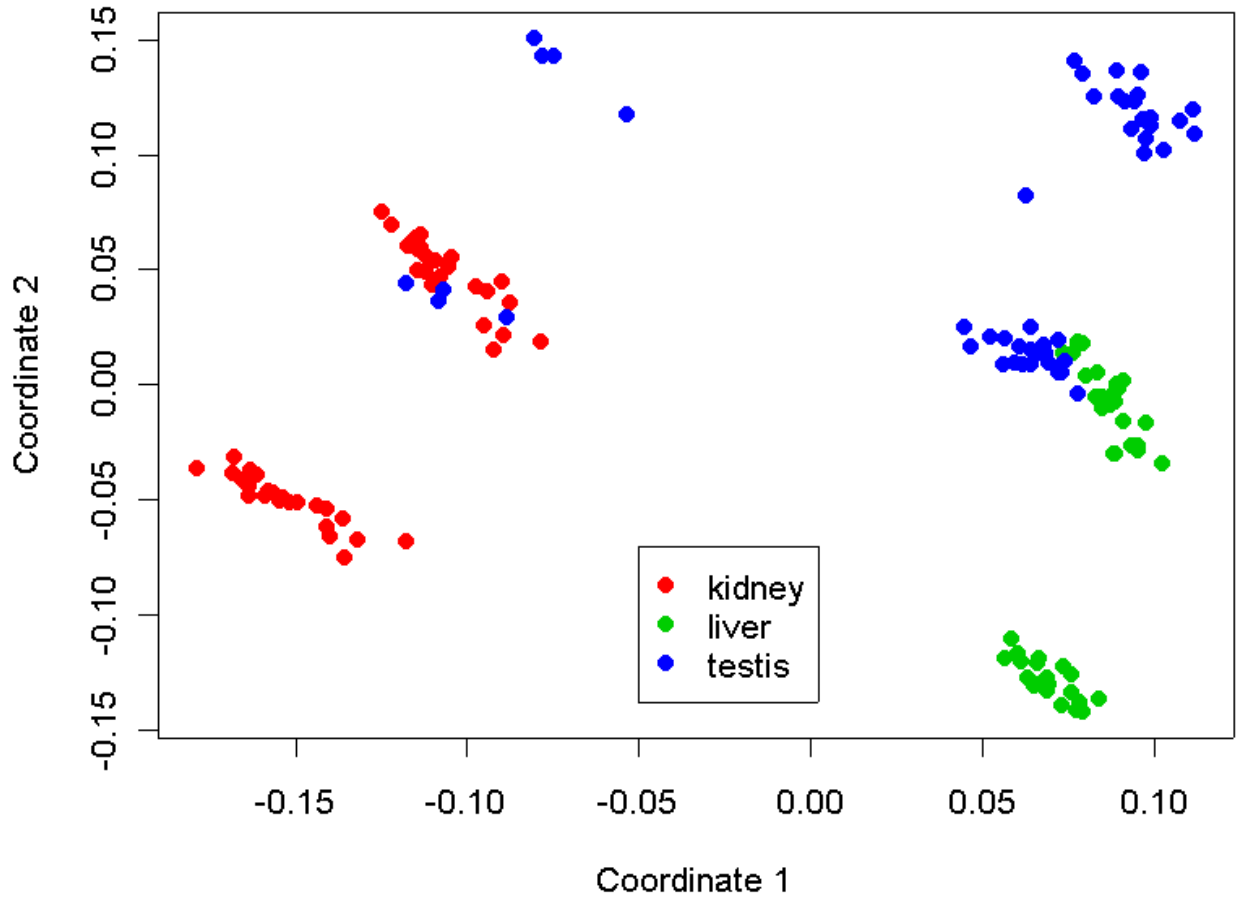
Multidimensional Scaling



Correlation distance, colored to indicate channel

Back to clustering methods

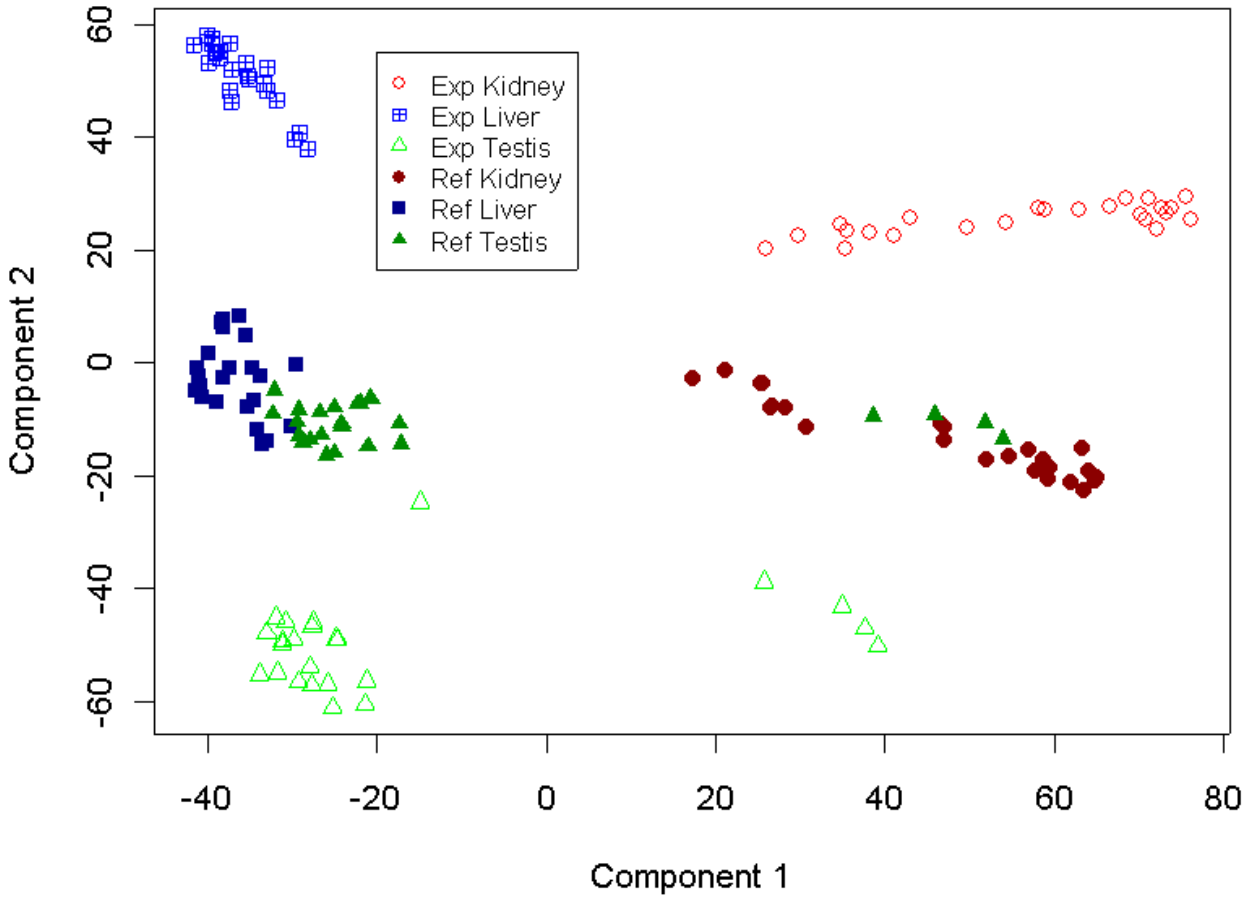
Multidimensional Scaling



Correlation distance, colored to indicate organ

Back to clustering methods

Principal components analysis



Euclidean distance, indicating channel and organ.

Back to clustering methods Forward to second PCA

Abnormal Behavior

Regardless of which exploratory method we use to look at the data, we see that something strange is happening here.

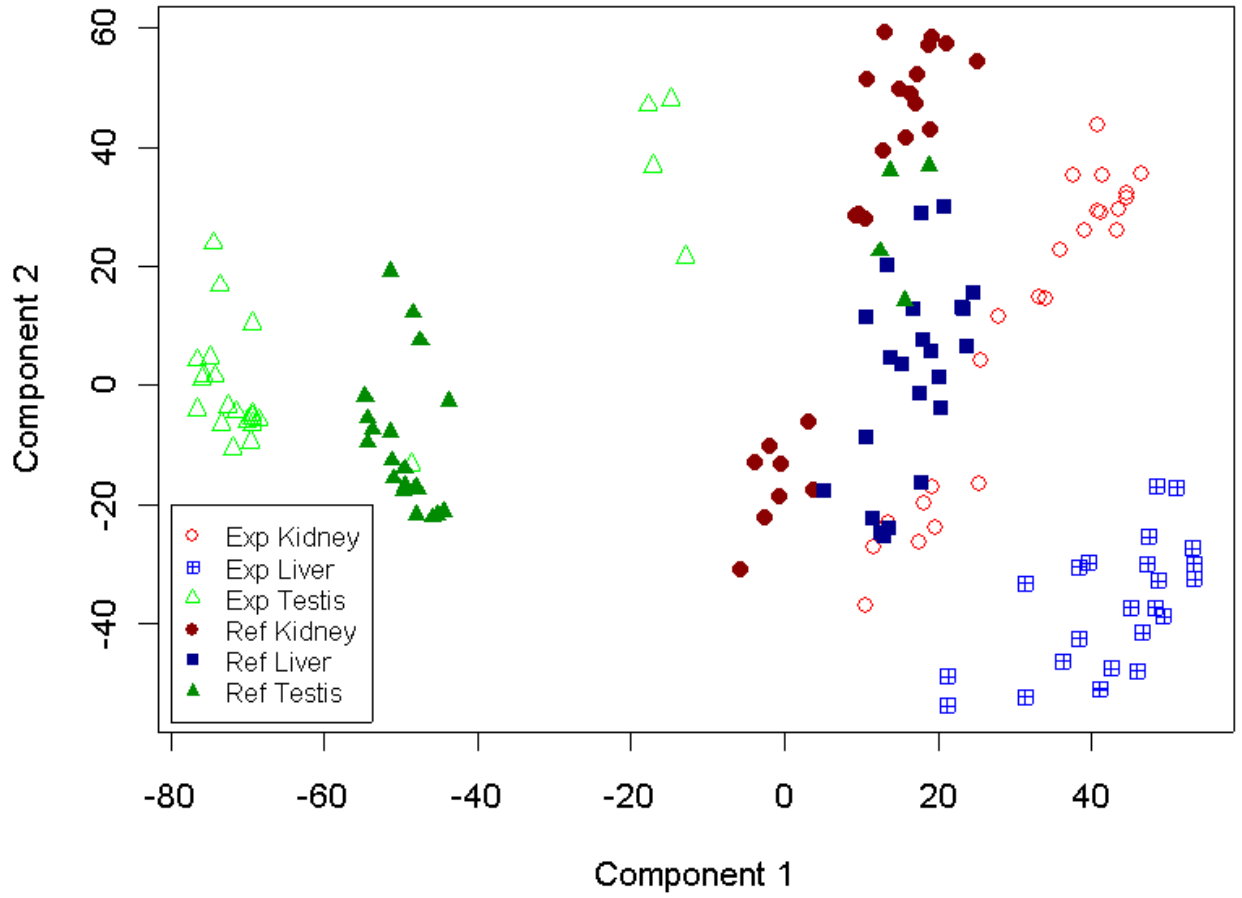
We might not have noticed this behavior if we had immediately gone to the log ratios instead of clustering the separate channels.

What might explain the presence of two different kinds of reference channels? First thought: dye swaps. But this doesn't make sense, since then we would expect the experimental channels to split the same way (giving us eight clusters in total).

Data merging

- Data was supplied in three files, one each for kidney, liver and testis.
- Each row in each file contained two kinds of annotations:
  1. Location (block, row, and column)
  2. Genetic material (IMAGE clone, UniGene ID)
- For our analysis, we merged the data using the annotations of genetic material.
- As it turns out, the locations did not agree
- So, we reordered data rows and merged on location...

PCA after merging data on location



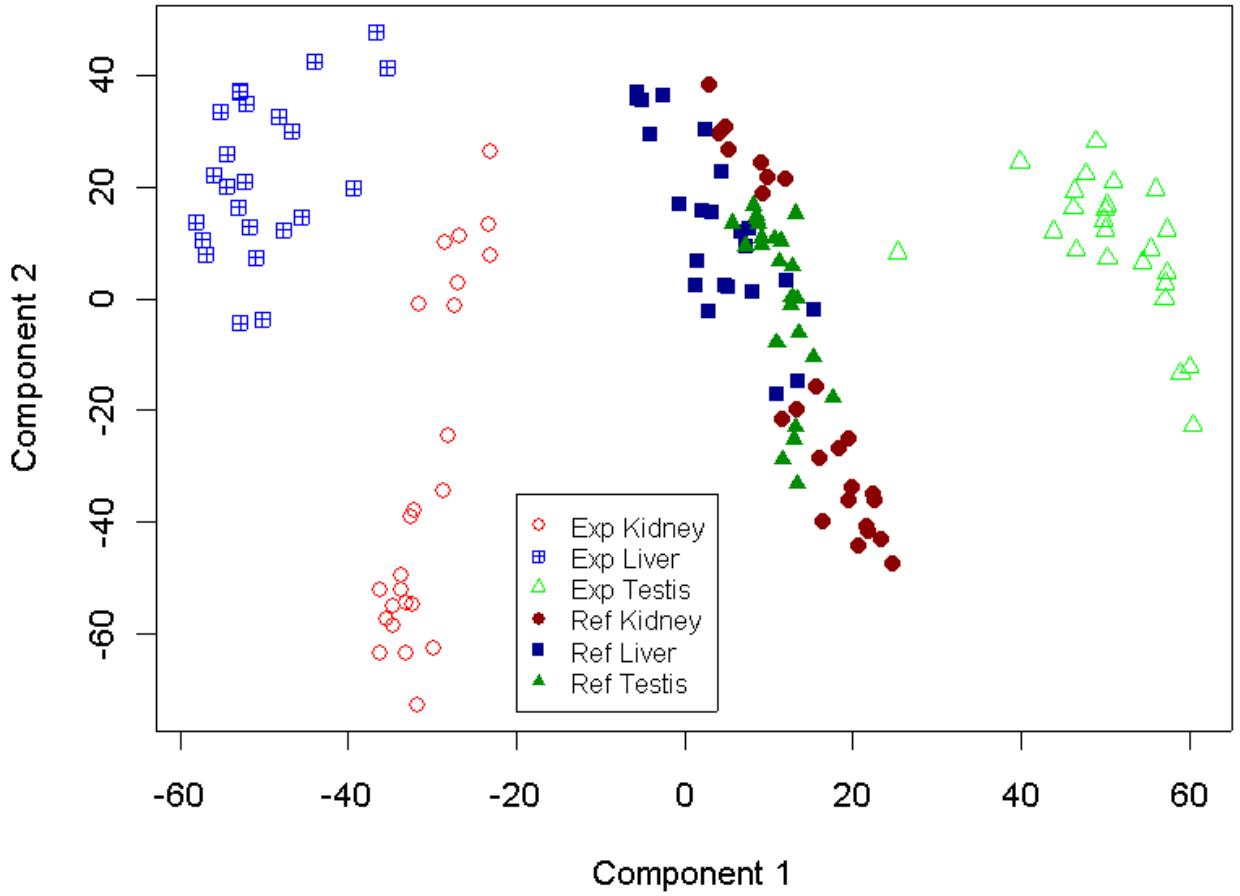
Yuck. So why are most of the testis references so weird?

Back to first PCA Forward to third PCA

Inspired guessing

- When the gene annotations are matched
  - Four of the testis reference channels are close to the kidney reference
  - Twenty of the testis reference are close to the liver reference
- When the location annotations are matched
  - Kidney, liver, and 4 testis references are close
  - The other 20 testis reference are off by themselves
- Conclusion: A data processing error occurred partway through the testis experiments.

Principal components, take 3



Finally, the picture we expected to start with!

Back to second PCA

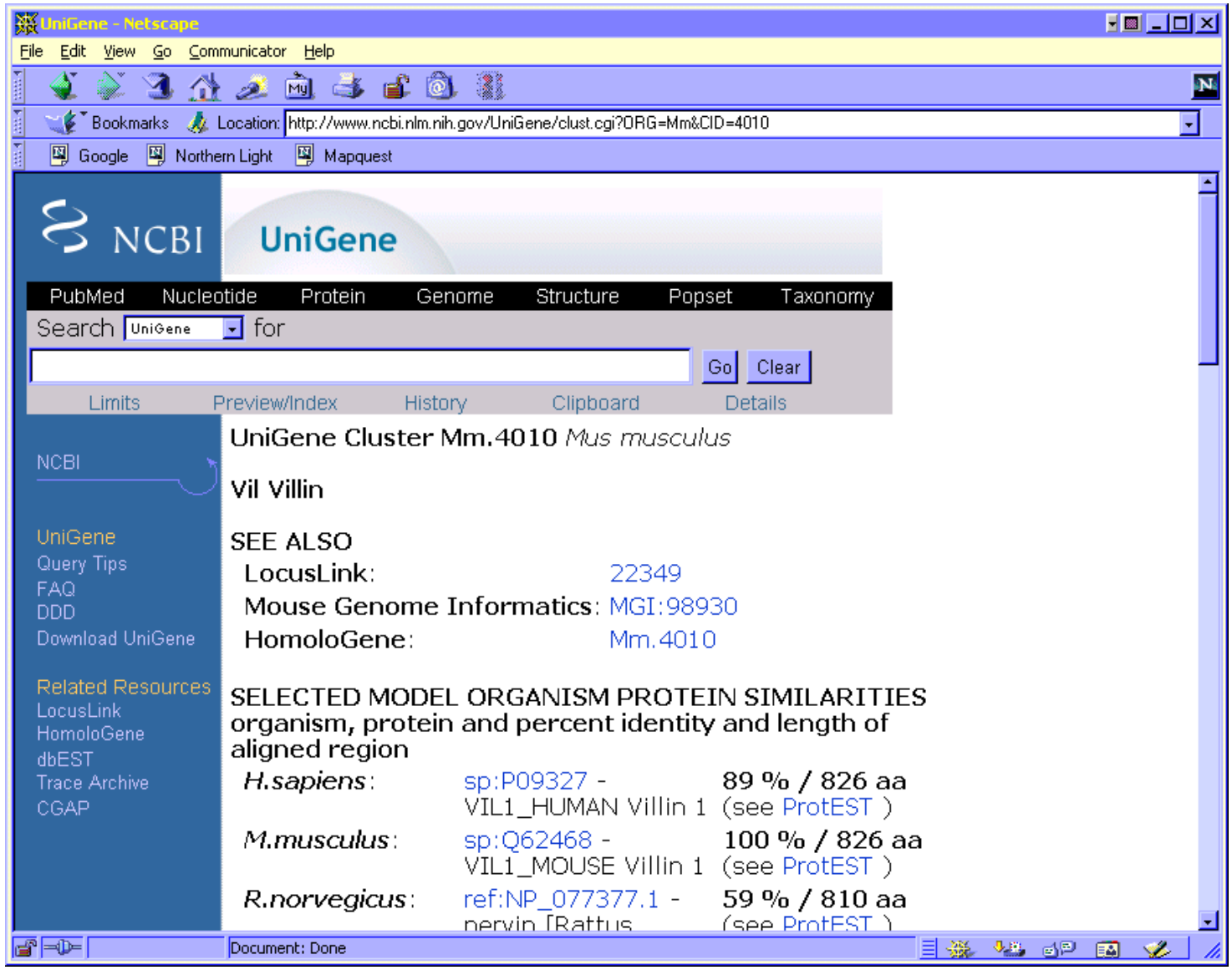
Every solution creates a new problem

- Solution: After reordering all liver experiments and twenty testis experiments by location
  - Can distinguish between the three organs
  - The reference samples all cluster together
- New Problem: There are now two competing ways to map from locations to genetic annotations (one from the kidney data, one from the liver data). Which one is correct?

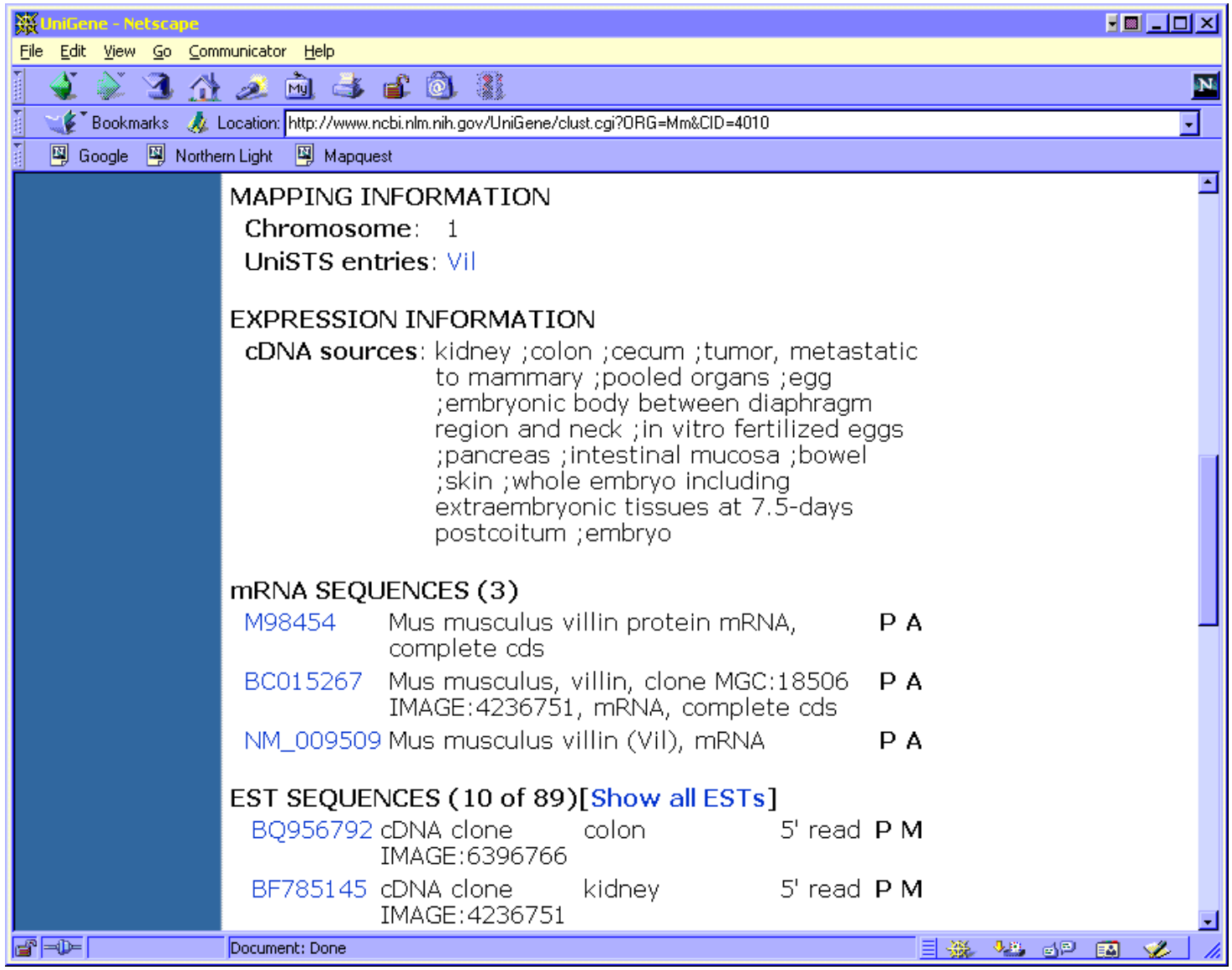
How big is the problem?

- Microarray contains 5304 spots
- Only 3372 (63.6%) spots have UniGene annotations that are consistent across the files
- So, 1932 (36.4%) spots have ambiguous UniGene annotations

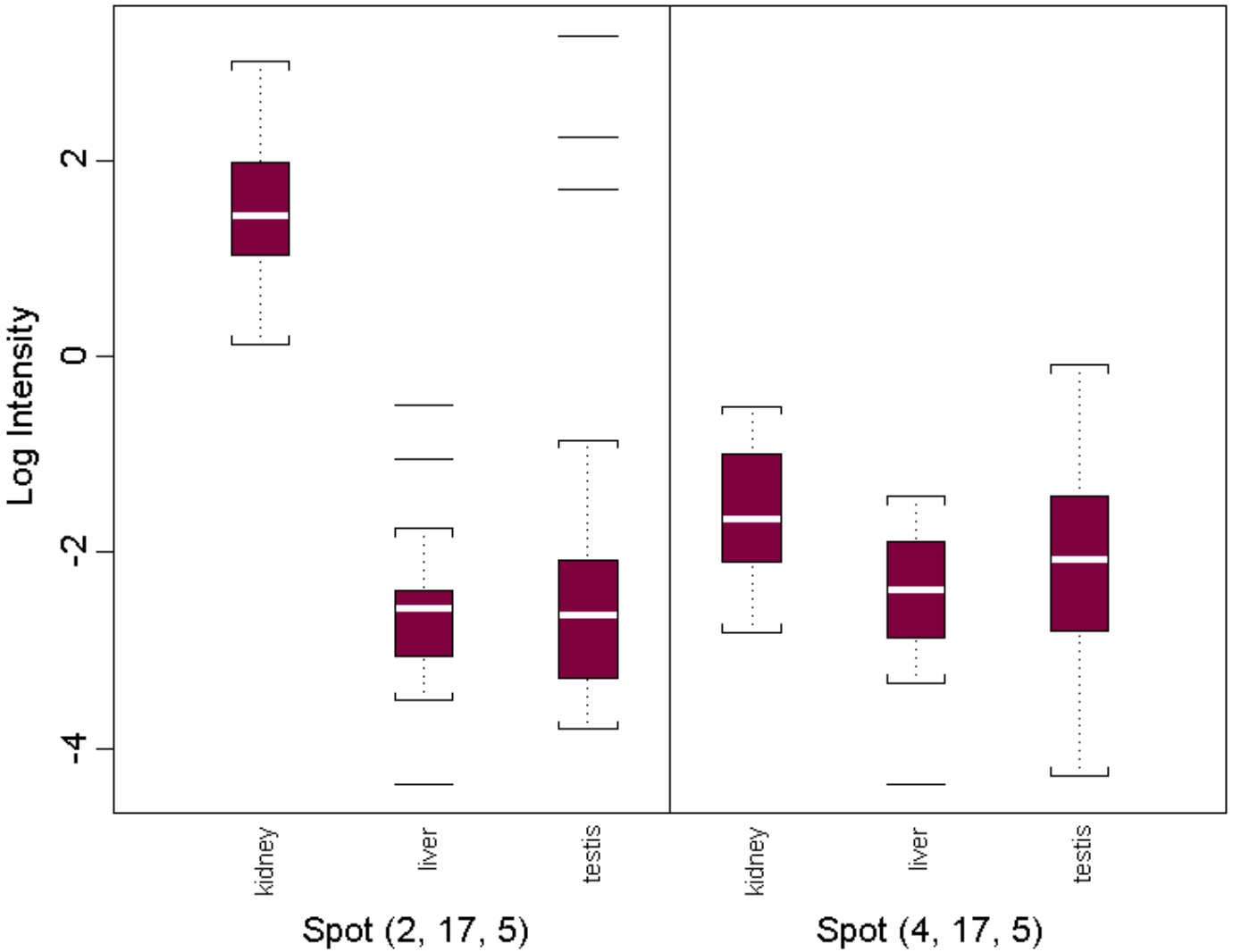
UniGene Example



UniGene Example



Villin Expression



Definition of abundance

- If the UniGene database entry for “gene expression” says that the cDNA sources of the clones found in a cluster included “kidney”, then we will say that the gene is abundant in kidney.

- Analogous definitions obviously apply for liver, testis, or other organs.

Abundance by consistency

Abundance	All UniGene	Consistent	Ambiguous
None	409	237	172
Kidney	129	76	53
Liver	284	169	115
Testis	372	231	141
Kidney, Liver	126	69	57
Kidney, Testis	226	146	80
Liver, Testis	960	609	351
All	2789	1835	963

Combining UniGene abundance with microarray data

- For each gene
  - Let $I = (K, L, T)$ be the binary vector of its abundance in three organs as recorded in the UniGene database.
  - Let $Y = (k, l, t)$ be the measured log intensity in the three organs.
- Model using a 3-dimensional multivariate normal distribution

$$Y|I = N_3(\mu_I, \Sigma_I)$$

- Average replicate experiments from same mouse with same dye to produce natural triplets of measurements.

Use consistently annotated genes to fit the model

Abundance	μ_K	μ_L	μ_T
None	2.027	2.129	2.012
Kidney	2.445	1.880	1.822
Liver	1.911	2.909	1.743
Testis	1.734	1.809	2.872
Kidney, Liver	3.282	3.051	1.961
Kidney, Testis	2.410	2.129	2.521
Liver, Testis	2.438	2.563	2.526
All	3.202	3.121	2.958

The estimates support the idea that (UniGene) abundant genes are expressed at higher levels than (UniGene) “rare” genes.

Distinguishing between competing sets of annotations

- Use parameters estimated from the genes with consistent annotations
- At the ambiguous spots, compute the log-likelihood of the observed data for each possible triple of abundance annotations
- Given a complete set of annotations, sum the log-likelihood values over all genes
  - Log-likelihood that the kidney data file contains the correct annotations is equal to $-52,241$
  - Log-likelihood that the liver data file contains the correct annotations is equal to $-60,183$

Scrambled rows

- Our "inspired guess" earlier was motivated by the idea that the rows containing the annotations had somehow been reordered.

- We permuted the rows 100 times to obtain empirical p-values for the observed log-likelihoods
  - P(kidney is correct) < 0.01
  - P(liver is correct) $= 0.57$.

- The log-likelihood of the kidney file annotations was not particularly close to the maximum of $-33,491$. This suggest that we can use the array data to refine the notion of "abundance" on a gene-by-gene basis.