Clustering

ACM

November 14, 2023

Example on Simulated Data

In this example we'll use some simulated data. The main reason for this is that simulating is the only way in which we know the "truth" is when it is generated.

First we'll generate some data from a mixture of Gaussian distributions with common covariance function.

```
library(printr)
library(clusterGeneration)
library(mclust)
library(EMCluster) #for the rand functions

set.seed(2)
M <- 1000 #number of subjects
num.vars <- 5 #number of outcomes
K <- 4 #number of clusters
prior_vec <-  c(0.2,0.15,0.2,0.45)

#Variance of the K clusters
cls_var <-genPositiveDefMat(num.vars, covMethod=c("unifcorrmat"),
                            rangeVar=c(0.2,5))$Sigma
round(cls_var,3)
```

4.217	-0.863	0.073	1.554	-0.234
-0.863	0.922	0.122	0.420	-0.121
0.073	0.122	1.867	-1.176	-0.120
1.554	0.420	-1.176	2.546	-0.062
-0.234	-0.121	-0.120	-0.062	0.916

```
#Mean of the K clusters
clst_mean <- matrix(rnorm(K*num.vars,0,2),K,num.vars)
round(clst_mean,3)
```

-0.733	0.649	-1.963	-4.356	1.108
-2.230	-1.137	1.647	2.947	2.421
-1.951	-2.069	-0.776	1.770	-1.285
2.242	-1.066	1.759	4.574	-0.315

```
clust_mem <- sample(1:K,M,replace=TRUE,prob = prior_vec)
data <- NULL
for(i in 1:K){
```

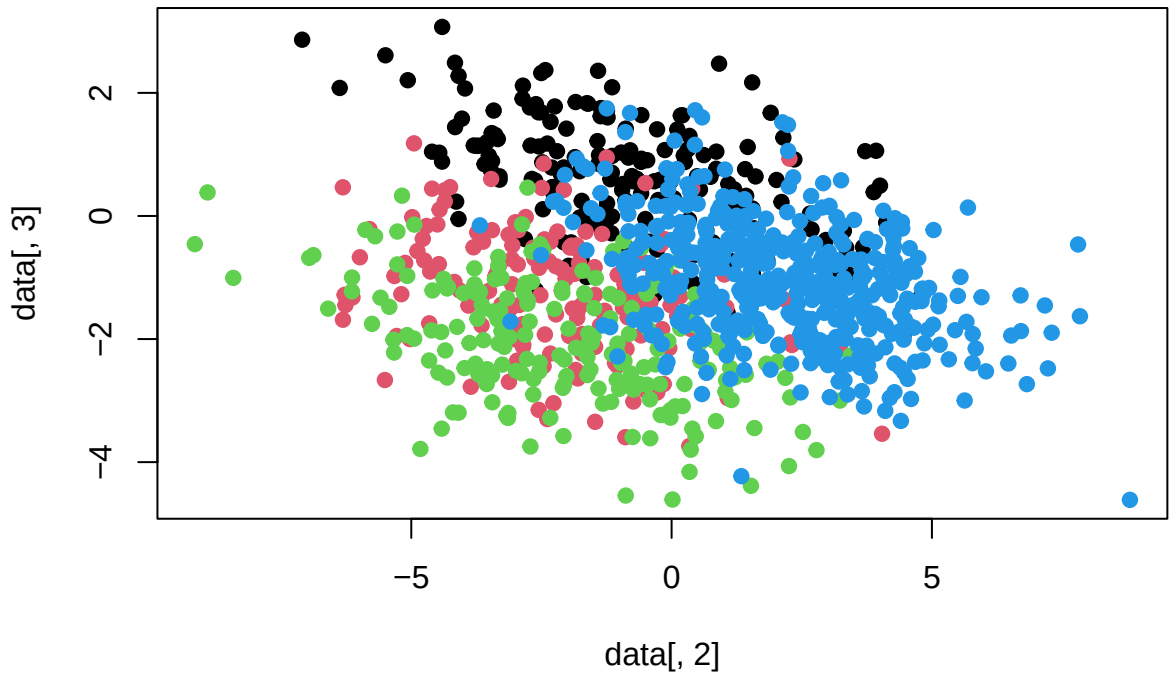
```
clus  <- mvrnorm(n=table(clust_mem)[i],clst_mean[i,], cls_var)
data <- rbind(data,cbind(i,clus))
}
table(clust_mem)
```

1	2	3	4
189	163	200	448

```
colnames(data) <- c("Cluster",paste(rep("X",num.vars),1:num.vars,sep="_"))
head(data)
```

Cluster	X_1	X_2	X_3	X_4	X_5
1	-0.2787224	-0.5454057	-4.387065	-3.545511	0.2433097
1	-0.2144903	0.1761455	-2.421134	-3.855347	1.4597301
1	1.5866256	-0.2007886	-4.370880	-1.331681	1.2339203
1	-2.8626456	1.9056216	-0.900932	-4.772957	1.6875881
1	0.2569547	0.8766283	-3.383705	-2.202340	0.8651877
1	-4.1582499	1.4435959	1.101295	-7.340681	1.6308381

```
plot(data[,2],data[,3],col=data[,1],pch=19)
```



Clustering with Gaussian mixture model

We'll use the Mclust function to cluster the data according to a Gaussian mixture model. This package allows for some different ways to structure the covariance matrix.

```
?Mclust
```

```
Model-Based Clustering
```

Description:

Model-based clustering based on parameterized finite Gaussian mixture models. Models are estimated by EM algorithm initialized by hierarchical model-based agglomerative clustering. The optimal model is then selected according to BIC.

Usage:

```
Mclust(data, G = NULL, modelNames = NULL,
     prior = NULL,
     control = emControl(),
     initialization = NULL,
     warn = mclust.options("warn"),
     x =  NULL,
     verbose = interactive(), ...)
```

Arguments:

data: A numeric vector, matrix, or data frame of observations. Categorical variables are not allowed. If a matrix or data frame, rows correspond to observations (n) and columns correspond to variables (d).

G: An integer vector specifying the numbers of mixture components (clusters) for which the BIC is to be calculated. The default is 'G=1:9'.

modelNames: A vector of character strings indicating the models to be fitted in the EM phase of clustering. The default is:

- for univariate data (d = 1): 'c("E", "V")'
- for multivariate data (n > d): all the models available in 'mclust.options("emModelNames")'
- for multivariate data (n <= d): the spherical and diagonal models, i.e. 'c("EII", "VII", "EEI", "EVI", "VEI", "VVI")'

The help file for 'mclustModelNames' describes the available models.

prior: The default assumes no prior, but this argument allows specification of a conjugate prior on the means and variances through the function 'priorControl'.
Note that, as described in 'defaultPrior', in the multivariate case only 10 out of 14 models may be used in conjunction with a prior, i.e. those available in _MCLUST_ up to version 4.4.

control: A list of control parameters for EM. The defaults are set by the call 'emControl()'.

```
initialization: A list containing zero or more of the following
          components:

          'hcPairs' A matrix of merge pairs for hierarchical clustering
              such as produced by function 'hc'.
              For multivariate data, the default is to compute a
              hierarchical agglomerative clustering tree by applying
              function 'hc' with model specified by
              'mclust.options("hcModelName")', and data transformation
              set by 'mclust.options("hcUse")'.
              All the input or a subset as indicated by the 'subset'
              argument is used for initial clustering.
              The hierarchical clustering results are then used to
              start the EM algorithm from a given partition.
              For univariate data, the default is to use quantiles to
              start the EM algorithm. However, hierarchical clustering
              could also be used by calling 'hc' with model specified
              as '"V"' or '"E"'.

          'subset' A logical or numeric vector specifying a subset of
              the data to be used in the initial hierarchical
              clustering phase.  No subset is used unless the number of
              observations exceeds the value specified by
              'mclust.options("subset")', which by default is set to
              2000 (see 'mclust.options').  Note that in this case to
              guarantee exact reproducibility of results a seed must be
              specified (see 'set.seed').

          'noise' A logical or numeric vector indicating an initial
              guess as to which observations are noise in the data. If
              numeric the entries should correspond to row indexes of
              the data. If supplied, a noise term will be added to the
              model in the estimation.

    warn: A logical value indicating whether or not certain warnings
          (usually related to singularity) should be issued.  The
          default is controlled by 'mclust.options'.

       x: An object of class ''mclustBIC''. If supplied, BIC values for
          models that have already been computed and are available in
          'x' are not recomputed.  All arguments, with the exception of
          'data', 'G' and 'modelName', are ignored and their values are
          set as specified in the attributes of 'x'. Defaults for 'G'
          and 'modelNames' are taken from 'x'.

 verbose: A logical controlling if a text progress bar is displayed
          during the fitting procedure. By default is 'TRUE' if the
          session is interactive, and 'FALSE' otherwise.

     ...: Catches unused arguments in indirect or list calls via
          'do.call'.

References:
```

```
     Scrucca L., Fop M., Murphy T. B. and Raftery A. E. (2016) mclust
     5: clustering, classification and density estimation using
     Gaussian finite mixture models, _The R Journal_, 8/1, pp. 289-317.

     Fraley C. and Raftery A. E. (2002) Model-based clustering,
     discriminant analysis and density estimation, _Journal of the
     American Statistical Association_, 97/458, pp. 611-631.

     Fraley C., Raftery A. E., Murphy T. B. and Scrucca L. (2012)
     mclust Version 4 for R: Normal Mixture Modeling for Model-Based
     Clustering, Classification, and Density Estimation. _Technical
     Report_ No. 597, Department of Statistics, University of
     Washington.

     C. Fraley and A. E. Raftery (2007) Bayesian regularization for
     normal mixture estimation and model-based clustering. _Journal of
     Classification_, 24, 155-181.
```

```
?mclustModelNames
```

```
MCLUST Model Names

Description:

     Description of model names used in the _MCLUST_ package.

Usage:

     mclustModelNames(model)

Arguments:

   model: A string specifying the model.

Details:

     The following models are available in package 'mclust':
     *univariate mixture*

     '"E"' equal variance (one-dimensional)

     '"V"' variable/unqual variance (one-dimensional)

     *multivariate mixture*

     '"EII"' spherical, equal volume

     '"VII"' spherical, unequal volume

     '"EEI"' diagonal, equal volume and shape

     '"VEI"' diagonal, varying volume, equal shape

     '"EVI"' diagonal, equal volume, varying shape
```

```
'"VVI"' diagonal, varying volume and shape

'"EEE"' ellipsoidal, equal volume, shape, and orientation

'"VEE"' ellipsoidal, equal shape and orientation (*)

'"EVE"' ellipsoidal, equal volume and orientation (*)

'"VVE"' ellipsoidal, equal orientation (*)

'"EEV"' ellipsoidal, equal volume and equal shape

'"VEV"' ellipsoidal, equal shape

'"EVV"' ellipsoidal, equal volume (*)

'"VVV"' ellipsoidal, varying volume, shape, and orientation

*single component*

'"X"' univariate normal

'"XII"' spherical multivariate normal

'"XXI"' diagonal multivariate normal

'"XXX"' ellipsoidal multivariate normal

(*) new models in 'mclust' version >= 5.0.0.
```

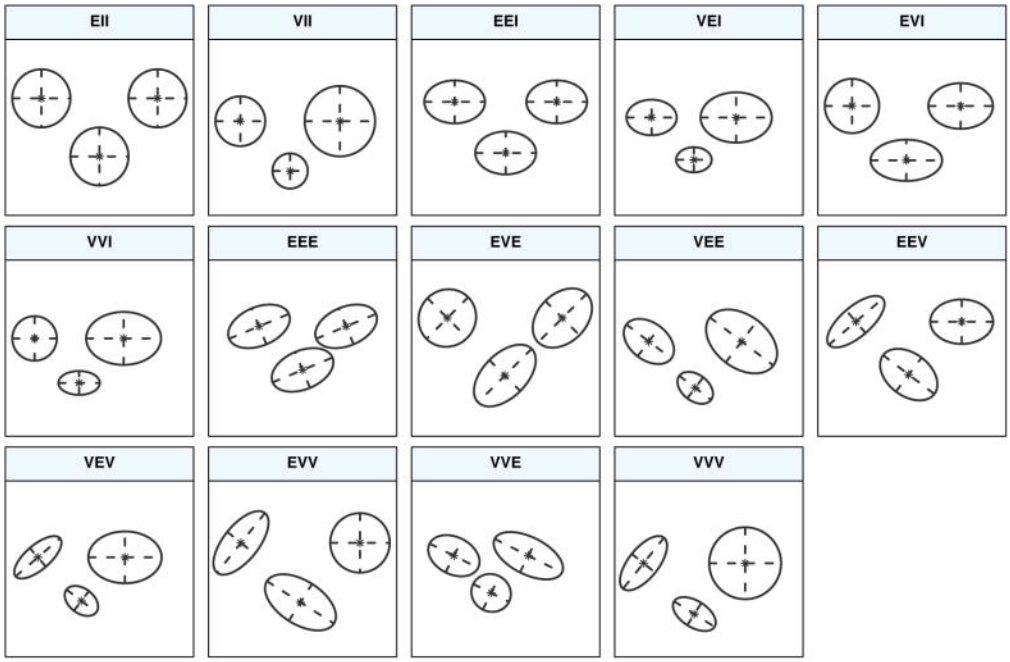


Figure 1: Ellipses of isodensity for each of the 14 Gaussian models obtained by eigen-decomposition in case of three groups in two dimensions.

Let's start out with the most general covariance structure:

```
HMM_diffcov <- Mclust(data[,-1], modelNames = "VVV")
summary(HMM_diffcov)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVV (ellipsoidal, varying volume, shape, and orientation) model with 4
## components:
##
##  log-likelihood    n df      BIC       ICL
##       -8443.177 1000 83 -17459.7 -17550.01
##
## Clustering table:
##   1   2   3   4
## 189 169 425 217
```

Note this has 83 parameters: $5 \times 4 = 20$ for the mean, $5 \times 4 = 20$ for the variance, $4 \times (5 \times 4)/2 = 40$ for the correlation, and 3 for the mixing proportions.)

```
est_class <- HMM_diffcov$classification
####accuracy rate of corrected Gaussian Mixture####
res <- table(est_class,data[,1])
res
```

est_class/	1	2	3	4
1	189	0	0	0
2	0	158	0	11
3	0	4	4	417
4	0	1	196	20

```
res_trans <- true_clus_func(data[,1], est_class)
res_trans
```

```
## $tabl
##     true_clus
## pred   1   2   3   4
##    1 189   0   0   0
##    2   0 158  11   0
##    3   0   4 417   4
##    4   0   1  20 196
##
## $ac_rate
## [1] 0.96
```

Rand Index

It turns out the notion of "accuracy" has some difficulties even with simulated data. Two measures that have been proposed to deal with this are the Rand and adjusted Rand indicies.

To understand RAND let δ_1 and δ_2 be two groups of a dataset. For example, one is an estimated grouping and the other it the true grouping. In all there are $\binom{n}{2}$ pairs of observations. Let a =number of pairs in the same subset in δ_1 and in the same subset in δ_2, and a =number of pairs in the a different subset in δ_1 and a

different subset in δ_2. Then

$$Rand = \frac{a+b}{\binom{n}{2}}.$$

The adjusted Rand index is the corrected-for-chance version of the Rand index.

```
?RRand
```

```
Rand Index and Adjusted Rand Index

Description:

     This function returns the Rand index and the adjusted Rand index
     for given true class ids and predicted class ids.

Usage:

     RRand(trcl, prcl, lab = NULL)

Arguments:

    trcl: true class ids.

    prcl: predicted class ids.

     lab: known ids for semi-supervised clustering.
```

```
HMM_v_true <- RRand(data[,1], est_class)
HMM_v_true
```

```
##    Rand adjRand  Eindex
##  0.9521  0.8851  0.1759
```

Now were going to try out some different covariance matrix options. Getting more simple as we go.

```
HMM_diffcor <- Mclust(data[,-1], modelNames = "VEV")
summary(HMM_diffcor)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEV (ellipsoidal, equal shape) model with 4 components:
##
##  log-likelihood    n df       BIC       ICL
##       -8450.356 1000 71 -17391.16 -17474.47
##
## Clustering table:
##   1   2   3   4
## 189 170 433 208
```

Note this has 71 parameters: $5 \times 4 = 20$ for the mean, 5 for the variance of the X_j's, 3 for the different variance multipliers for each cluster, $4 \times (5 \times 4)/2 = 40$ for the correlation, and 3 for the mixing proportions. This model has 8 parameters for the variance, which is as low as we can go without assuming all the X_j's have equal variance.

```
HMM_1cor4var <- Mclust(data[,-1], modelNames = "VVE")
summary(HMM_1cor4var)
```

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust VVE (ellipsoidal, equal orientation) model with 4 components:

 log-likelihood    n df      BIC       ICL
      -8457.243 1000 53 -17280.6 -17360.19

Clustering table:
  1   2   3   4
189 165 442 204
```

Note this has 53 parameters: $5 \times 4 = 20$ for the mean, $5 \times 4 = 20$ for the variance, $5 \times 4/2 = 10$ for the correlation, and 3 for the mixing proportions.

```
HMM_1covp <- Mclust(data[,-1], modelNames = "VEE")
summary(HMM_1covp)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VEE (ellipsoidal, equal shape and orientation) model with 4 components:
##
##  log-likelihood    n df      BIC       ICL
##       -8462.542 1000 41 -17208.3 -17285.73
##
## Clustering table:
##   1   2   3   4
## 189 166 442 203
```

Note this has 41 parameters: $5 \times 4 = 20$ for the mean, 5 for the variance of the X_j's, 3 for the different variance multipliers for each cluster, $5 \times 4/2 = 10$ for the correlation, and 3 for the mixing proportions.

```
HMM_1cov <- Mclust(data[,-1], modelNames = "EEE")
summary(HMM_1cov)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 4
## components:
##
##  log-likelihood    n df       BIC       ICL
##       -8466.392 1000 38 -17195.28 -17272.16
##
## Clustering table:
##   1   2   3   4
## 189 167 444 200
```

Note this has 38 parameters: $5 \times 4 = 20$ for the mean, 5 for the variance of the X_j's, $5 \times 4/2 = 10$ for the correlation, and 3 for the mixing proportions. This is the true model

This package can give Bayesian Information Criterion (BIC) for multiple values of K.

```
HMM_1cov$BIC
```

```
## Bayesian Information Criterion (BIC):
##         EEE
## 1 -21125.29
## 2 -17839.63
## 3 -17753.72
## 4 -17195.28
## 5 -17234.62
## 6 -17269.95
## 7 -17308.55
## 8 -17326.14
## 9 -17367.73
##
## Top 3 models based on the BIC criterion:
##     EEE,4     EEE,5     EEE,6
## -17195.28 -17234.62 -17269.95
```

Since $K = 4$ has the best BIC, the classification it gives will correspond to that. Let's see how this compares to the first HMM:

```
####accuracy rate of corrected Gaussian Mixture####
est_class2 <- HMM_1cov$classification

res_trans <- true_clus_func(data[,1], est_class2)
res_trans
```

```
## $tabl
##     true_clus
## pred   1   2   3   4
##    1 189   0   0   0
##    2   0 158   9   0
##    3   0   4 430  10
##    4   0   1   9 190
##
## $ac_rate
## [1] 0.967
```

```
HMM2_v_true <- RRand(data[,1], est_class2)
HMM3_v_true <- RRand(data[,1], HMM_1covp$classification)
HMM4_v_true <- RRand(data[,1], HMM_diffcor$classification)
HMM5_v_true <- RRand(data[,1], HMM_1cor4var$classification)

data.frame(measure = c("Rand","adjRand"), Diff_cov = c(HMM_v_true$Rand,HMM_v_true$adjRand),
                        Diff_cor2 = c(HMM4_v_true$Rand, HMM4_v_true$adjRand),
           One_cor = c(HMM5_v_true$Rand, HMM5_v_true$adjRand),
           One_cov_pVar = c(HMM3_v_true$Rand, HMM3_v_true$adjRand),
           One_cov = c(HMM2_v_true$Rand, HMM2_v_true$adjRand) )
```

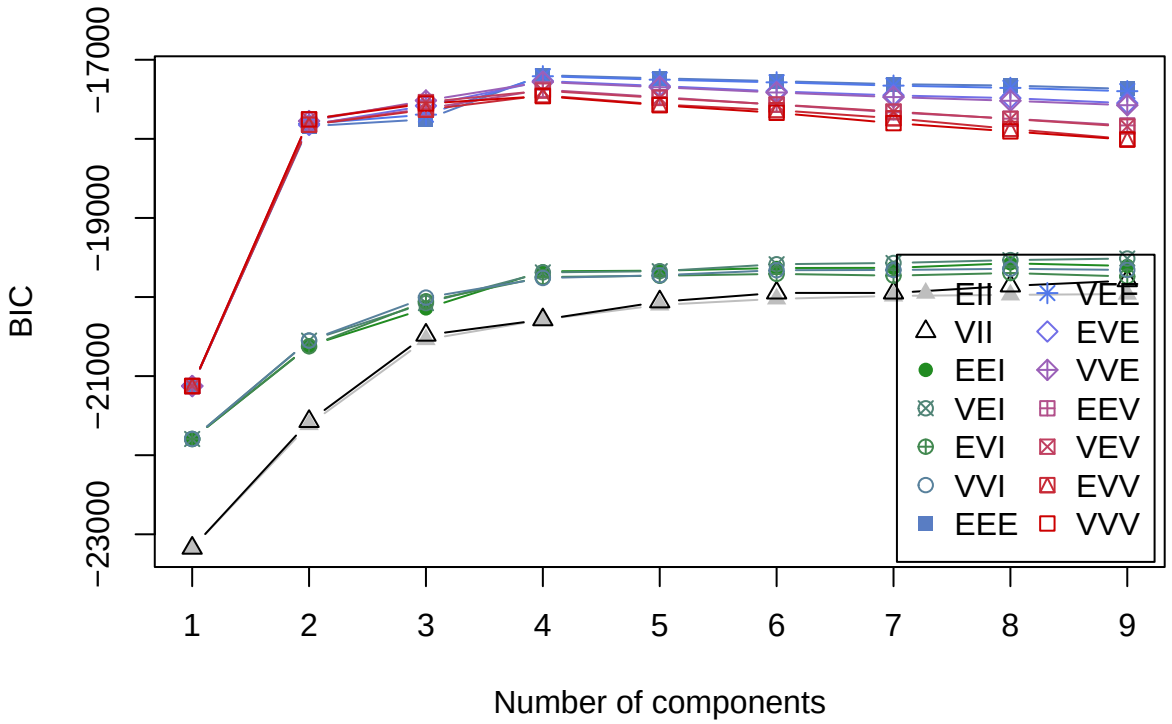
measure	Diff_cov	Diff_cor2	One_cor	One_cov_pVar	One_cov
Rand	0.9520501	0.9592232	0.9625586	0.9602082	0.9601481
adjRand	0.8851425	0.9026367	0.9109835	0.9053888	0.9053258

We can actually run with each method and compare the BIC values

```
BIC <- mclustBIC(data[,-1])
BIC
```

```
## Bayesian Information Criterion (BIC):
##         EII       VII       EEI       VEI       EVI       VVI       EEE
## 1 -23174.55 -23174.55 -21795.03 -21795.03 -21795.03 -21795.03 -21125.29
## 2 -21612.05 -21572.80 -20617.19 -20551.16 -20622.91 -20550.87 -17839.63
## 3 -20537.27 -20480.92 -20143.75 -20076.61 -20053.36 -20006.80 -17753.72
## 4 -20286.39 -20286.59 -19674.87 -19688.40 -19747.34 -19759.46 -17195.28
## 5 -20099.87 -20059.97 -19666.68 -19673.12 -19730.99 -19724.91 -17234.62
## 6 -20023.07 -19947.94 -19632.90 -19585.46 -19707.85 -19660.78 -17269.95
## 7 -19985.94 -19948.28 -19630.91 -19569.34 -19729.17 -19657.90 -17308.55
## 8 -19972.60 -19859.00 -19573.81 -19532.95 -19696.71 -19643.01 -17326.14
## 9 -19962.71 -19792.95 -19606.84 -19512.96 -19740.27 -19657.57 -17367.73
##         VEE       EVE       VVE       EEV       VEV       EVV       VVV
## 1 -21125.29 -21125.29 -21125.29 -21125.29 -21125.29 -21125.29 -21125.29
## 2 -17807.75 -17823.90 -17774.31 -17815.23 -17753.56 -17822.67 -17753.70
## 3 -17693.85 -17581.52 -17515.30 -17618.43 -17534.99 -17631.95 -17553.29
## 4 -17208.30 -17268.52 -17280.60 -17378.53 -17391.16 -17450.07 -17459.70
## 5 -17251.95 -17332.08 -17346.67 -17472.00 -17482.76 -17565.57 -17574.96
## 6 -17285.10 -17399.52 -17413.65 -17568.37 -17564.93 -17637.37 -17670.67
## 7 -17327.04 -17450.58 -17471.79 -17664.37 -17654.54 -17738.94 -17801.23
## 8 -17355.67 -17484.98 -17518.41 -17744.80 -17747.60 -17874.59 -17905.68
## 9 -17397.18 -17549.25 -17574.58 -17829.94 -17844.83 -18002.91 -18010.16
##
## Top 3 models based on the BIC criterion:
##     EEE,4     VEE,4     EEE,5
## -17195.28 -17208.30 -17234.62
```

```
plot(BIC)
```



K-means Clustering

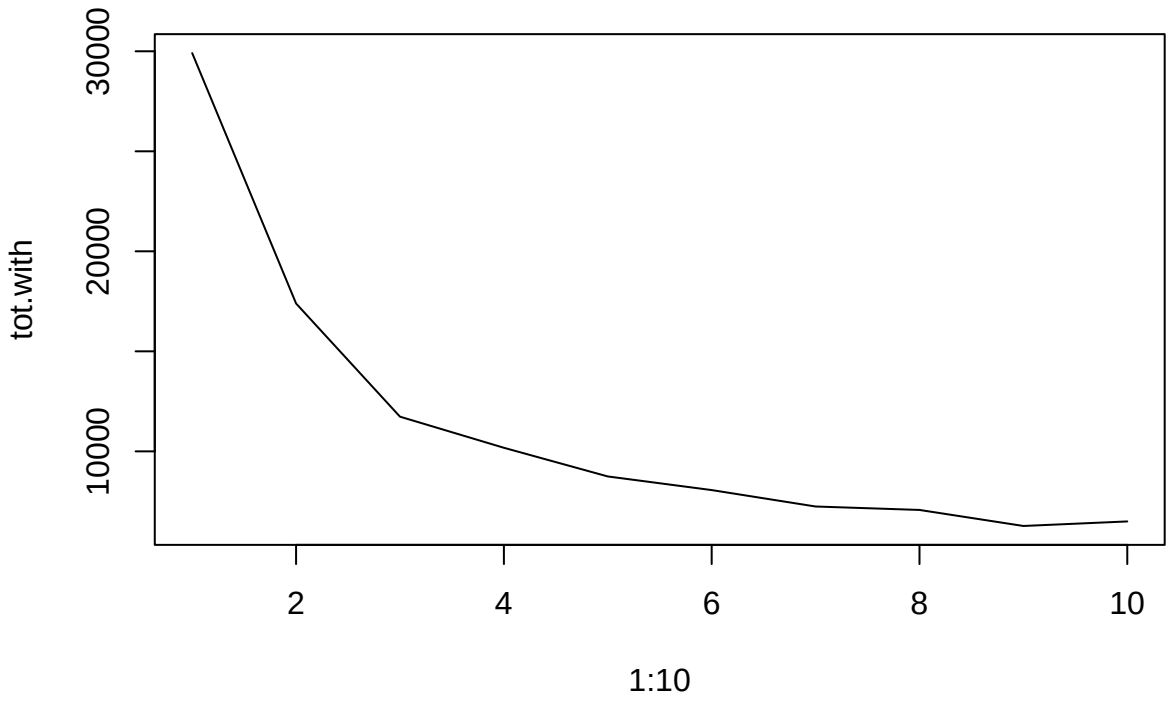
Now let's cluster the data using K-means, and compare the results.

```
?kmeans
```

```
K-Means Clustering

Description:

     Perform k-means clustering on a data matrix.

Usage:

     kmeans(x, centers, iter.max = 10, nstart = 1,
            algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
                          "MacQueen"), trace = FALSE)
     ## S3 method for class 'kmeans'
     fitted(object, method = c("centers", "classes"), ...)

Arguments:

       x: numeric matrix of data, or an object that can be coerced to
          such a matrix (such as a numeric vector or a data frame with
          all numeric columns).

 centers: either the number of clusters, say k, or a set of initial
          (distinct) cluster centres.  If a number, a random set of
          (distinct) rows in 'x' is chosen as the initial centres.

iter.max: the maximum number of iterations allowed.

  nstart: if 'centers' is a number, how many random sets should be
          chosen?

algorithm: character: may be abbreviated.  Note that '"Lloyd"' and
          '"Forgy"' are alternative names for one algorithm.

  object: an R object of class '"kmeans"', typically the result 'ob' of
          'ob <- kmeans(..)'.

  method: character: may be abbreviated. '"centers"' causes 'fitted' to
          return cluster centers (one for each input point) and
          '"classes"' causes 'fitted' to return a vector of class
          assignments.

   trace: logical or integer number, currently only used in the default
          method ('"Hartigan-Wong"'): if positive (or true), tracing
          information on the progress of the algorithm is produced.
          Higher values may produce more tracing information.

     ...: not used.
```

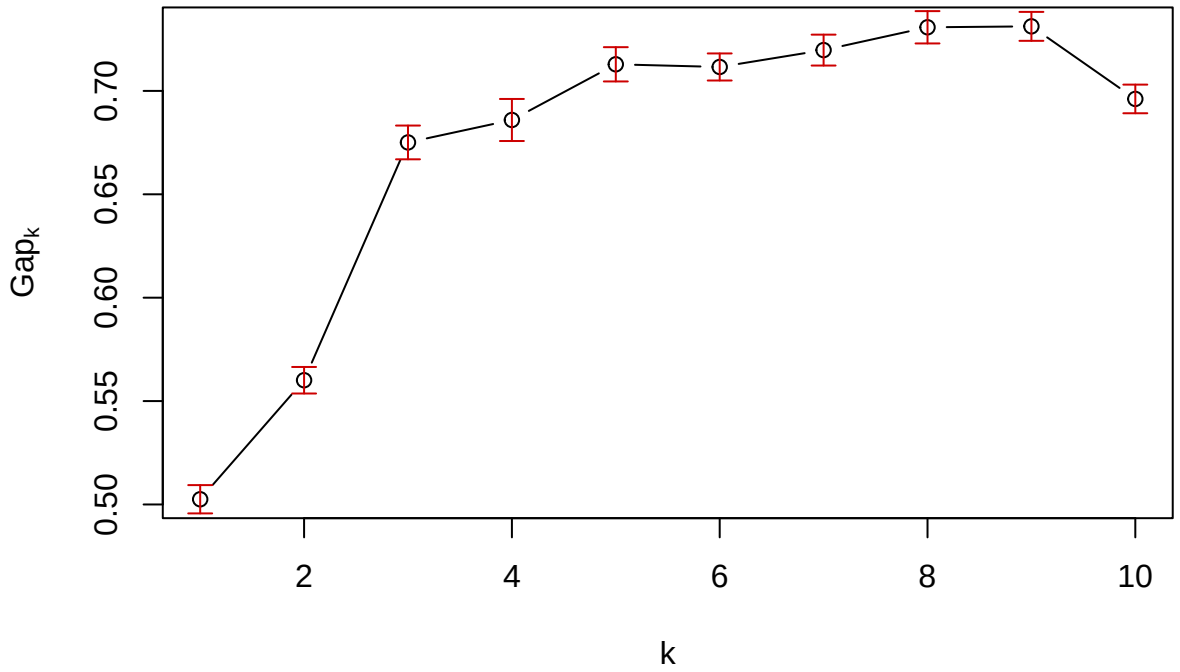
Let's look at K-means with different values of K. Here we'll plot the within class dissimilarity (i.e., $W(\delta)$)

```
tot.with <- NULL
for(k in 1:10){
cl<-kmeans(data[,-1],k)
tot.with <- c(tot.with,cl$tot.withinss)
}
plot(1:10,tot.with,type="l")
```



We can also calculate the gap statistic:

```
library(cluster)
set.seed(1234)
Gap_stat <- clusGap(data[,-1],FUNcluster = kmeans,K.max = 10)
plot(Gap_stat)
```



The Gap statistic at $K = 3$ is not quite one standard deviation from the Gap statistic at $K = 4$. Let's check the results:

```
Gap_stat
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = data[, -1], FUNcluster = kmeans, K.max = 10)
## B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstSEmax', SE.factor=1): 5
##           logW   E.logW       gap      SE.sim
##  [1,] 7.476281 7.978797 0.5025163 0.006839619
##  [2,] 7.214520 7.774596 0.5600766 0.006415696
##  [3,] 7.025841 7.700919 0.6750777 0.008163259
##  [4,] 6.949301 7.635238 0.6859362 0.010168840
##  [5,] 6.872317 7.585176 0.7128591 0.008290481
##  [6,] 6.828186 7.539757 0.7115712 0.006541361
##  [7,] 6.783090 7.502826 0.7197356 0.007475020
##  [8,] 6.738919 7.469683 0.7307645 0.007812478
##  [9,] 6.709528 7.440741 0.7312125 0.006995624
## [10,] 6.719168 7.415281 0.6961129 0.006923961
```

```
0.6859362 - 0.6750777
```

```
## [1] 0.0108585
```

We'll re-fit the data with $K = 5$ clusters and compare to our previous results.

```
cl<-kmeans(data[,-1],5)
names(cl)
```

```
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```r
K_means_v_true <- RRand(data[,1], cl[[1]])
K_means_v_HMM <- RRand(cl[[1]], est_class)
K_means_v_true
```

```
##    Rand adjRand  Eindex
##  0.8355  0.5741  0.1283
```

```r
HMM_v_true
```

```
##    Rand adjRand  Eindex
##  0.9521  0.8851  0.1759
```

```r
K_means_v_HMM
```

```
##    Rand adjRand  Eindex
##  0.8349  0.5656  0.1236
```

K-mediods Clustering

Lets try the same using K-mediods:

```r
?pam
```

```
Partitioning Around Medoids

Description:

     Partitioning (clustering) of the data into 'k' clusters "around
     medoids", a more robust version of K-means.

Usage:

     pam(x, k, diss = inherits(x, "dist"),
         metric = c("euclidean", "manhattan"),
         medoids = if(is.numeric(nstart)) "random",
         nstart = if(variant == "faster") 1 else NA,
         stand = FALSE, cluster.only = FALSE,
         do.swap = TRUE,
         keep.diss = !diss && !cluster.only && n < 100,
         keep.data = !diss && !cluster.only,
         variant = c("original", "o_1", "o_2", "f_3", "f_4", "f_5", "faster"),
         pamonce = FALSE, trace.lev = 0)

Arguments:

       x: data matrix or data frame, or dissimilarity matrix or object,
          depending on the value of the 'diss' argument.

          In case of a matrix or data frame, each row corresponds to an
          observation, and each column corresponds to a variable.  All
          variables must be numeric.  Missing values ('NA's) _are_
          allowed-as long as every pair of observations has at least
          one case not missing.

          In case of a dissimilarity matrix, 'x' is typically the
          output of 'daisy' or 'dist'.  Also a vector of length
```

n*(n-1)/2 is allowed (where n is the number of observations), and will be interpreted in the same way as the output of the above-mentioned functions. Missing values ('NA's) are _not_ allowed.

k: positive integer specifying the number of clusters, less than the number of observations.

diss: logical flag: if TRUE (default for 'dist' or 'dissimilarity' objects), then 'x' will be considered as a dissimilarity matrix. If FALSE, then 'x' will be considered as a matrix of observations by variables.

metric: character string specifying the metric to be used for calculating dissimilarities between observations. The currently available options are "euclidean" and "manhattan". Euclidean distances are root sum-of-squares of differences, and manhattan distances are the sum of absolute differences. If 'x' is already a dissimilarity matrix, then this argument will be ignored.

medoids: NULL (default) or length-'k' vector of integer indices (in '1:n') specifying initial medoids instead of using the '_build_' algorithm.

nstart: used only when 'medoids = "random"': specifies the _number_ of random "starts"; this argument corresponds to the one of 'kmeans()' (from R's package 'stats').

stand: logical; if true, the measurements in 'x' are standardized before calculating the dissimilarities. Measurements are standardized for each variable (column), by subtracting the variable's mean value and dividing by the variable's mean absolute deviation. If 'x' is already a dissimilarity matrix, then this argument will be ignored.

cluster.only: logical; if true, only the clustering will be computed and returned, see details.

do.swap: logical indicating if the *swap* phase should happen. The default, 'TRUE', correspond to the original algorithm. On the other hand, the *swap* phase is much more computer intensive than the *build* one for large n, so can be skipped by 'do.swap = FALSE'.

keep.diss, keep.data: logicals indicating if the dissimilarities and/or input data 'x' should be kept in the result. Setting these to 'FALSE' can give much smaller results and hence even save memory allocation _time_.

pamonce: logical or integer in '0:6' specifying algorithmic short cuts as proposed by Reynolds et al. (2006), and Schubert and Rousseeuw (2019, 2021) see below.

```
 variant: a 'character' string denoting the variant of PAM algorithm to
          use; a more self-documenting version of 'pamonce' which
          should be used preferably; note that '"faster"' not only uses
          'pamonce = 6' but also 'nstart = 1' and hence 'medoids =
          "random"' by default.

trace.lev: integer specifying a trace level for printing diagnostics
          during the build and swap phase of the algorithm.  Default
          '0' does not print anything; higher values print increasingly
          more.
```

```
pam_fit <-pam(data[,-1], K, metric = "manhattan")
K_mediod_v_true <- RRand(data[,1], pam_fit$clustering)
K_mediod_v_true
```

```
##    Rand adjRand  Eindex
##  0.7790  0.4545  0.1571
```

```
HMM_v_true
```

```
##    Rand adjRand  Eindex
##  0.9521  0.8851  0.1759
```

```
K_means_v_true
```

```
##    Rand adjRand  Eindex
##  0.8355  0.5741  0.1283
```

Example on Microarray Data

Here is an example that has been used in the literature:

```
library("ISLR")
?NCI60
```

```
NCI 60 Data

Description:

     NCI microarray data. The data contains expression levels on 6830
     genes from 64 cancer cell lines. Cancer type is also recorded.

Usage:

     NCI60
```

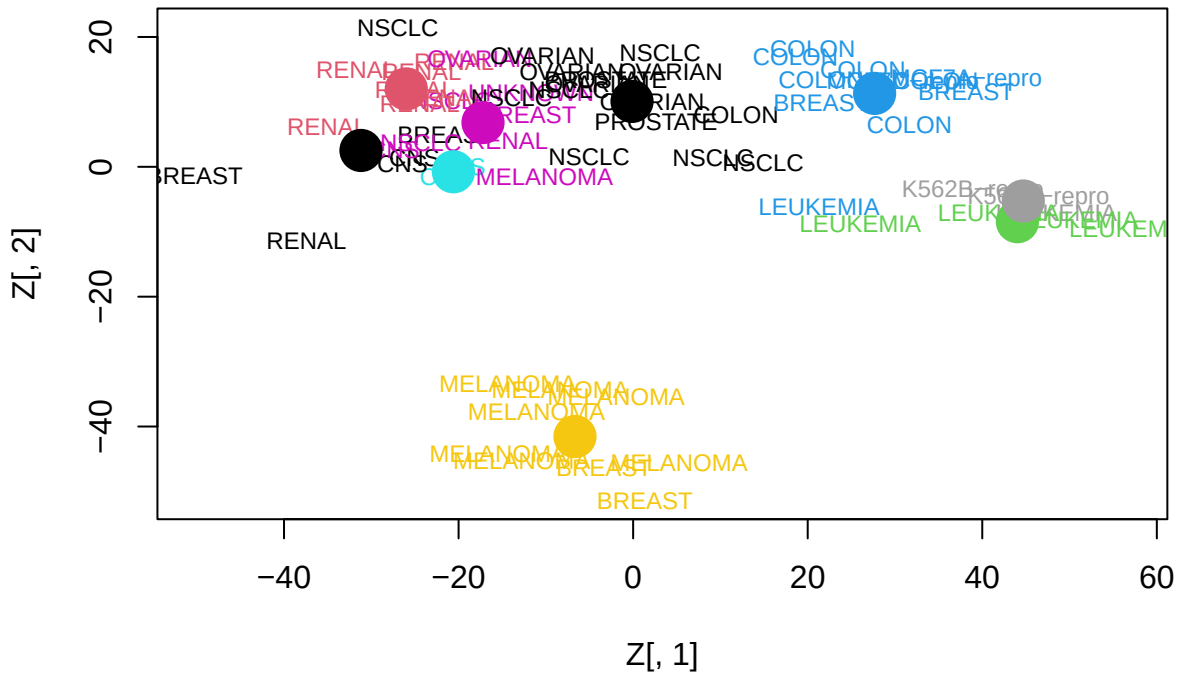
```
ncidat = t(NCI60$data)
colnames(ncidat) = NCI60$labs

dim(ncidat)
```

```
[1] 6830   64
```

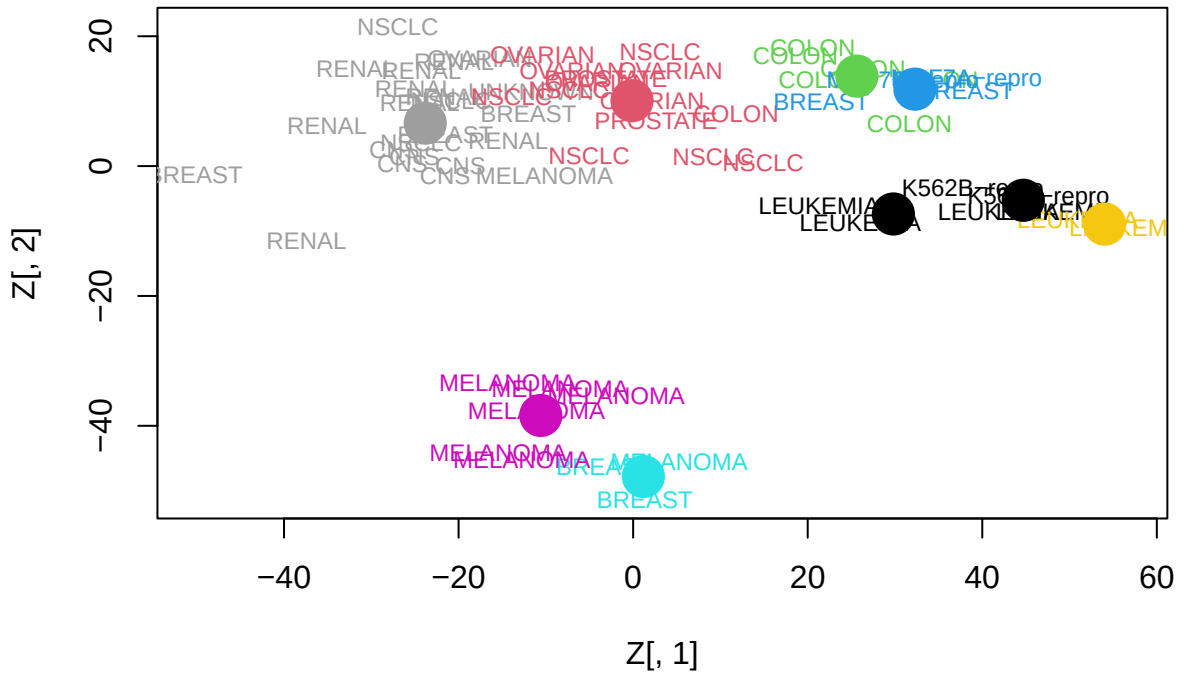
```
unique(colnames(ncidat))
```

```
 [1] "CNS"         "RENAL"       "BREAST"      "NSCLC"       "UNKNOWN"
 [6] "OVARIAN"     "MELANOMA"    "PROSTATE"    "LEUKEMIA"    "K562B-repro"
[11] "K562A-repro" "COLON"       "MCF7A-repro" "MCF7D-repro"
```

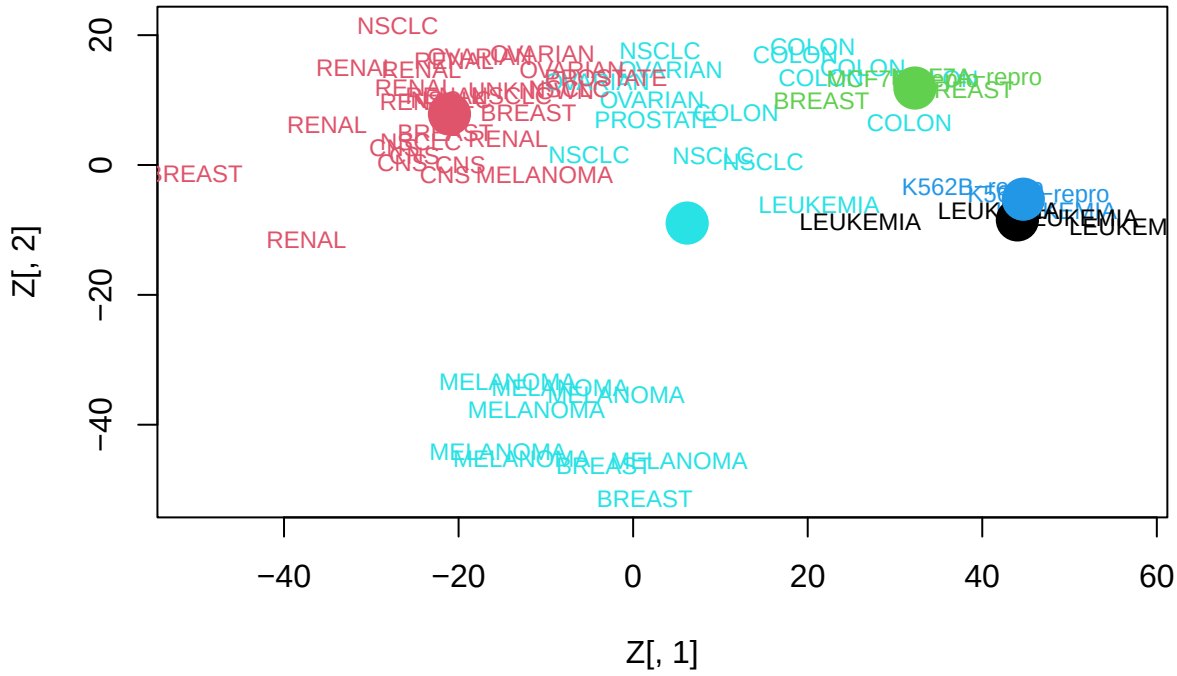
```
############
#apply K-means
K = 9
km = kmeans(t(ncidat),centers=K)

#how do we visualize K-means results?

#PCA - take SVD to get solution
#center genes, but don't scale
X = t(scale(t(ncidat),center=TRUE,scale=FALSE))
sv = svd(t(X));
U = sv$u
V = sv$v
D = sv$d
Z = t(X)%*%V;

plot(Z[,1],Z[,2],col=km$cluster,type="n")
text(Z[,1],Z[,2],colnames(ncidat),cex=.75,col=km$cluster)
cens = km$centers
points(cens%*%V[,1],cens%*%V[,2],col=1:K,pch=16,cex=3)
```




```
#Re-run and see if solution changes
K = 9
km = kmeans(t(ncidat),centers=K)
plot(Z[,1],Z[,2],col=km$cluster,type="n")
text(Z[,1],Z[,2],colnames(ncidat),cex=.75,col=km$cluster)
cens = km$centers
points(cens%*%V[,1],cens%*%V[,2],col=1:K,pch=16,cex=3)
```

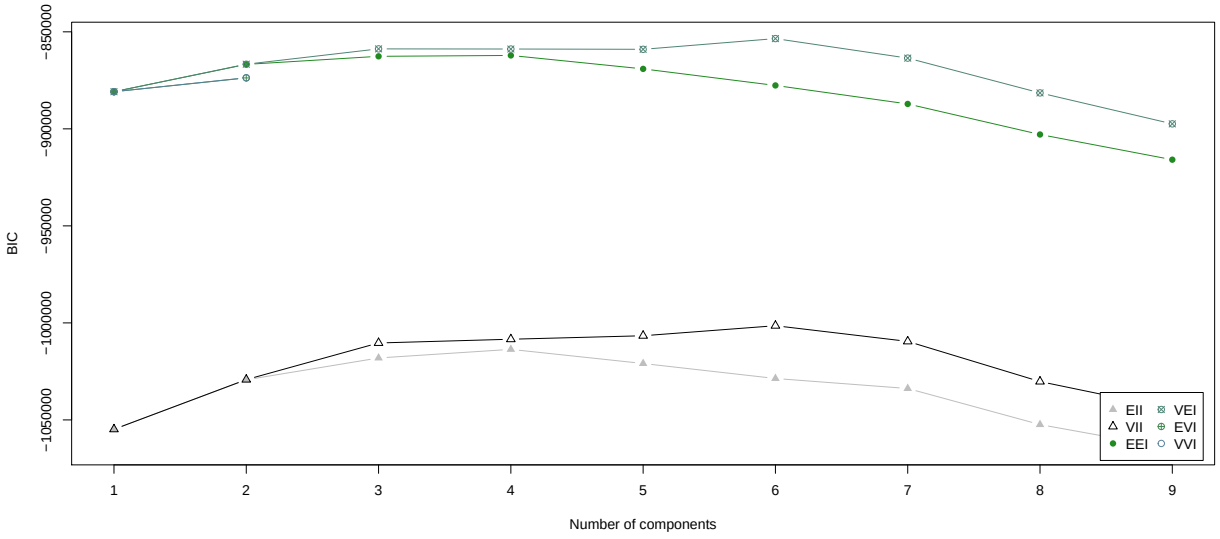


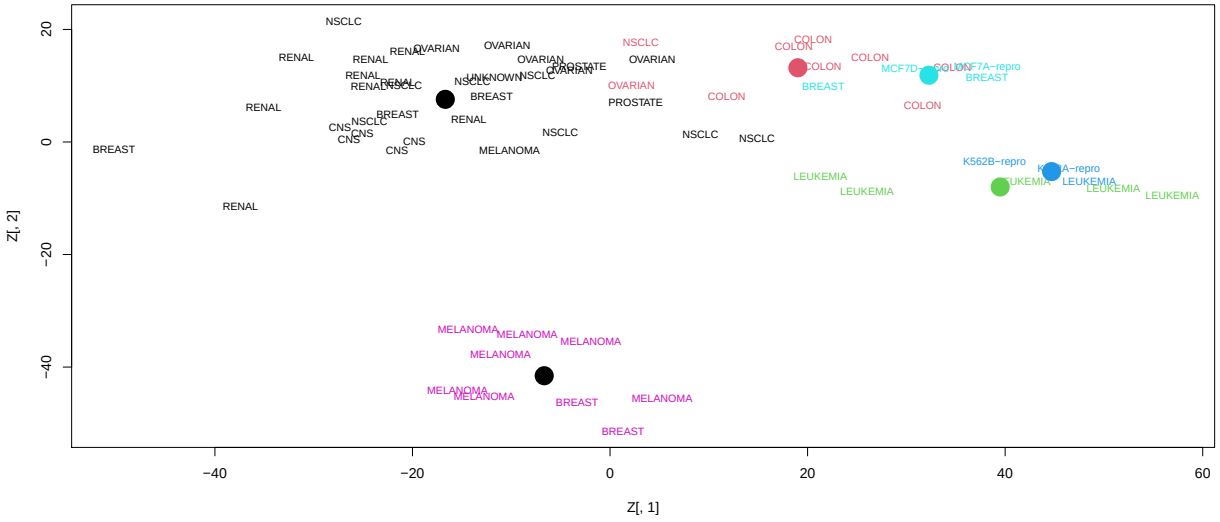

```
#try different K
K = 5
km = kmeans(t(ncidat),centers=K)
plot(Z[,1],Z[,2],col=km$cluster,type="n")
text(Z[,1],Z[,2],colnames(ncidat),cex=.75,col=km$cluster)
cens = km$centers
points(cens%*%V[,1],cens%*%V[,2],col=1:K,pch=16,cex=3)
```



Let's try with the Gaussian mixture model:

```r
nci_GMM <- mclustBIC(t(ncidat))
plot(nci_GMM)
nci_GMM_fit <- Mclust(t(ncidat), G = 6, modelNames = "VEI")

K_hat <- as.numeric(nci_GMM_fit$classification)
plot(Z[,1],Z[,2],col=K_hat,type="n")
text(Z[,1],Z[,2],colnames(ncidat),cex=.75,col=K_hat)
cens = t(nci_GMM_fit$parameters$mean)
points(cens%*%V[,1],cens%*%V[,2],col=1:K,pch=16,cex=3)
```





Hierachical Clustering

```
?hclust
```

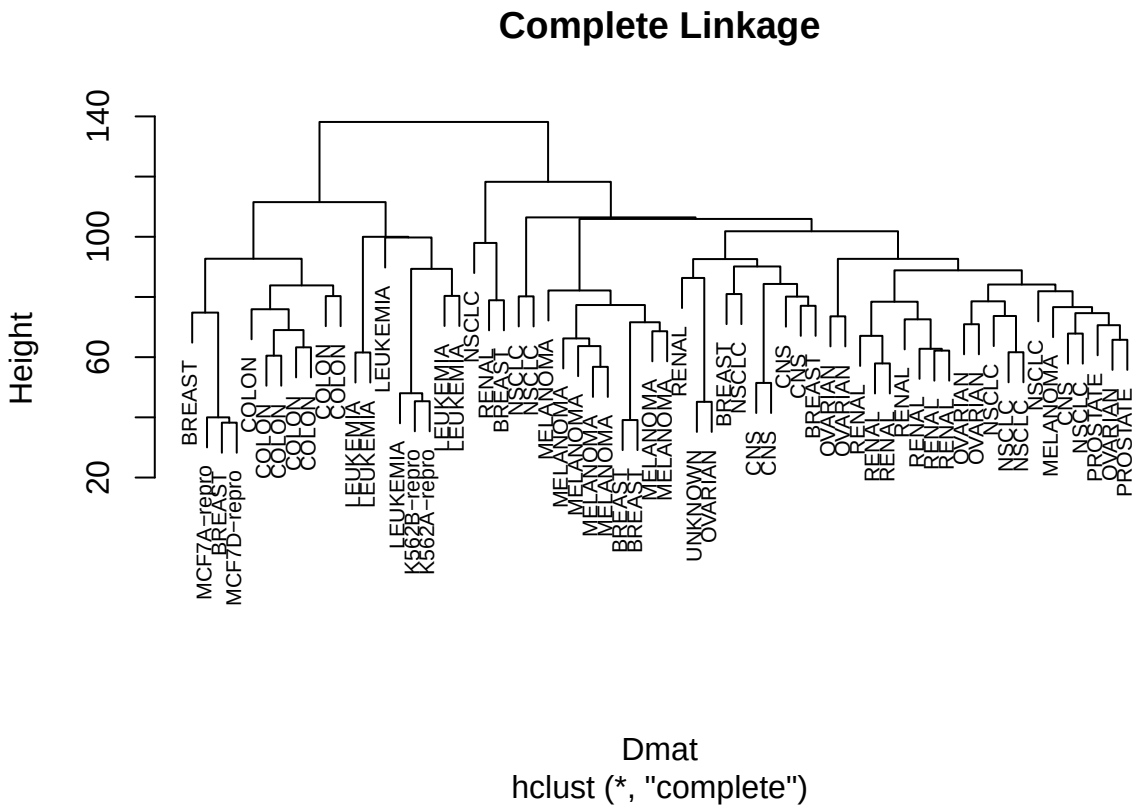
```
Hierarchical Clustering

Description:

     Hierarchical cluster analysis on a set of dissimilarities and
     methods for analyzing it.

Usage:

     hclust(d, method = "complete", members = NULL)

     ## S3 method for class 'hclust'
     plot(x, labels = NULL, hang = 0.1, check = TRUE,
          axes = TRUE, frame.plot = FALSE, ann = TRUE,
          main = "Cluster Dendrogram",
          sub = NULL, xlab = NULL, ylab = "Height", ...)

Arguments:

       d: a dissimilarity structure as produced by 'dist'.

  method: the agglomeration method to be used.  This should be (an
          unambiguous abbreviation of) one of '"ward.D"', '"ward.D2"',
          '"single"', '"complete"', '"average"' (= UPGMA), '"mcquitty"'
          (= WPGMA), '"median"' (= WPGMC) or '"centroid"' (= UPGMC).

 members: 'NULL' or a vector with length size of 'd'. See the 'Details'
          section.

       x: an object of the type produced by 'hclust'.

    hang: The fraction of the plot height by which labels should hang
          below the rest of the plot.  A negative value will cause the
          labels to hang down from 0.

   check: logical indicating if the 'x' object should be checked for
          validity.  This check is not necessary when 'x' is known to
          be valid such as when it is the direct result of 'hclust()'.
          The default is 'check=TRUE', as invalid inputs may crash R
          due to memory violation in the internal C plotting code.

  labels: A character vector of labels for the leaves of the tree.  By
          default the row names or row numbers of the original data are
          used.  If 'labels = FALSE' no labels at all are plotted.

axes, frame.plot, ann: logical flags as in 'plot.default'.

main, sub, xlab, ylab: character strings for 'title'.  'sub' and 'xlab'
          have a non-NULL default when there's a 'tree$call'.
```

```
...: Further graphical arguments.  E.g., 'cex' controls the size
     of the labels (if plotted) in the same way as 'text'.

Details:

     This function performs a hierarchical cluster analysis using a set
     of dissimilarities for the n objects being clustered.  Initially,
     each object is assigned to its own cluster and then the algorithm
     proceeds iteratively, at each stage joining the two most similar
     clusters, continuing until there is just a single cluster.  At
     each stage distances between clusters are recomputed by the
     Lance-Williams dissimilarity update formula according to the
     particular clustering method being used.

     A number of different clustering methods are provided.  _Ward's_
     minimum variance method aims at finding compact, spherical
     clusters.  The _complete linkage_ method finds similar clusters.
     The _single linkage_ method (which is closely related to the
     minimal spanning tree) adopts a 'friends of friends' clustering
     strategy.  The other methods can be regarded as aiming for
     clusters with characteristics somewhere between the single and
     complete link methods.  Note however, that methods '"median"' and
     '"centroid"' are _not_ leading to a _monotone distance_ measure,
     or equivalently the resulting dendrograms can have so called
     _inversions_ or _reversals_ which are hard to interpret, but note
     the trichotomies in Legendre and Legendre (2012).

     Two different algorithms are found in the literature for Ward
     clustering.  The one used by option '"ward.D"' (equivalent to the
     only Ward option '"ward"' in R versions <= 3.0.3) _does not_
     implement Ward's (1963) clustering criterion, whereas option
     '"ward.D2"' implements that criterion (Murtagh and Legendre 2014).
     With the latter, the dissimilarities are _squared_ before cluster
     updating.  Note that 'agnes(*, method="ward")' corresponds to
     'hclust(*, "ward.D2")'.

     If 'members != NULL', then 'd' is taken to be a dissimilarity
     matrix between clusters instead of dissimilarities between
     singletons and 'members' gives the number of observations per
     cluster.  This way the hierarchical cluster algorithm can be
     'started in the middle of the dendrogram', e.g., in order to
     reconstruct the part of the tree above a cut (see examples).
     Dissimilarities between clusters can be efficiently computed
     (i.e., without 'hclust' itself) only for a limited number of
     distance/linkage combinations, the simplest one being _squared_
     Euclidean distance and centroid linkage.  In this case the
     dissimilarities between the clusters are the squared Euclidean
     distances between cluster means.

     In hierarchical cluster displays, a decision is needed at each
     merge to specify which subtree should go on the left and which on
     the right.  Since, for n observations there are n-1 merges, there
     are 2^{(n-1)} possible orderings for the leaves in a cluster tree,
     or dendrogram.  The algorithm used in 'hclust' is to order the
```

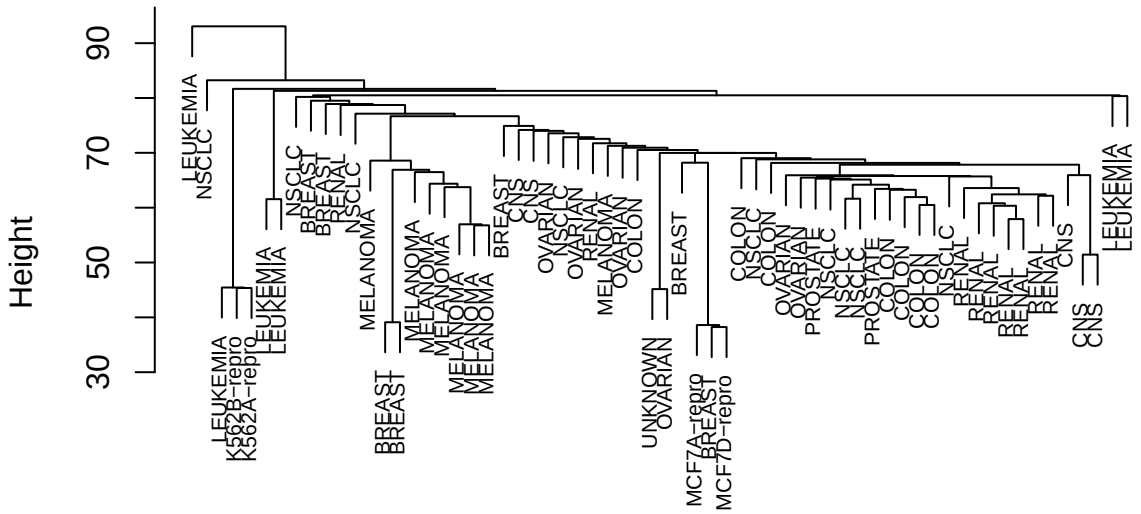
```
subtree so that the tighter cluster is on the left (the last,
i.e., most recent, merge of the left subtree is at a lower value
than the last merge of the right subtree).  Single observations
are the tightest clusters possible, and merges involving two
observations place them in order by their observation sequence
number.
```

Now we'll do the same analysis using Hierachical Clustering.

```
#complete linakge - Euclidean distance
cols = as.numeric(as.factor(colnames(ncidat)))
Dmat = dist(t(ncidat))
com.hclust = hclust(Dmat,method="complete")
plot(com.hclust,cex=.7,main="Complete Linkage")
```

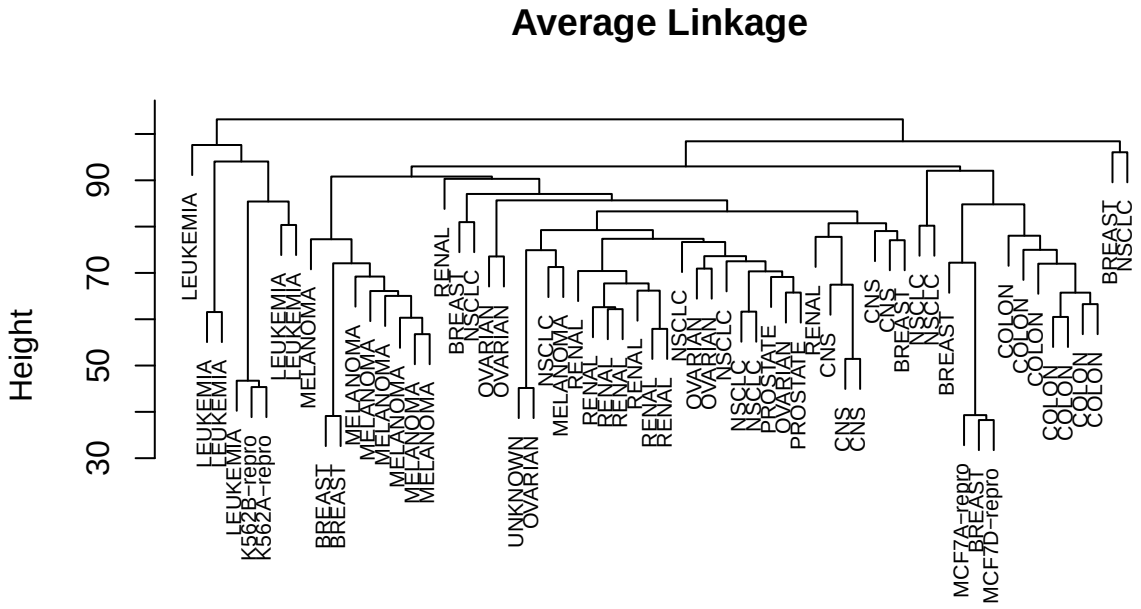



```
#single linakge
sing.hclust = hclust(Dmat,method="single")
plot(sing.hclust,cex=.7,main="Single Linkage")
```



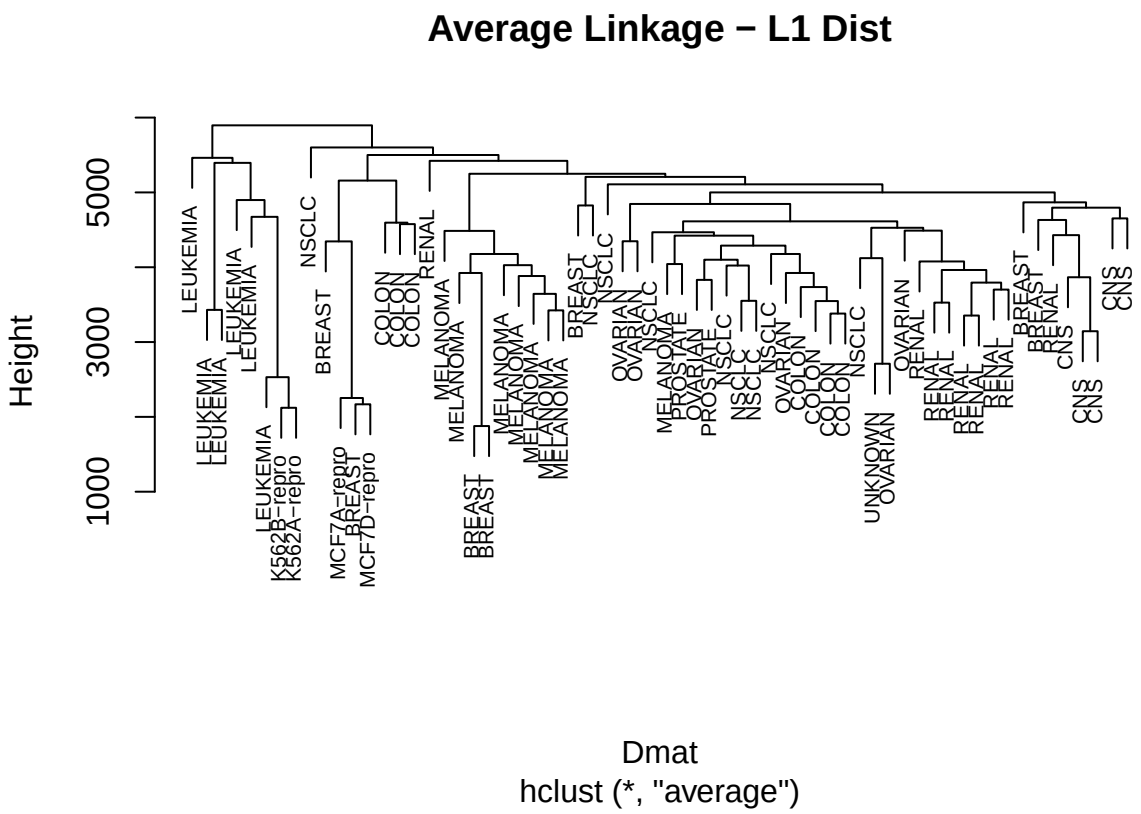
Dmat
hclust (*, "single")

```
#average linakge
ave.hclust = hclust(Dmat,method="average")
plot(ave.hclust,cex=.7,main="Average Linkage")
```



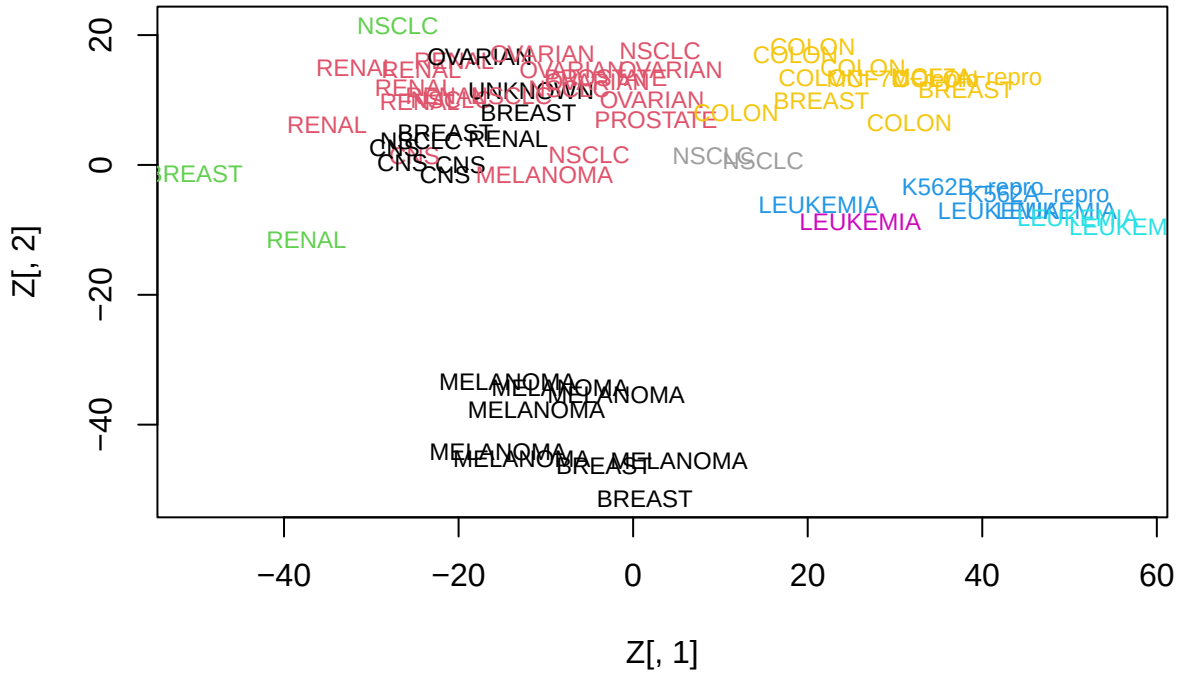
Dmat
hclust (*, "average")

```
#complete linkage with different distances
Dmat = dist(t(ncidat),method="manhattan") #L1 distance
ave.hclust = hclust(Dmat,method="average")
plot(ave.hclust,cex=.7,main="Average Linkage - L1 Dist")
```



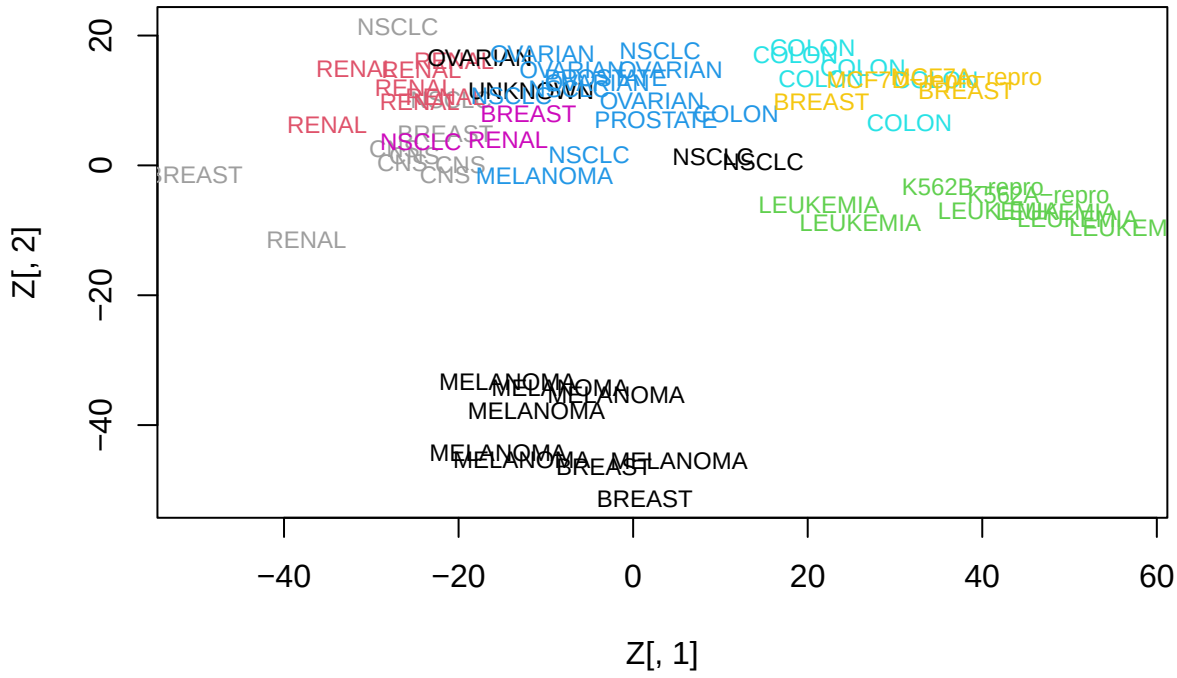
Now we'll do some cutting of the tree:

```
com.hclust_cut <- cutree(com.hclust,k=9)

plot(Z[,1],Z[,2],col=com.hclust_cut,type="n")
text(Z[,1],Z[,2],colnames(ncidat),cex=.75,col=com.hclust_cut)
```



Let's compare this with the K-means result.

```
K = 9
km = kmeans(t(ncidat),centers=K)
plot(Z[,1],Z[,2],col=km$cluster,type="n")
text(Z[,1],Z[,2],colnames(ncidat),cex=.75,col=km$cluster)
```

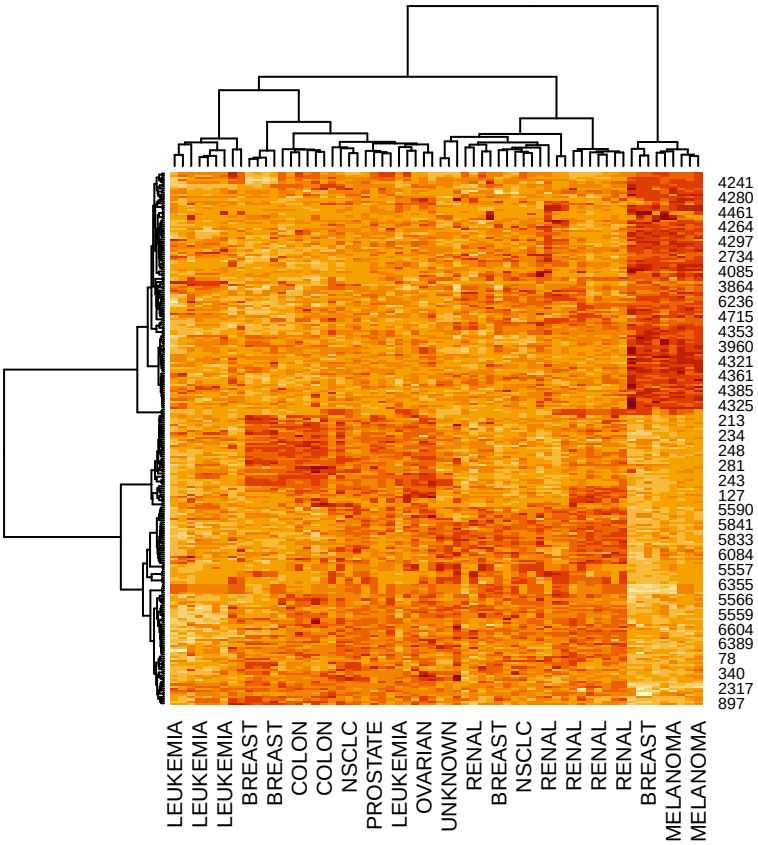


Biclustering and the Cluster Heatmap

We didn't discuss this one in the notes, but upon visualization it pretty easy to see what's going on.

Now were going to look at the same data using Biclustering with a Cluster Heatmap.

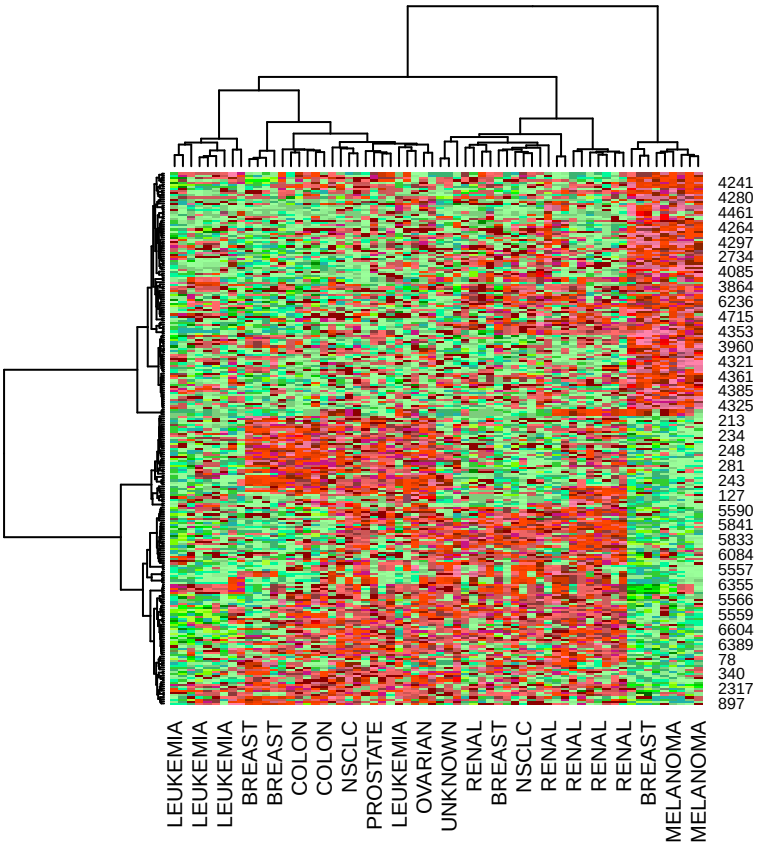
```
ncidat = t(NCI60$data)
colnames(ncidat) = NCI60$labs

#filter genes using PCA
X = t(scale(t(ncidat),center=TRUE,scale=FALSE))
sv = svd(t(X));
V = sv$v

#PC loadings - visualize data by limiting to top genes in magnitude of the PC loadings

j = 2
ord = order(abs(V[,j]),decreasing=TRUE)
x = as.matrix(X[ord[1:250],])

#cluster heatmap - uses Ward's linkage (complete is default)
heatmap(x, hclustfun=function(x)hclust(x,method="ward.D") )
```




```
##We can also define our own colors
bb = grep("green",colors())
cc = grep("red",colors())
gcol2 = colors()[c(bb[1:28],cc)]
gcol2
```

```
##  [1] "darkgreen"          "darkolivegreen"     "darkolivegreen1"
##  [4] "darkolivegreen2"    "darkolivegreen3"    "darkolivegreen4"
##  [7] "darkseagreen"       "darkseagreen1"      "darkseagreen2"
## [10] "darkseagreen3"      "darkseagreen4"      "forestgreen"
## [13] "green"              "green1"             "green2"
## [16] "green3"             "green4"             "greenyellow"
## [19] "lawngreen"          "lightgreen"         "lightseagreen"
## [22] "limegreen"          "mediumseagreen"     "mediumspringgreen"
## [25] "palegreen"          "palegreen1"         "palegreen2"
## [28] "palegreen3"         "darkred"            "indianred"
## [31] "indianred1"         "indianred2"         "indianred3"
## [34] "indianred4"         "mediumvioletred"    "orangered"
## [37] "orangered1"         "orangered2"         "orangered3"
## [40] "orangered4"         "palevioletred"      "palevioletred1"
## [43] "palevioletred2"     "palevioletred3"     "palevioletred4"
## [46] "red"                "red1"               "red2"
## [49] "red3"               "red4"               "violetred"
## [52] "violetred1"         "violetred2"         "violetred3"
## [55] "violetred4"
```

```
#cluster heatmap - uses Ward's linkage (complete is default)
heatmap(x,col=gcol2,hclustfun=function(x)hclust(x,method="ward.D"))
```


4241
4280
4461
4264
4297
2734
4085
3864
6236
4715
4353
3960
4321
4361
4385
4325
213
234
248
281
243
127
5590
5841
5833
6084
5557
6355
5566
5559
6604
6389
78
340
2317
897
LEUKEMIA
LEUKEMIA
LEUKEMIA
BREAST
BREAST
COLON
COLON
NSCLC
PROSTATE
LEUKEMIA
OVARIAN
UNKNOWN
RENAL
BREAST
NSCLC
RENAL
RENAL
RENAL
RENAL
BREAST
MELANOMA
MELANOMA