ELEMENTS OF DATA SCIENCE AND STATISTICAL LEARNING

SPRING 2017

Week 6

OUTLINE

- Principal Component Analysis (PCA)
- Clustering: K-Means
- Clustering: Hierarchical

OUTLINE

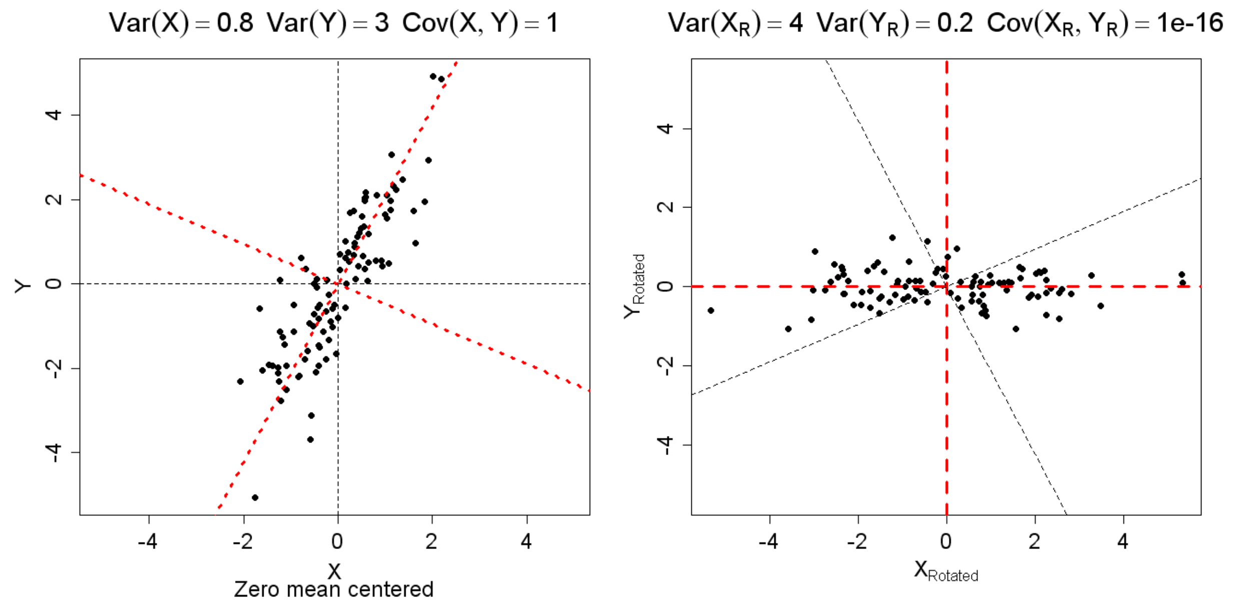
- The topic of today's and next lectures is *unsupervised learning*
- Unsupervised learning is a collection of methods and approaches aimed at finding "interesting" (or suspicious!) structure in the data in the *absence* of known outcome
- Important for understanding the data, discovery and/or hypothesis generation, dimensionality reduction:
  - identifying distinct groups of customers, patients, experimental samples, stocks, etc;
  - One can follow up trying to understand what makes the groups different; what do these groups correlate with (disease subtype? survival outcome? voting pattern?
  - Even in the presence of the defined outcome, if very distinct groups of observations are present, they may show qualitatively different response and we might be able to improve performance of our *supervised* learning by preprocessing data and/or instituting additional measurements aimed at placing observations into one of the discovered groups more reliably
- Important for sanity checks and "data QC"
  - Do my experimental samples under treatment X "group together" in some sense and away from untreated controls?

OUTLINE

- Principal Component Analysis (PCA)
- Clustering: K-Means
- Clustering: Hierarchical

PRINCIPAL COMPONENT ANALYSIS (PCA)

- The main, qualitative idea of PCA is simple: find orthogonal basis, such that the variance of the data is the largest along the first direction (first principal component, or PC1), followed by the variance along PC2, etc
- This is simply a rotation in the multi-dimensional space defined by the predictor variables. This rotation diagonalizes the covariance matrix
- The rationale behind PCA: we *hope* that the direction(s) of the largest variance is where the signal is strongest.
- As PCA is a rotation, the new coordinate vectors (new “variables”) are linear transformations of the original variables



COVARIANCE MATRIX

- Covariance between two variables X, Y is defined as:

$Cov(X,Y) = E[(x-\mu_X)(y-\mu_Y)]$ (note that the correlation coefficient is simply $\rho(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}$)

- Covariance matrix is the matrix of pairwise covariances calculated among $X_1, \ldots, X_M$
- The above expression is the exact definition in terms of the random variables (and their distributions)
  - When we have finite-size samples, we use estimators; for two samples of size N of variables X_i and X_j we have:

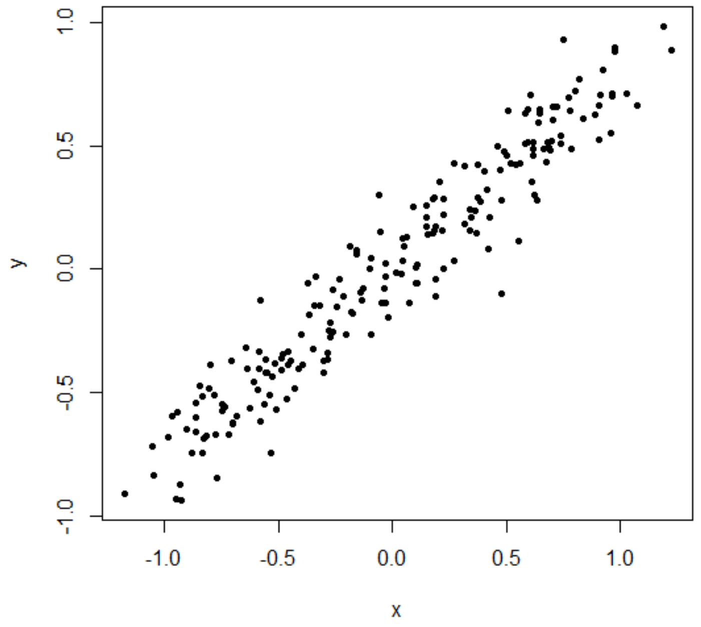
$$C_{ij} = \frac{1}{N-1}\sum_{k=1}^{N}(x_{ik}-\bar{x}_i)(x_{jk}-\bar{x}_j)$$

  - Note that diagonal elements (i=j) of covariance are simply $Var(X_i)$
  - Covariance matrix is diagonal when all pairs of variables are uncorrelated

BUILDING AN EXAMPLE

- First (trivial) example
- Let us build just two random variables with association between them

```
> x=-100:100
> y=0.8*x
> x=(x+10*rnorm(length(x)))/100
> y=(y+10*rnorm(length(y)))/100
> plot(x,y,pch=19,cex=0.7)
> cov(x,y); cor(x,y)
[1] 0.2650011
[1] 0.9614664
# we can also calculate covariance manually:
> sum((x-mean(x))*(y-mean(y)))/(length(x)-1)
[1] 0.2650011
```

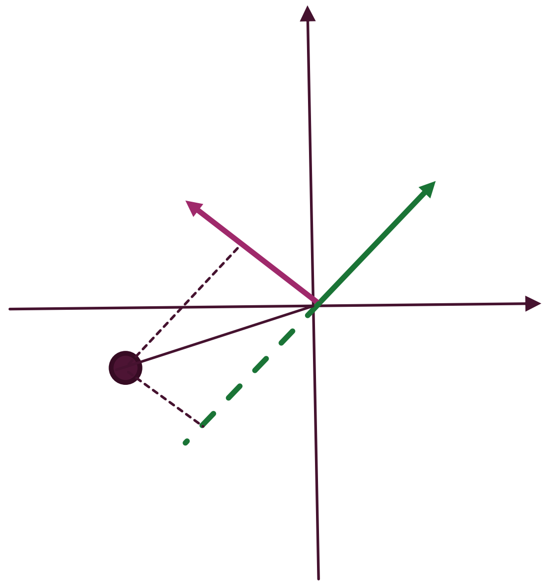


COMPUTATION OF PRINCIPAL COMPONENTS

- `prcomp()` is all it takes to run PCA in R [note that we use scaling here!]
- For educational purposes only, we also calculate covariance matrix and "manually" diagonalize it

```
> x.n<-(x-mean(x))/sd(x)
> y.n<-(y-mean(y))/sd(y)
> df=data.frame(x=x,y=y)
> df.n<-data.frame(x=x.n,y=y.n)
> px<-prcomp(df,retx=TRUE,scale=TRUE)
> px
Standard deviations:
[1] 1.4005236 0.1962997

Rotation:
        PC1        PC2
x 0.7071068 -0.7071068
y 0.7071068  0.7071068
> cov(df.n) # covariance of scaled
          x         y
x 1.0000000 0.9614664
y 0.9614664 1.0000000
```

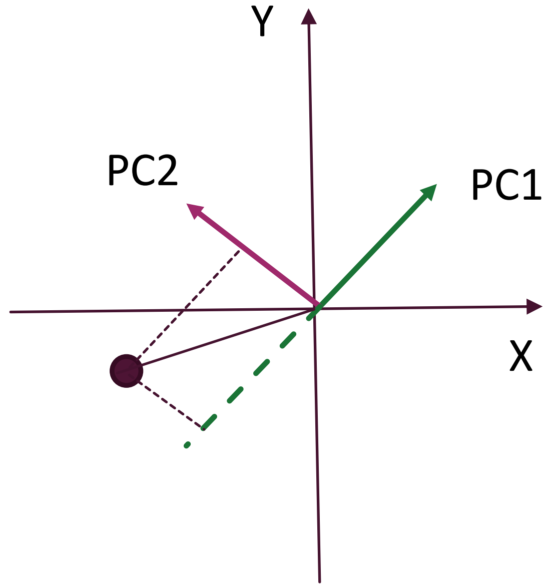
```
> eigen(cov(df.n))
$values
[1] 1.96146642 0.03853358

$vectors
          [,1]       [,2]
[1,] 0.7071068 -0.7071068
[2,] 0.7071068  0.7071068
> sqrt(eigen(cov(df.n))$values)
[1] 1.4005236 0.1962997
```



VARIABLE TRANSFORMATION

- Principal components define orthogonal directions, with max variance along PC1, followed by PC2 etc
- Each "point" (observation with particular realization (X,Y)=(x,y)) can be represented in (projected onto) the new coordinates
- This is a linear transformation Z_1=a X + b Y; Z_2= c X + d Y, with coefficients given by the rotation matrix (components of PC1/PC2 in the original coordinates X,Y)

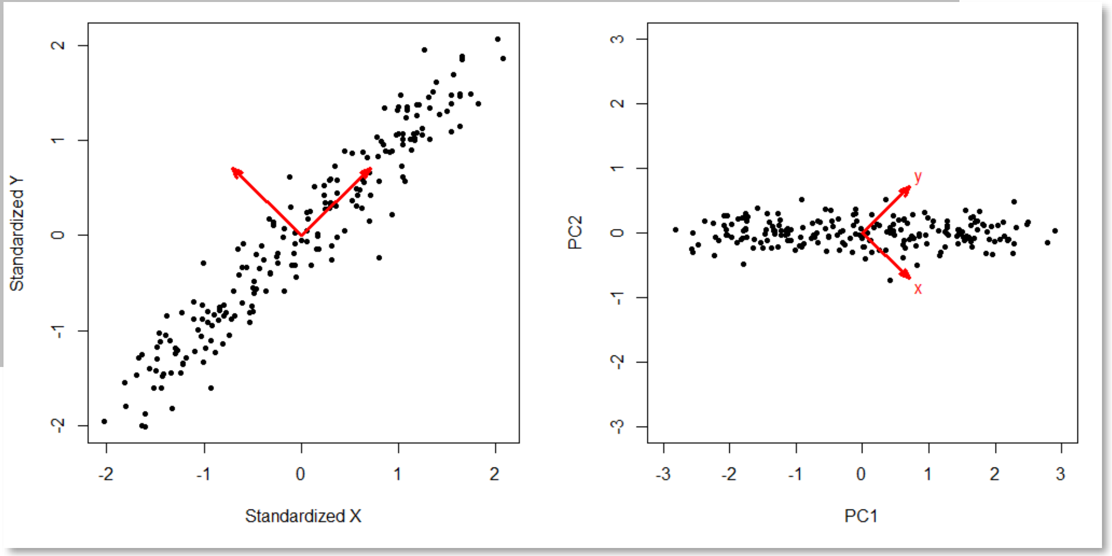
```
> px$x[1:5,]
            PC1         PC2
[1,] -2.814294  0.04932501
[2,] -2.549006  0.00628545
[3,] -2.238006  0.15599989
[4,] -2.576767 -0.25904136
[5,] -2.464949 -0.18985451
> as.matrix(df.n[1:5,]) %*% px$rotation
        PC1         PC2
1 -2.814294  0.04932501
2 -2.549006  0.00628545
3 -2.238006  0.15599989
4 -2.576767 -0.25904136
5 -2.464949 -0.18985451
```



ORIGINAL VS PC COORDINATE SYSTEMS

- Columns of the rotation matrix (eigenvectors of the covariance matrix) are the PC directions in the original coordinates
- Rows of the rotation matrix (columns of the transposed matrix) are the original variable (X, Y, …) directions in the PC coordinate system

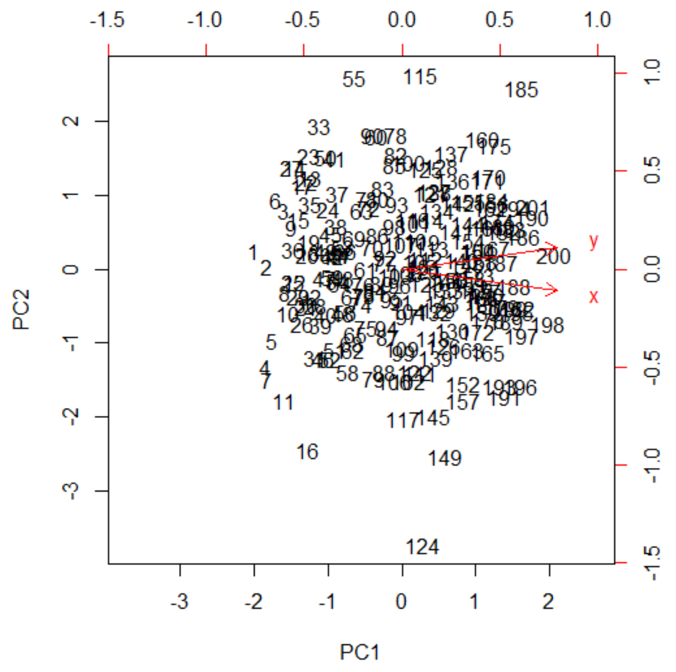
```
> oldpar=par(mfrow=c(1,2))
> plot(x.n,y.n,pch=19,cex=0.7,xlab="Standardized X",ylab="Standardized Y")
> arrows(0,0,px$rotation[1,],px$rotation[2,], length=0.1,angle=20, col="red",lwd=3)
> plot(px$x,pch=19,cex=0.7,xlab="PC1",ylab="PC2",xlim=c(-3,3),ylim=c(-3,3))
> arrows(0,0,px$rotation[,1],px$rotation[,2], length=0.1,angle=20, col="red",lwd=3)
> text(px$rotation[,1]*1.2,px$rotation[,2]*1.2, rownames(px$rotation), col="red")
> par(oldpar)
> px$x[1:5,] %*% t(px$rotation)
             x         y
[1,] -2.024885 -1.955129
[2,] -1.806864 -1.797975
[3,] -1.692818 -1.472201
[4,] -1.638880 -2.005219
[5,] -1.608735 -1.877229
```

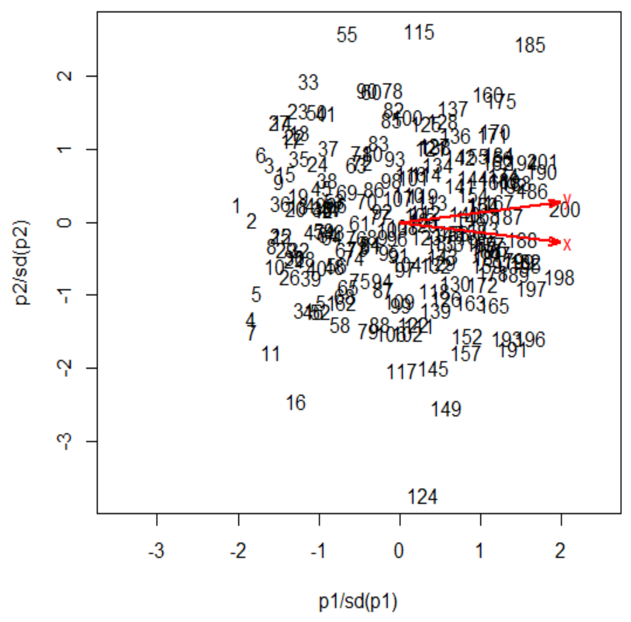


BIPLOT

- Biplot is a special variant of PC projection of the data:
  - The transformed coordinates in PC coordinate system are standardized'
  - The directions of original variables are shown with arrows
  - Below we illustrate this step by step and draw biplot manually

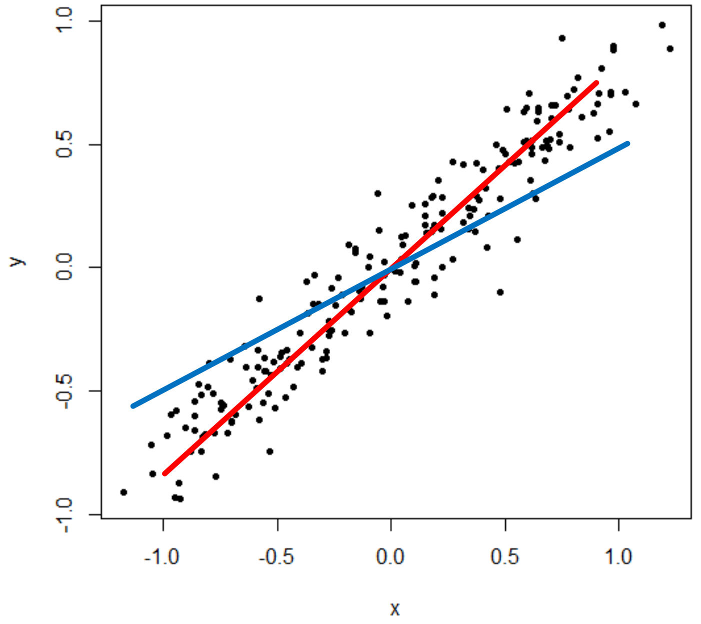
```
> biplot(px,pc.biplot=T)
> p1=px$x[,1]; p2=px$x[,2]
> plot(p1/sd(p1),p2/sd(p2),xlim=c(-3.5,2.5),
    type='n')
> text(p1/sd(p1),p2/sd(p2),1:length(p1))
> arrows(0,0,2*sd(p1)*px$rotation[,1],
      2*sd(p2)*px$rotation[,2],length=0.1,
      angle=20, col="red",lwd=2)
> text(2.1*sd(p1)*px$rotation[,1],
       2.1*sd(p2)*px$rotation[,2],
       c('x','y'),col='red')
```





PC PROJECTION IS A CLOSE REPRESENTATION OF THE DATA

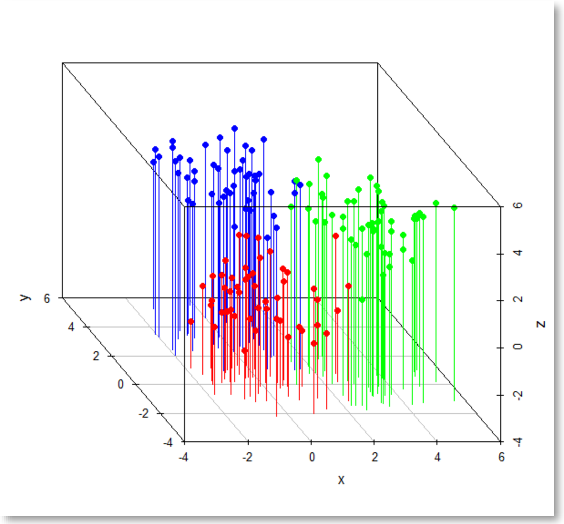
- So far we were describing principal components as “following the variance”
- Equivalent interpretation: the direction(s)/hyperplane defined by first principal component(s) is “the closest to the data”
  - Clearly, we need to closely follow the direction(s) of largest amplitude of variation in the data in order to sty as close to as many data points as possible
  - Red = close to the data (also highest variance)
  - Blue = not as close…

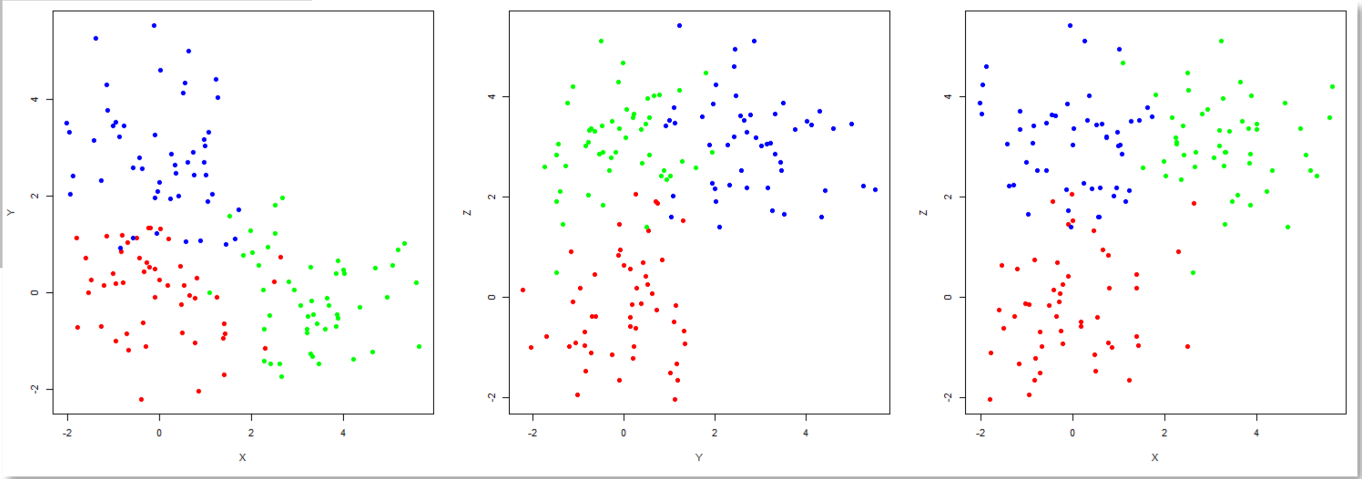


3D EXAMPLE

- Let us perform a little bit more interesting simulation
- Now we have 3 variables, i.e. the data points (observations) “live” in 3D

```
> N=50
> x=c(rnorm(N,0),rnorm(N,0),rnorm(N,3))
> y=c(rnorm(N,0),rnorm(N,3),rnorm(N,0))
> z=c(rnorm(N,0),rnorm(N,3),rnorm(N,3))
> col=c(rep("red",N),rep("blue",N),rep("green",N))
> oldpar=par(mfrow=c(1,3),pch=19,cex=0.7)
> plot(x,y,xlab="X",ylab="Y",col=col)
> plot(y,z,xlab="Y",ylab="Z",col=col)
> plot(x,z,xlab="X",ylab="Z",col=col)
> par(oldpar)
> library(scatterplot3d)
> scatterplot3d(x,y,z,color=col,pch=19,
   cex.symbols=1.5,angle=125,type='h')
```

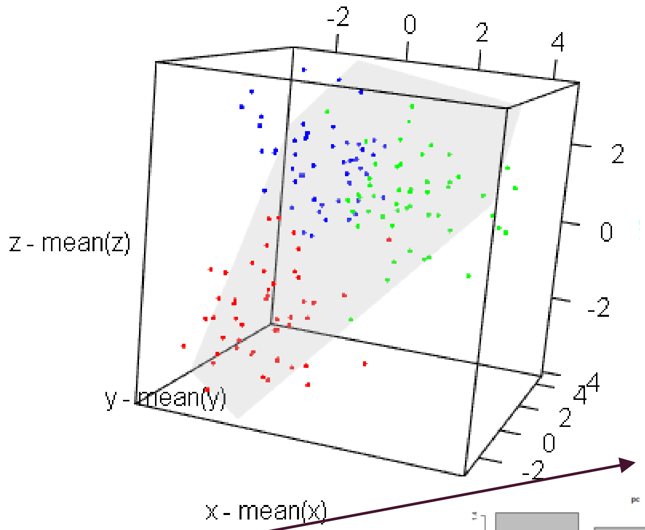


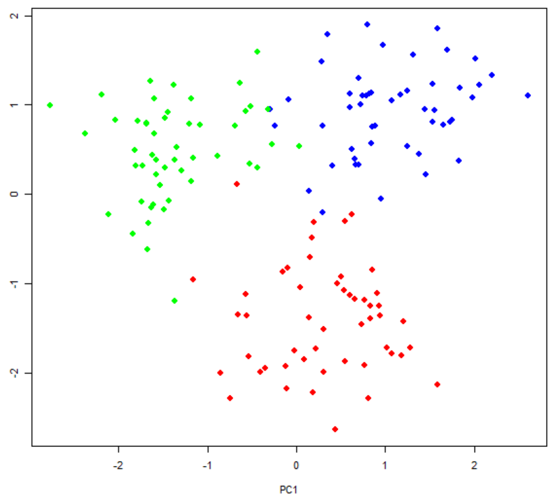


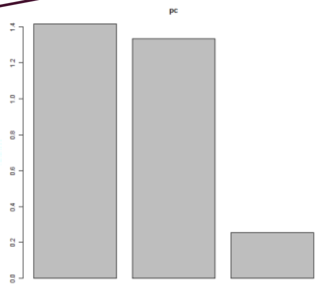
PC1-PC2 IS NOW MAXIMUM VARIANCE PLANE

- PCA finds the plane in which the centers of 3 clusters are located. Depending on the goal of the analysis,
  - We can say that we reduced dimensionality/found the transformed coordinates which represent data well in D=2
  - And/or we can better see the clusters when projected onto the correct hyperplane – if we are after possible structure in the data

```
> pc=prcomp(cbind(x,y,z),scale=T)
> pc$rotation
         PC1       PC2        PC3
x -0.7622305 0.2589493 -0.5932536
y  0.6281541 0.5171899 -0.5813235
z -0.1562915 0.8157572  0.5568780
> library(rgl)
> plot3d(x-mean(x),y-mean(y),
     z-mean(z),pch=19,col=col,cex=1.5)
> planes3d(pc1[2]*pc2[3]-pc2[2]*pc1[3],
   pc1[3]*pc2[1]-pc1[1]*pc2[3],
   pc1[1]*pc2[2]-pc2[1]*pc1[2],
  ambient="grey",alpha=0.3)
> plot(pc$x[,1:2],pch=19,col=col,cex=1.2)
> plot(pc)
```







NCI60 DATASET

- Gene expression profiling data for 60+ tumor samples
  - Any living cell is a fully automated biochemical factory, which constantly makes proteins to support its function
  - There are ~20K genes in human genome, each gene coding for a protein
  - Depending on the cell's type (stem cell, pluripotent cell, fully differentiated tissue cell, tumor cell) and state (amount of nutrients or stress, willingness to send "signals" to and type of "signals" coming from other cells), different subsets of genes are "expressed", in different absolute quantities.
  - Expression levels across all genes characterize a cell very extensively (albeit not completely!)
  - "Central dogma": DNA$\rightarrow$mRNA$\rightarrow$protein. It is mRNA that is measured most often. With modern technologies it is feasible to measure "expression levels", i.e. (relative) amounts of mRNA across all ~20K genes and more. NCI60 is a very "old" dataset (circa 2000) with only ~6800 genes measured.
  - Hence: each sample is an independent "observation". Each gene (mRNA) is a "feature" (random variable). We measure, simultaneously, ~6,800 different features for each observation!

```
> library(ISLR)
> data(NCI60)
```

```
> class(NCI60)
[1] "list"
> names(NCI60)
[1] "data" "labs"
```

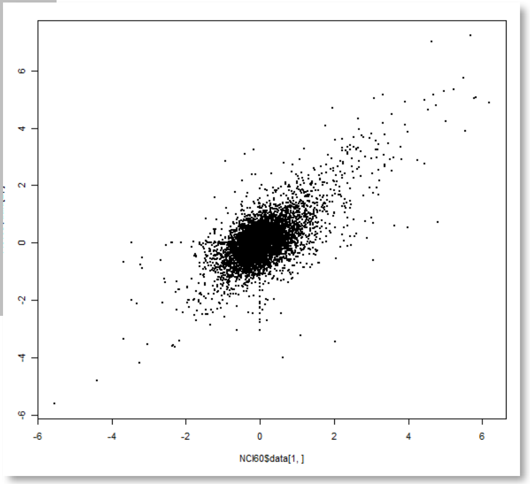
FIRST LOOK AT THE DATA

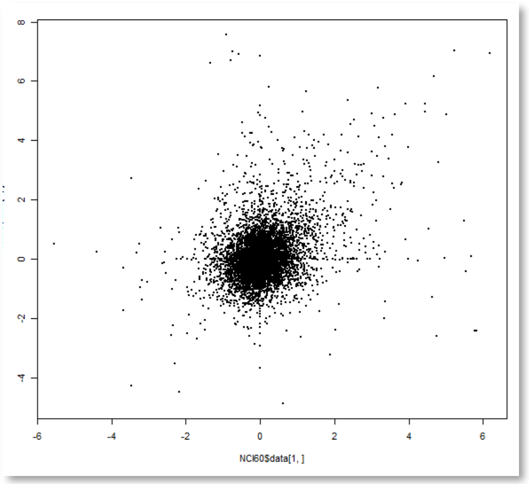
- NCI60 variable is a list
  - $data: matrix of expression values (log sample/control); $lab: sample tumor type

```
> NCI60$data[1:5,1:5]
          1         2         3         4         5
V1 0.300000  1.180000  0.550000  1.140000 -0.265000
V2 0.679961  1.289961  0.169961  0.379961  0.464961
V3 0.940000 -0.040000 -0.170000 -0.040000 -0.605000
V4 0.280000 -0.310000  0.680000 -0.810000  0.625000
V5 0.485000 -0.465000  0.395000  0.905000  0.200000
> dim(NCI60$data)
[1]   64  6830
> NCI60$labs[1:5]
[1] "CNS"     "CNS"     "CNS"     "RENAL"   "BREAST"
>  plot(NCI60$data[1,],NCI60$data[2,],cex=0.5,pch=19)
>  plot(NCI60$data[1,],NCI60$data[4,],cex=0.5,pch=19)
```

Genes (variables)

Actually, "log-ratio"

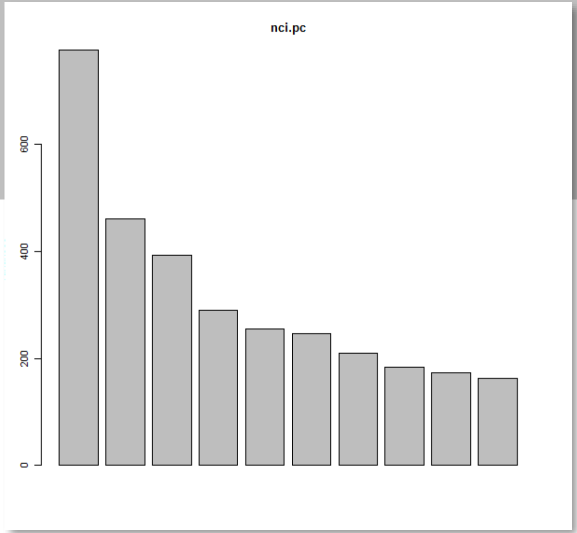




IS THERE ANY STRUCTURE?

- Our first instinct is to run PCA and check for structure in the data (or for potential irregularities!)

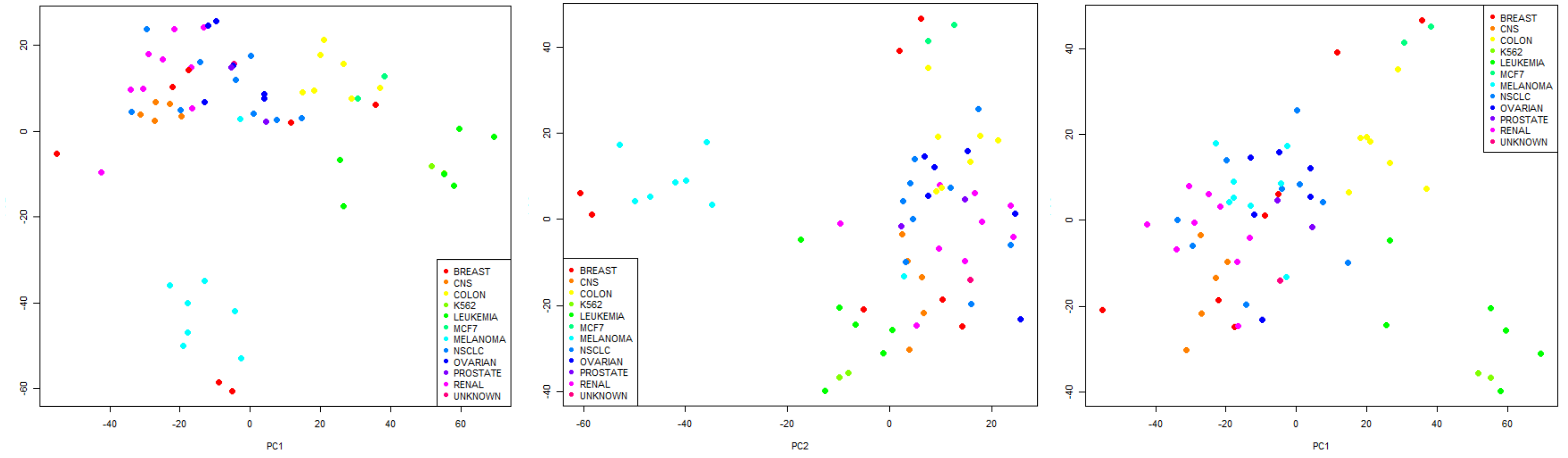
```
> nci.pc = prcomp(NCI60$data,scale=T)
> plot(nci.pc)
> sum((nci.pc$sdev[1:3])^2)/sum(nci.pc$sdev^2) # note how 1st 3 PCs account for ~25% variance
[1] 0.2386699
> sum((nci.pc$sdev[1:10])^2)/sum(nci.pc$sdev^2) # 1st 10 PCs account for ~50% variance
[1] 0.4612564
> unique(NCI60$labs)
 [1] "CNS"          "RENAL"        "BREAST"       "NSCLC"        "UNKNOWN"      "OVARIAN"      "MELANOMA"
 [8] "PROSTATE"     "LEUKEMIA"     "K562B-repro" "K562A-repro" "COLON"        "MCF7A-repro" "MCF7D-repro"
> t.type=sub("[ADB]-.*","",NCI60$labs)
> palette=rainbow(12) # pick 12 colors
> colors=palette[factor(t.type)] # note repeated indexing!
> plot(nci.pc$x[,1:2],pch=19,cex=1.5,col=colors)
> legend("bottomright",col=palette,legend=levels(factor(t.type)),pch=19)
```



Figures in the next slide…

TUMOR TYPES CLUSTER IN FIRST 3 PCS

- Clear separation of (at least some) tumor types is observed in PC1-PC2 and PC2-PC3 plots.

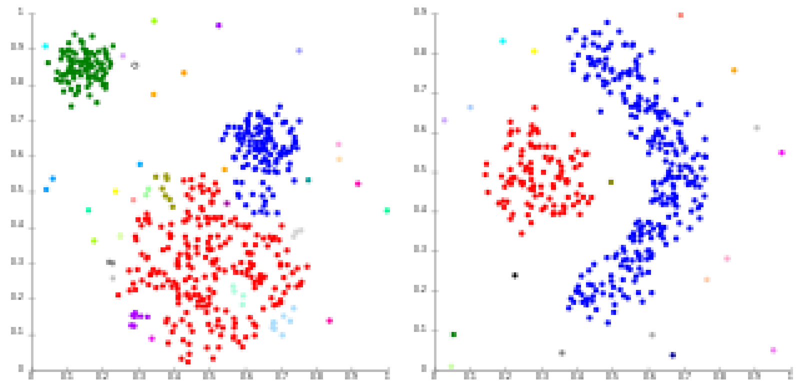


OUTLINE

- Principal Component Analysis (PCA)
- Clustering: K-Means
- Clustering: Hierarchical

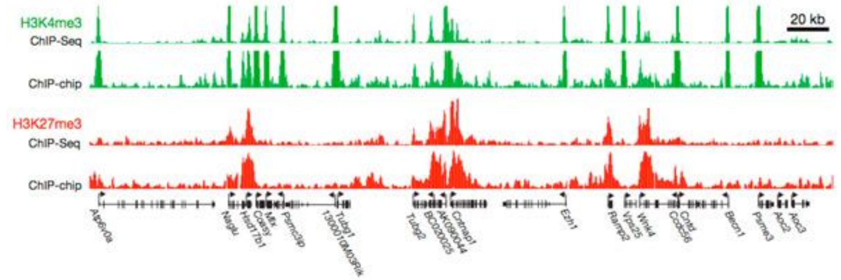
CLUSTER ANALYSIS

- Cluster analysis is a large research and application domain in its own
- The goal is intuitively clear and easy (?) to formulate: in a large group of objects, find subgroups such that the members of the same subgroup are “more similar” or “closer” to each other than to the members of different subgroups
  - Applications: Marketing research, econometrics, sociology (group population/consumers into market segments, social strata etc), biology (evolution, population genetics, high-throughput screening, etc – clustering genomic variants, genes, samples, protein structures), imaging, recommender systems. …
  - Methods: for such a diverse group of applications and questions, many different methods are bound to exist. The differenes encompass both the algorithmic variations as well as different definitions of “similarity” (depending on the definition and structure of the data, some algorithms may not be even applicable)



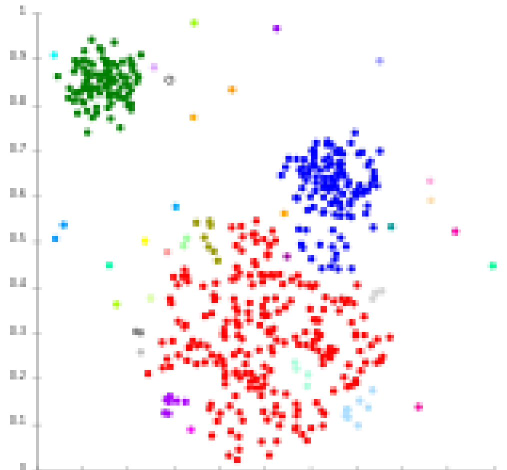
SO WHAT DOES “CLOSE” MEAN?

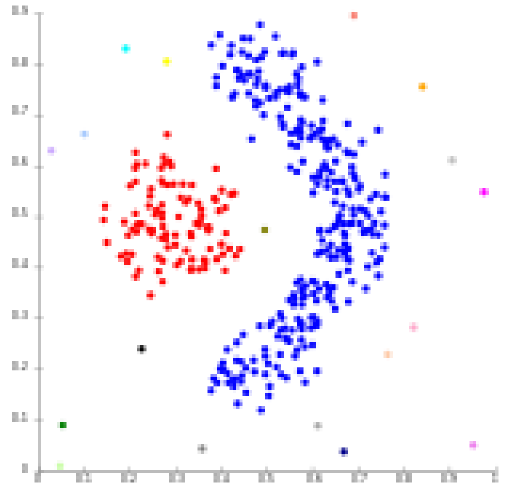
- From staring at 2D plots, we might be compelled to use Euclidian distance:
  - For two points A, B in 2D, the distance $d(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$, or in general case of p dimensions, $d(A,B) = \sqrt{\sum_{i=1}^{p}(x_{Ai} - x_{Bi})^2}$
  - However, we may consider genetic sequences to be close when the count of positions where they differ is small (at each position: same=0, difference=1) → Hamming distance
  - Developing on the previous example, we can improve the analysis by taking into account the fact that some changes in genomic sequence occur much less frequently than others, so we can introduce proper weights (one low frequency change = larger distance than one very frequent change) → evolutionary distance
  - We may call gene expression profiles (across multiple samples/conditions or across different time points) or stock historical profiles “similar” when they strongly correlate → correlation based distance
  - We can call sets or train of peaks more “similar’ when their overlap is larger → Jaccard distance



HOW TO CHOOSE LINKAGE?

- The choice of pairwise distance most appropriate for the problem at hand is very important, of course
- But it is also important how we grow the clusters: the simplest and classical example are elongated/irregular/concave clusters:
- Not every algorithm can deal with the clusters on the right: despite the points on the two ends of the blue cluster are very far from each other, we need to favor the "continuity" of the cluster





K-MEANS CLUSTERING

- One of the simplest clustering methods. Works surprisingly well in *simple* situations
- The assumption: the data form convex clusters ("centroids"); the goal: locate the "centers of mass" of the respective clusters and assign points to the centroid they are the closest to.
- The algorithm requires the researcher to specify the number of clusters K to partition the data into
  - This drawback can be partially ameliorated by repeating clustering with different values of K, then choosing the "best" result (conceptually, very much alike comparing supervised models with different numbers of variables included!)
- If we have p variables, $X_1, \ldots, X_p$, then each observation is a point $(x_1,\ldots,x_p)$ in p-dimensional space.
- Requires Euclidean distance

```
• Randomly assign all $N$ observations to clusters 1 through $K$
• *Iterate* until convergence is achieved:
    • Calculate the mean ("the center") $M_j$ of each cluster $j=1\ldots K$
    • Reassign to each cluster $j$ all the points that are closer to $M_j$ than to the
      center of any other cluster
```

IMPLEMENTING K-MEANS

- Here is a simple implementation of the K-means algorithm (it also shows a movie!)

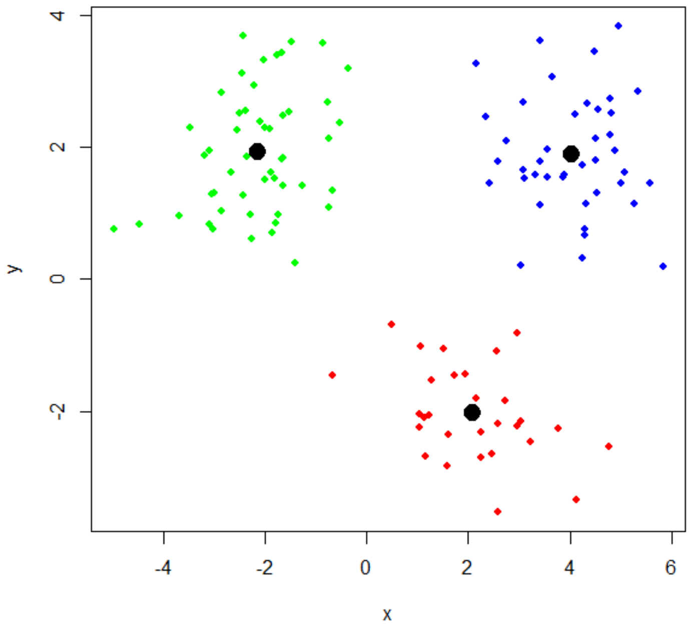
```
km = function(x, K, Nmax=1000, movie=F, sleep=1) {
  clusters = sample(1:K,nrow(x),replace=T)

  cluster.centers=matrix(data=NA,ncol=ncol(x),nrow=K)
  distances=matrix(data=NA,nrow=nrow(x),ncol=K)
  if ( movie ) { colors=rainbow(K) }

  for ( i in 1:Nmax ) {
    for ( j in 1:K ) {
      if ( sum(clusters==j)== 0 ) { next }
      cluster.centers[j,] =
         apply(x[clusters==j, ],2,mean)
      distances[ , j] =
         apply(x,1,function(point) {
            sqrt(sum((point-cluster.centers[j,])^2))})
    }
    if ( movie ) {
      plot(x,pch=19,cex=0.8,col=colors[clusters],
            main=paste("N=",i,", centers",sep=""),...)
      points(cluster.centers,pch=19,cex=2,
               col="black",bg=colors)
      Sys.sleep(sleep)
    }
    new.clusters = apply(distances,1,which.min)
    if ( movie ) {
      plot(x,pch=19,cex=0.8,col=colors[new.clusters],
             main=paste("N=",i,", update",sep=""),...)
      points(cluster.centers,pch=19,cex=2,
             col="black",bg=colors)
      Sys.sleep(sleep)
    }
    if ( sum(new.clusters!=clusters) == 0 ) { break }
    clusters=new.clusters
  }
  cat(i, " iterations performed\n",sep="" )
  return(new.clusters)
}
```

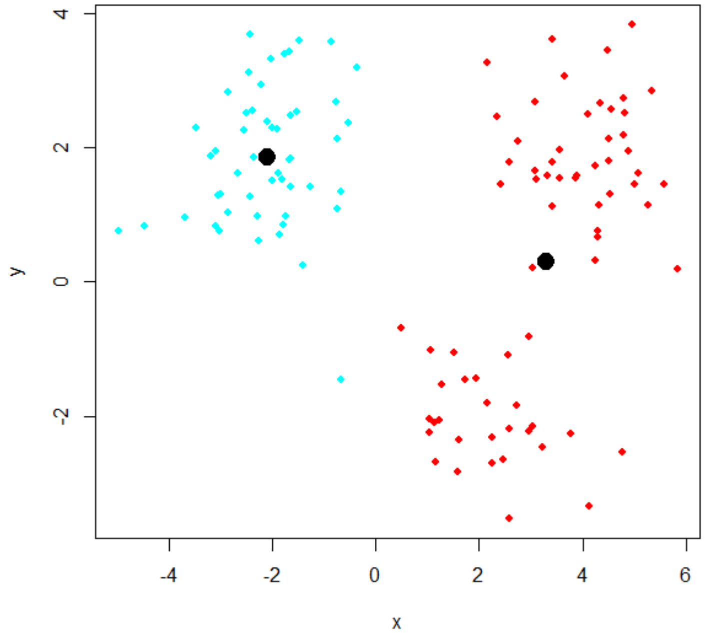
x=observations (obs1_var1 obs1_var2 …
obs2_var1 obs2_var2…
clusters=(cluster_obs1, cluster_obs2, …)
cluster_centers = (center1_var1, center1_var2…
center2_var1, center2_var2,…)
Distances = (obs1_distance_to_center1 obs1_distance_to_center2 …
obs2_distance_to_center1 obs2_distance_to_center2 …

TESTING K-MEANS ALGORITHM

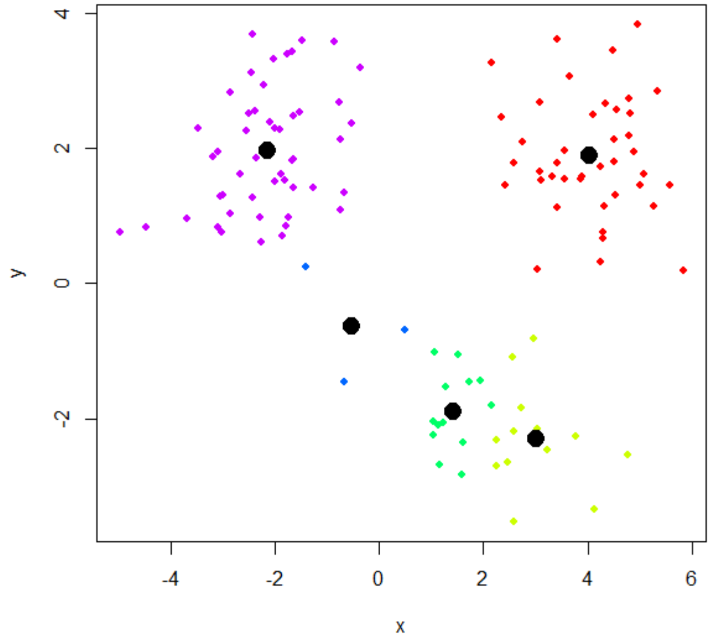
- Generate data with 3 clusters, run with different K:

```
> x=c(rnorm(30,mean=2),rnorm(50,mean=-2),rnorm(40,mean=4))
> y=c(rnorm(30,mean=-2),rnorm(50,mean=2),rnorm(40,mean=2))
> km(cbind(x,y),K=3,movie=T)
```





K=2

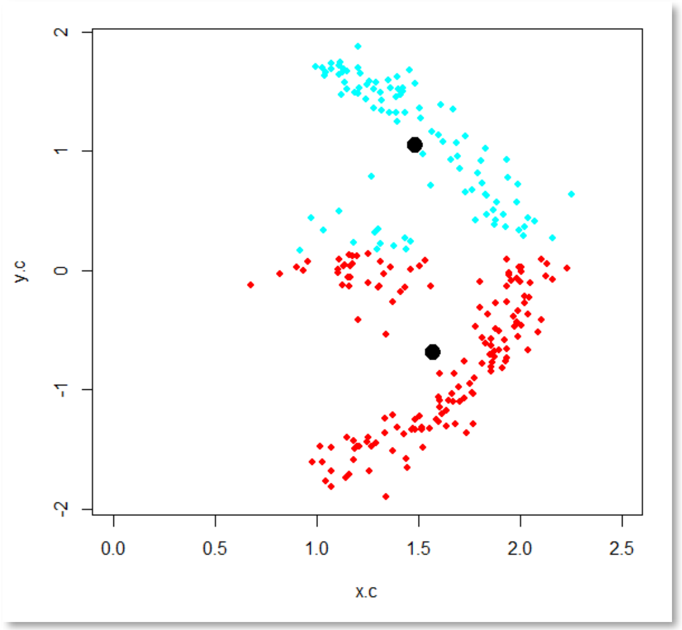


K=5

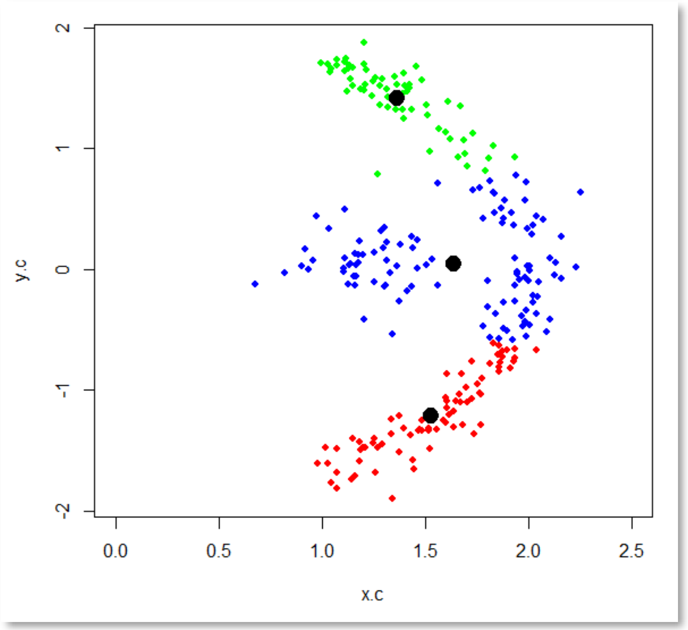
CONCAVE SHAPES: BAD FOR K-MEANS!

- One example of data that K-means cannot deal with:

```
# use polar coordinates:
> a=runif(200,min=-pi/3,max=pi/3)
> r=2+rnorm(200,sd=0.1)
> x.c=c(r*cos(a),rnorm(50,mean=1.2,sd=0.2))
> y.c=c(r*sin(a),rnorm(50,mean=0,sd=0.2))
> km(cbind(x,y),2,movie=T,xlim=c(0,2.5))
```



K=2

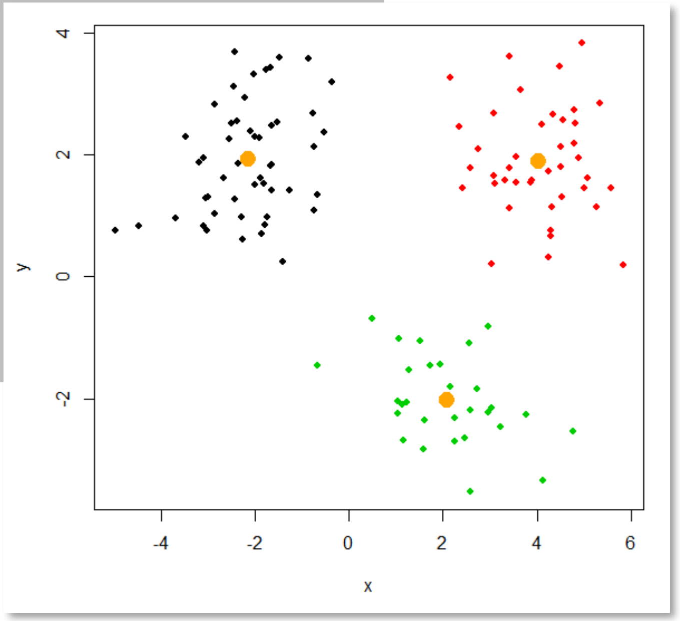


K=3

R HAS KMEANS IMPLEMENTATION TOO!

- R has an implementation that's of course more efficient than our quick code:

```
> kfit.x=kmeans(cbind(x,y),3,iter.max=100,nstart=3)
# how "wide" are clusters compared to how far they are:
> kfit.x$tot.withinss/kfit.x$totss
[1] 0.1401825
>
> kfit.x$cluster
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 ...
>
> plot(x,y,cex=0.9,pch=19,col=kfit.x$cluster)
> kfit.x$centers
          x         y
1 -2.137796  1.922972
2  4.032446  1.897659
3  2.088556 -2.033608
> points(kfit.x$centers,col="orange",cex=2,pch=19)
>
```



Try finding clusters in "raw" NCI60! You won't see much success…

OUTLINE

- Principal Component Analysis (PCA)
- Clustering: K-Means
- Clustering: Hierarchical

HIERARCHICAL CLUSTERING: THE CONCEPT

- Belongs to a class of "agglomerative partitioning" methods: starts from the bottom (completely dissociated data) and progressively assembles points into larger and larger clusters
  - Top-down partitioning methods that start from assigning all the data to a single cluster and then progressively split it into parts also exist but will not be discussed here
- The algorithm starts with each data point in its own "cluster" and proceeds as follows:

> - Assign each point into a separate cluster, so the number of clusters K=N
> - *Iterate* until all data are merged into a single cluster, K=1:
>   - Calculate the pairwise distances between all the K clusters currently held
>   - Find two clusters separated by the smallest distance and merge them; now we have K-1 clusters

- The algorithm keeps and returns the (binary) tree of all merges performed as we will see shortly. That tree represents the hierarchy of distances between the clusters at all levels of partitioning
- We can (and should) examine that tree and decide which K is right.

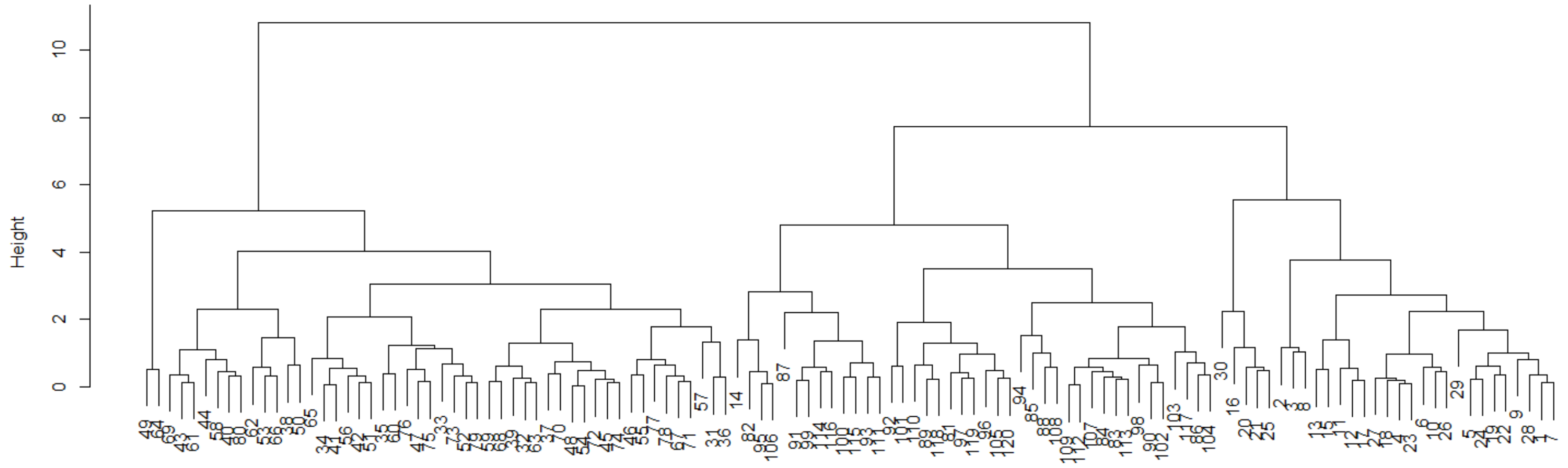
MERGING CLUSTERS

- But how do we decide on the “distance” between clusters?
- At the very first step each cluster is a point, so we can decide on any specific distance between points that is appropriate for the problem in hand, and that will be the distance between the initial clusters
- When we have aggregated clusters, we have another choice to make: *linkage*
  - Single linkage: given two clusters A,B take all pairwise distances d(a,b) between a$\in$A and b$\in$B. Define d(A,B) as the *smallest* of such distances (i.e. distance between the clusters is the distance between their “edges” facing each other)
  - Average linkage: take all pairwise distances d(a,b) between a$\in$A and b$\in$B and define d(A,B) as the average of all those distances. Good and robust distance: two clusters must be close to each other as a whole in order to have small average *d*. Note that this metric might be not the best for very elongated, curved and in general asymmetric clusters (single linkage might work better for those)
  - Complete linkage: out of all d(a,b) where a$\in$A and b$\in$B, the distance d(A,B) is defined as the *largest* among them. The distance between two clusters is the distance between their “outward-facing” edges
  - Ward distance: not a distance strictly speaking (cannot be written down in a closed form), but a procedure: all pairs of clusters are examined, and a pair is chosen such that the increase in the within-cluster variance is the smallest out of all possible choices. In other words, the Ward clustering method explicitly tries to keep the clusters as “tight” as possible at every iteration of the algorithm. Strictly speaking, Ward requires the underlying metrics to be Euclidean.

HIERARCHICAL CLUSTERING: SIMPLE EXAMPLE

- Let us first cluster the data we simulated earlier. All it takes is calculating the distance (will be saved in a special distance object returned by function `dist()` and passing that distance object to built-in function `hclust()`.
  - `dist()` has optional "method" argument that defaults to "euclidean"
  - The height of the branch reflects the distance between clusters: the leaves down below are very close and merged first; some resulting clusters are still close and merged soon after. The structure of the tree suggests 3-4 clusters

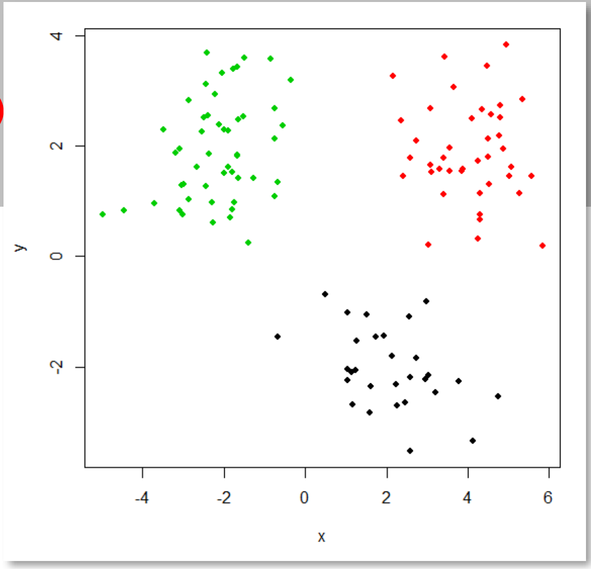
```
> d=dist(cbind(x,y))
> plot(hclust(d))
```

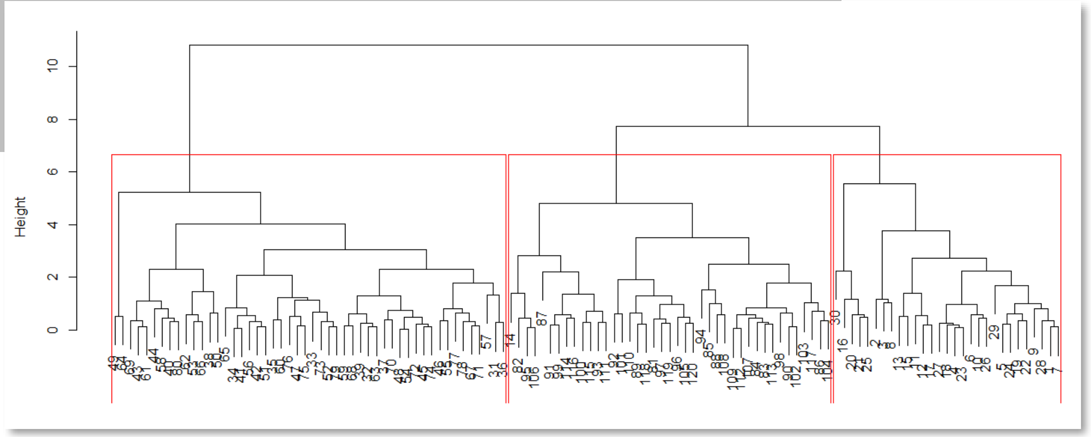


CUTTING THE TREE

- We can make a specific cut through the tree and assign clusters
  - Can request specific height or specific number of clusters – optional args `k` and `h` to `cutree()` and `rect.hclust()`

```
# returns the clusters in the order the points where submitted to dist/hclust,
# not the order of the leaves!
> cutree(hclust(d),k=3)
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 …
> plot(x,y,col=cutree(hclust(d),k=3),pch=19,cex=0.8)  # visualize the clusters
# visualize the cut:
> plot(hclust(d))
> rect.hclust(hclust(d),k=3)
```

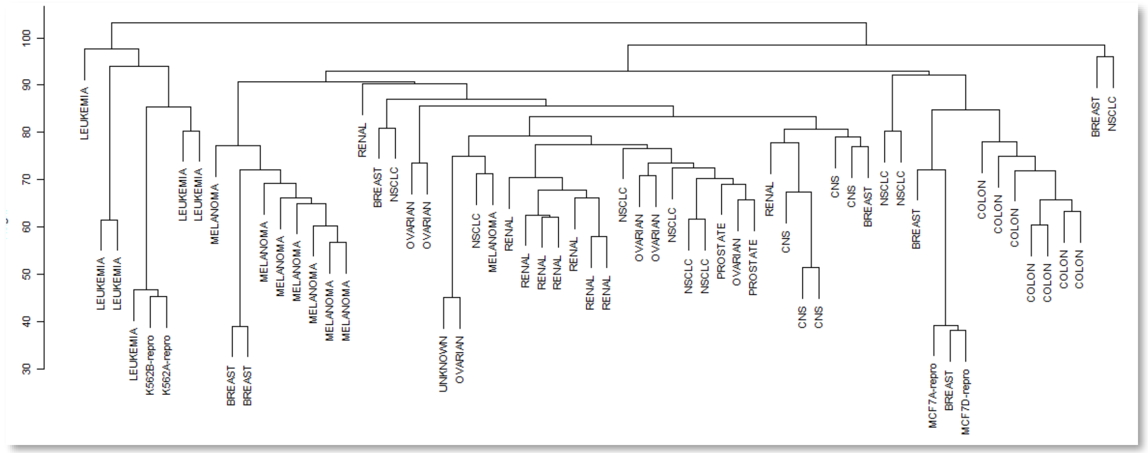


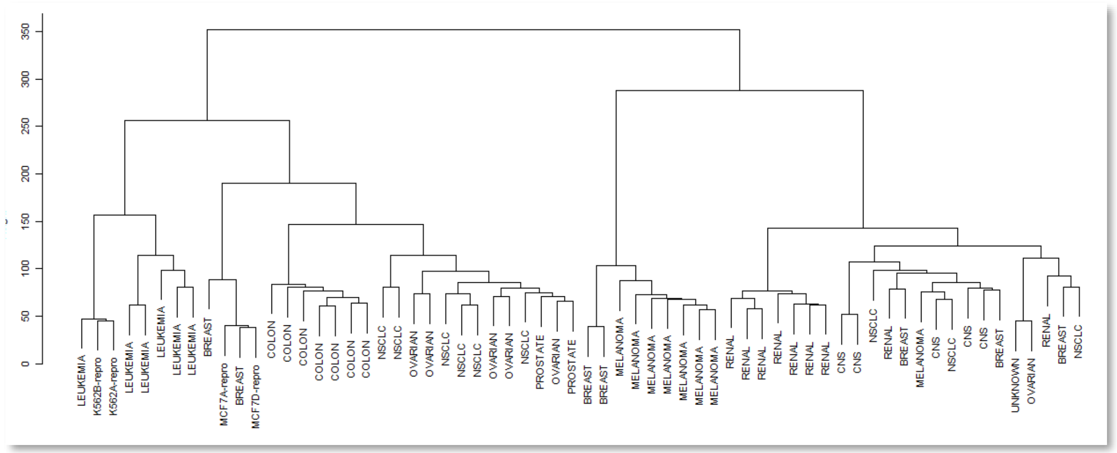


CAN WE HANDLE A REAL DATASET?

- Let's try NCI60! We will be using just the Euclidean distance for now. We will talk more about distances next week!

```
# to have leaves automatically annotated:
> rownames(NCI60$data)=NCI60$labs
> plot(hclust(dist(NCI60$data),method="average"))
> plot(hclust(dist(NCI60$data),method="ward.D"))
```





A TEMPERED APPROACH TO INTERPRETING THE RESULTS OF CLUSTERING

- "clustering can be a very useful and valid statistical tool if used properly"
- "small decisions in how clustering is performed, such as how the data are standardized and what type of linkage is used, can have a large effect on the results"
- "we recommend performing clustering with different choices of these parameters, and looking at the full set of results in order to see what patterns consistently emerge"
  - This is contrary to oft cited advice to "choose a single correct clustering method and not evaluate any others"
  - That is very easy to abuse by trying multiple methods and justifying the one giving best results in hindsight
- "we recommend clustering subsets of the data in order to get a sense of the robustness of the clusters obtained"
  - I.e. "consensus clustering" averaging results over random samples variables and observations (e.g. ConsensusClusterPlus)
- "must be careful about how the results of a clustering analysis are reported"
  - "should not be taken as the absolute truth about a data set"
  - "they should constitute a starting point for the development of a scientific hypothesis and further study"
    - "preferably on an independent data set"

ISLR Ch.10.3.3

SUMMARY

- Unsupervised learning methods aim at finding “interesting” structure in the data, in the absence of known outcome
- PCA works both as a dimensionality reduction method AND a method for unsupervised exploration of the data in its potentially “most interesting” projection
- Clustering methods aim specifically at recognizing groups of “close”, or “similar” objects in the data
  - Many different algorithms, we considered only K-means and Hierarchical clustering
  - Many choices for the “distance” between data points are possible, we should choose the one that appropriately reflects the concept of “similarity” between points/observations in the problem at hand
  - Hierarchical clustering also offers few different options for defining the distance between clusters and thus different orders of murging.