Handout 4

Stefan Zohren

22 February 2016

Introduction to unsupervised learning

So far we looked at regression models, classification models, as well as model selection and regularisation. In all cases we had labelled data, consisting of a response variable and p predictors:

$$\text{labelledData} = (\mathbf{y}, \mathbf{x}_1, ..., \mathbf{x}_p)$$

where each entry is a column vector of the size of the number of observations n.

All the above learning algorithms fall under the category of **supervised learning**. In particular, we use the data for which we know the response variable to train a model which can predict the response variable for a point in predictor space for which the response is not known. In the case, we are provided with data for which we know the response variable we speak of **labelled data**.

In this handout we focus on what is known as **unsupervised learning**. In unsupervised learning we have **unlabelled data**, i.e. we simply have:

$$\text{unlabelledData} = (\mathbf{x}_1, ..., \mathbf{x}_p)$$

Instead of training a model to predict a certain response, we analyse models which find certain structures within the unlabelled data.

Our first model is **principal component analysis (PCA)** which aims to find a small number q of predictors $(\mathbf{u}_1, ..., \mathbf{u}_q)$, with $q \leq p$, where each predictor is a combination of the previous predictors, in such a way that those new predictors explain most of the variance of the data.

Our second class of models are **clustering algorithms**, which aim to cluster points in predictor space, thus being able to label them according to their cluster labels.

Principal component analysis

Principal component analysis (PCA) can be understood as a dimensional reduction of the data. This can be very useful for high-dimensional data, such as in genomics, to reduce the numbers of predictors. It can also be useful for visualisation. For example, we could have a higher-dimensional predictor space and would like to plot a two-dimensional projection of the observations in this space. PCA provides us with a way to find the directions of the two-dimensional space to project on, where the data varies most.

Let us illustrate this in a concrete example with `R` which should make the principal clear. For simplicity we look at a two-dimensional space which we would like to reduce to one dimension, i.e. p=2 and q=1. Let us sample n=20 observations in this space, where for simplicity of the discussion we consider predictors which have mean 0 and standard deviation 1.

```
set.seed(10)
x1 <- rnorm(20)
x2 <- rnorm(20) + 2*x1
data <- data.frame(matrix(c(x1,x2),ncol=2))
```

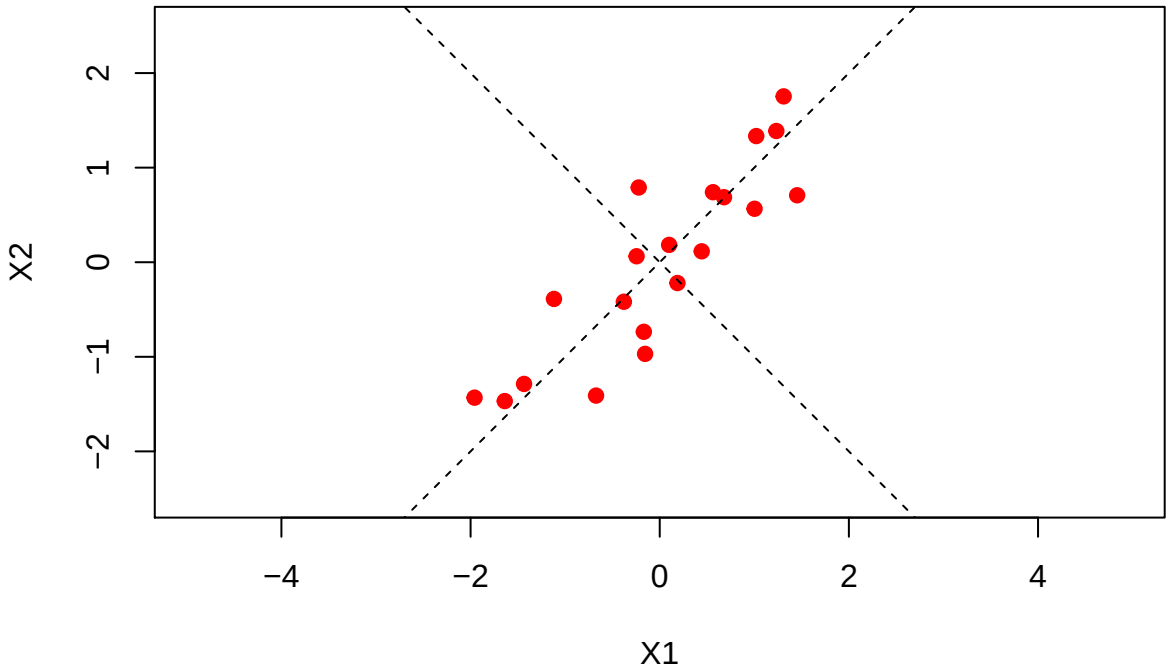
We use our own function `standardise` defined as

```
standardise <- function(x,t.means,t.sds) {
  (x - rep(t.means, rep.int(nrow(x), ncol(x))))/
    rep(t.sds, rep.int(nrow(x), ncol(x)))
}
```

to standardise the data:

```
t.means = apply(data,2,mean)
t.sds = apply(data,2,sd)
data1 <- standardise(data,t.means,t.sds)
```

The data points are shown below. Note that both `x1` as well as `x2` are predictors. There is no response variable here.



The plot also shows the directions of the so-called **loading vectors** of the first and second principal component as dashed lines. Let us explain this in more detail. The loading vector of the first principal component shown as the dashed line going from bottom left to up right, is the direction in which the data varies most. In other words, we are looking for the loading vector spanning a one-dimensional space for which when we project the data onto this space it has the biggest variance. Let us denote

$$z_{i1} = u_{11}x_{i1} + ... + u_{p1}x_{ip} = \sum_{j=1}^{p} u_{j1}x_{ij}, \quad i = 1, ..., n$$

where $\mathbf{u}_1 = (u_{11}, ..., u_{p1})$ is the loading vector. Here $\mathbf{z}_1$ is the first **principal component**. Since the loading vector only describes a direction, it is represented by a normalised vector i.e. $||\mathbf{u}_1|| := \sum_{j=1}^{p} u_{1j}^2 = 1$ (we do not need the length of a vector to know its direction, hence we can set it to one). Saying that the first principal component corresponds to the loading vector in which direction the data varies most is equivalent to finding the normalised vector $\mathbf{u}_1$ which maximises the sample variance of z_i, i.e. $\text{Var}(\mathbf{z}_1) = \frac{1}{n}\sum_{i=1}^{n} z_{i1}^2$. Note that the latter is exactly the sample variance, since $\text{Mean}(\mathbf{z}_1) = 0$, which follows from the fact that $\text{mean}(\mathbf{x}_i) = 0$ for all i=1,. . . ,n. Mathematically, the loading vector $\mathbf{u}_1$ is thus the solution to the optimisation problem

$$\max_{\mathbf{u}_1} \left(\frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} u_{j1} x_{ij} \right)^2 \right), \quad \text{subject to } ||\mathbf{u}_1|| = 1.$$

The second principal component is then found by the loading vector in which direction the variance of the data is maximised and which is uncorrelated to the first principal component. To be uncorrelated the second loading vector must be orthogonal to the first loading vector. Generally speaking the loading vector of the m-th principal component must be orthogonal to the loading vector of the first m-1 principal components:

$$\max_{\mathbf{u}_m} \left(\frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} u_{jm} x_{ij} \right)^2 \right), \quad \text{subject to } ||\mathbf{u}_m|| = 1 \text{ and } \mathbf{u}_m \perp (\mathbf{u}_1, ..., \mathbf{u}_{m-1}).$$

Let us determine the principal components for the above example using `R`. The function for PCA is called `prcomp`:

```
pca <- prcomp(data1)
```

We can look at the loading vectors through the attribute `rotations`

```
pca$rotation
```

```
##           PC1        PC2
## X1 -0.7071068 -0.7071068
## X2 -0.7071068  0.7071068
```

Here the first column is the loading vector $\mathbf{u}_1$, while the second column is the loading vector $\mathbf{u}_2$, each spanning one of the dashed lines.

We can also obtain the standard deviations of the principal components, i.e. $(\text{sd}(\mathbf{z}_1), \text{sd}(\mathbf{z}_2))$

```
pca$sdev
```

```
## [1] 1.3679868 0.3586254
```

or the variances of the principal components by squaring it:

```
pca.var <- pca$sdev^2
```

Note that both of them sum to

```
sum(pca.var)
```

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

This is because it is equal to the variance of our data set which is

$$Var(data) := Var(\mathbf{x}_1) + ... + Var(\mathbf{x}_p) = Var(\mathbf{z}_1) + ... + Var(\mathbf{z}_p).$$

We see that the variance of the data is divided into the variances of the principal components, in decreasing order, starting from the first principal component. It thus makes sense to look at the **proportion of variance explained (PVE)** by each principal component. We can calculate it as

```
pca.pve <- pca.var/sum(pca.var)
pca.pve
```

```
## [1] 0.9356939 0.0643061
```

Thus we see that the first principal component already explains around 93 % of the variance of the data (the observations for all predictors). Instead of using the two predictors, we could simply construct a new predictor, our first principal component, which already explains most of the variance. We could thus reduce our problem from two dimensions to one dimension. Here this is not so surprising since both predictors were correlated by construction. In higher-dimensional examples it is instructive to plot the PVE and see after which number of principal components the new PVE becomes small. Say if we have 100 predictors, but already the first 5 principal components explain 95 % of the variance of the data, we could simple work with the reduced problem where we have only 5 predictors given by the first 5 principal components.

Exercise: Repeat the above analysis. Try to reproduce the above plots as well.

Case study: NCI60 data

The NCI60 cancer cell line microarray data, is a data set included in the package `ISLR` of your text book An Introduction to Statistical Learning. It is also discussed in the computer labs in Chapter 10 of the text book.

```
library(ISLR)
nci.data <- NCI60$data
dim(nci.data)
```

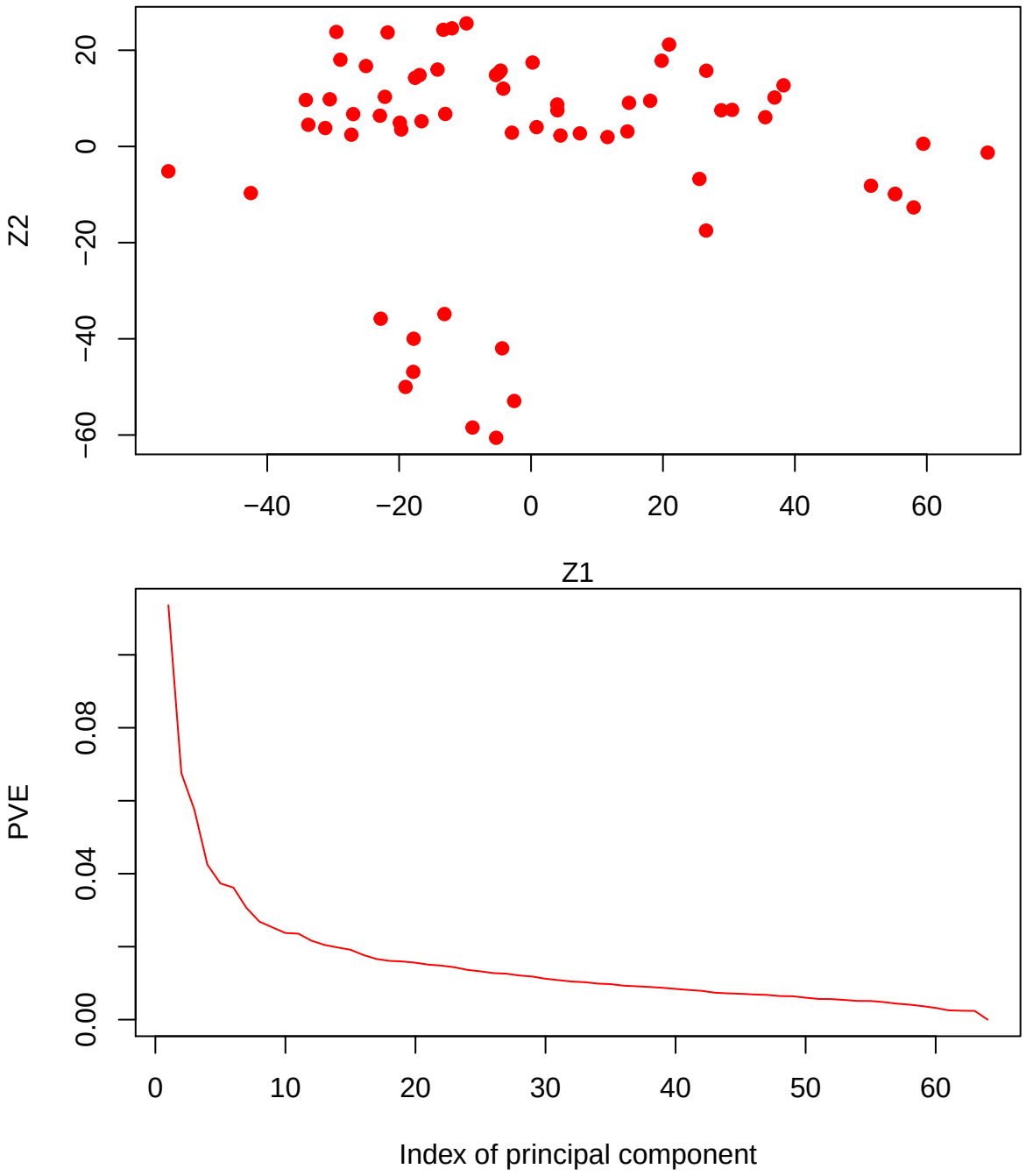
```
## [1]   64 6830
```

It has n = 64 observations and p = 6830 predictors. We also standardise the data. Note that we do this by hand using the function `standardise` which we defined above:

```
t.means = apply(nci.data,2,mean)
t.sds = apply(nci.data,2,sd)
nci.data <- standardise(nci.data,t.means,t.sds)
```

In case our data would not be standardised, we could do the PCA using additional attributes `center=TRUE,scale=TRUE` inside of `prcomp`, but we prefer to rescale it by hand since we need it in standardised form for later analyses as well.

Exercise: Do a PCA on the NCI60 cancer cell line microarray data. Note that you should standardise the data. Plot the data with respect to the loadings of the first two principal components. Also provide a plot of the PVE. Your plots should look like these:



Optional project: Principal component regression and classification

Having reduced the unlabelled data (predictors) to a space of lower dimension, we can use this smaller number of principal components as new predictors for regression or classification - if we have a response variable in addition (which one ignores during the PCA).

We can apply this to the Arcene data set which we discussed in the case study of handout 3.

Optional Exercise Perform a PCA on the predictors of the Arcene data set. Plot the PVE to obtain a reduced number of principal components.

Optional exercise: Perform a logistic regression on the Arcene data set, but now on the reduced set of principal components. (Hints: Be careful to standardise training and test sets separately, use our `standardise` function to do so, where the mean and standard deviation should always be taken from the training set. Use the loading vectors from the training set to construct your new predictors both for the training and test set (i.e. do not perform a new PCA on the test set).)

Clustering algorithms

In **principal component analysis (PCA)** our aim was to **obtain a lower-dimensional representation** of the unlabelled data. This reduced data can then be used for visualisation or as inputs to other supervised learning algorithms which previously suffered from the high-dimensional nature of the data. Alternatively, it can also be used for compression.

In **clustering algorithms** we aim to **assign labels to data** according to some kind of similarity measure of different points. If some group of points is 'similar', then we assign a given label to them. This could for example be used to cluster customers of different kinds depending on their shopping patterns. If some customers of a given cluster then show interest in a new product, we could send out recommendations to the other customers in the same cluster.

K-means clustering

In K-means clustering we assign a given number of labels to the unlabelled data. To formulate precisely how the labels are assigned, let us introduce the within cluster variance. For a given cluster, we define

$$W(C_k) = \frac{1}{|C_k|} \sum_{m,n \in C_k} \sum_{j=1}^{p} (x_{mj} - x_{nj})^2$$

Here C_k is the set of points in the k-th cluster and $|C_k|$ is the number of points in this set. Given a predefined number of clusters K, we aim to find the choice of disjoint clusters whose union is the entire set of points and which minimise the sum over within cluster variances:

$$\min_{C_1,...,C_k} (W(C_1) + ... + W(C_K))$$

Solving this optimisation problem is very difficult since it involves a combinatorial search over all possible subsets, but there exists an approximate algorithm which is guaranteed to find a local minimum, which often coincides with the global minimum of the above optimisation problem.

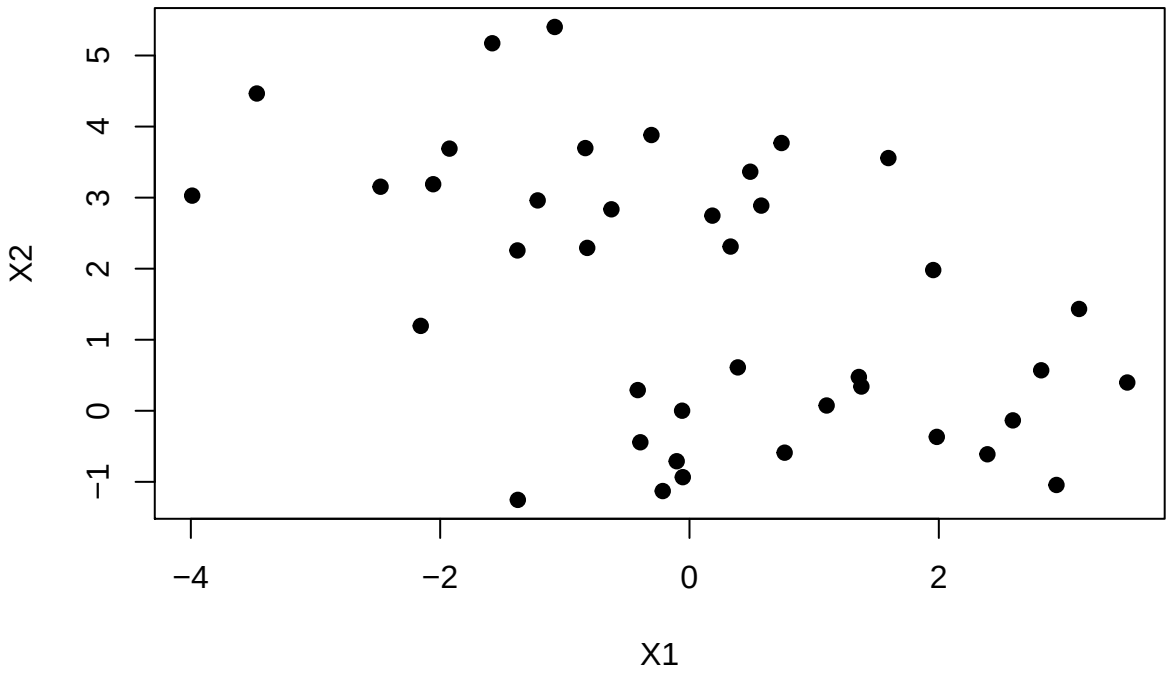
The algorithm works as follows:

- Label the points randomly with K different labels
- Iterate the following until convergence:
  - Calculate the centroid of each cluster (centre of mass)
  - Assign each point to the closed cluster

Since the algorithm converges to a local minimum, we can repeat it for a number of random starting sets and choose the one with the smallest objective function.

Let us illustrate this with a toy example of simulated data

```
set.seed(1)
x1 <- c(rnorm(10),rnorm(10,mean=2),rnorm(10,mean=-2),rnorm(10))
x2 <- c(rnorm(10,mean=3),rnorm(10),rnorm(10,mean=3),rnorm(10))
data2 <- data.frame(matrix(c(x1,x2),ncol=2))
plot(X2~X1,data=data2,pch=19)
```

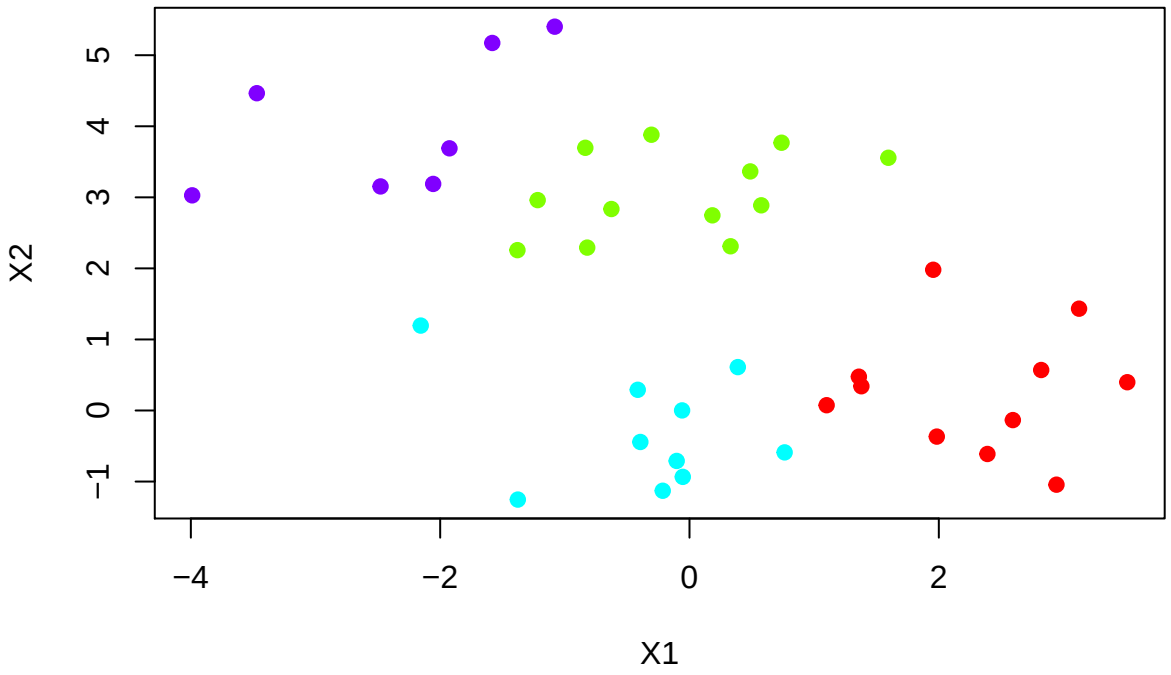


The K-means clustering algorithm is implemented with the function `kmeans`. Let us perform a clustering with K=4 clusters:

```
K=4
clust1 <- kmeans(data2, K ,nstart = 10)
```

We extract the cluster labels of each point using the attribute `cluster`.

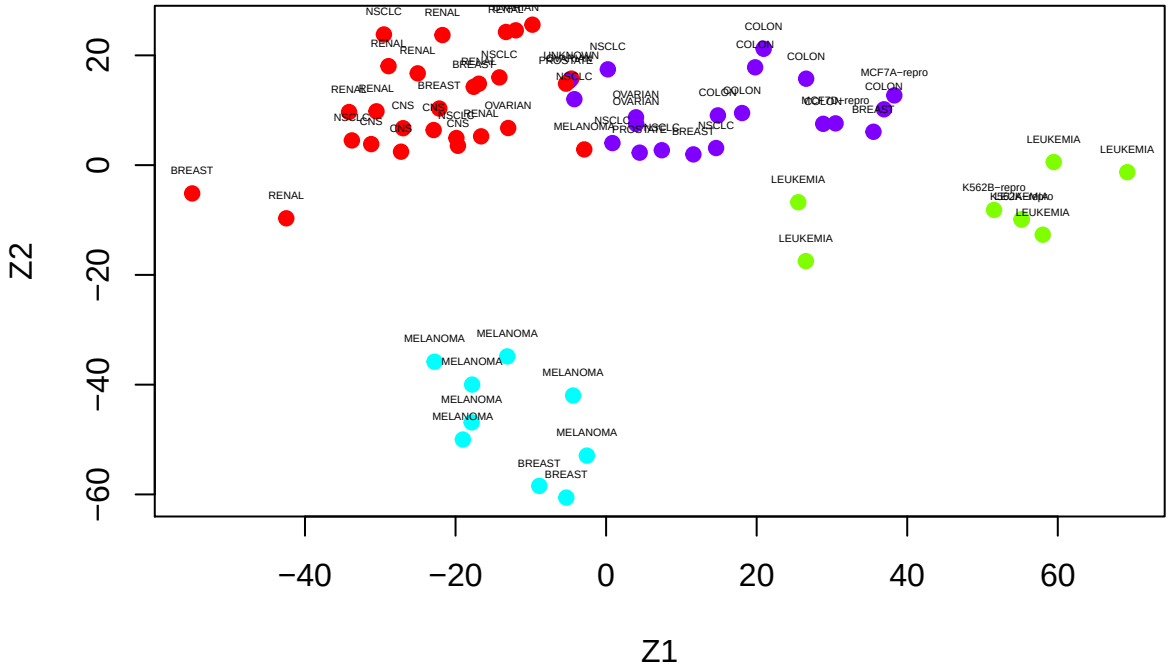
Exercise Perform K-means clustering on this toy data for K=2,3,4 and plot the results as coloured points. Below you see an example of the corresponding plot for K=4.



Case study: NCI60 data (continuation)

We can now also return to the cancer cell line microarray data.

Exercise: In fact, the points in the cancer cell line microarray data correspond to different types of cancer. Perform a K-means clustering with K = 4 clusters. Plot the data on the two-dimensional representation given by the first two principal components showing each cluster as a different colour (use the command `rainbow`).



The NCI60 data set comes with the true cancer labels which are provided in `NCI60$labs` and which we also added to the plot using the `text` function. We see that the clustering of the K-means algorithms

sometimes coincides with the grouping due to the class labels (cancer types), in particular for MELANOMA and LEUKEMIA cancer.

Hierarchical clustering

Hierarchical clustering, as the name suggests, produces a whole hierarchy of clusters. In particular, it produces clusters of size n, n-1, . . . , 1. In a bottom-up approach starting from each observation being its own cluster and then merging pairs of clusters.

In a nutshell, the **algorithm for hierarchical clustering** works as follows:

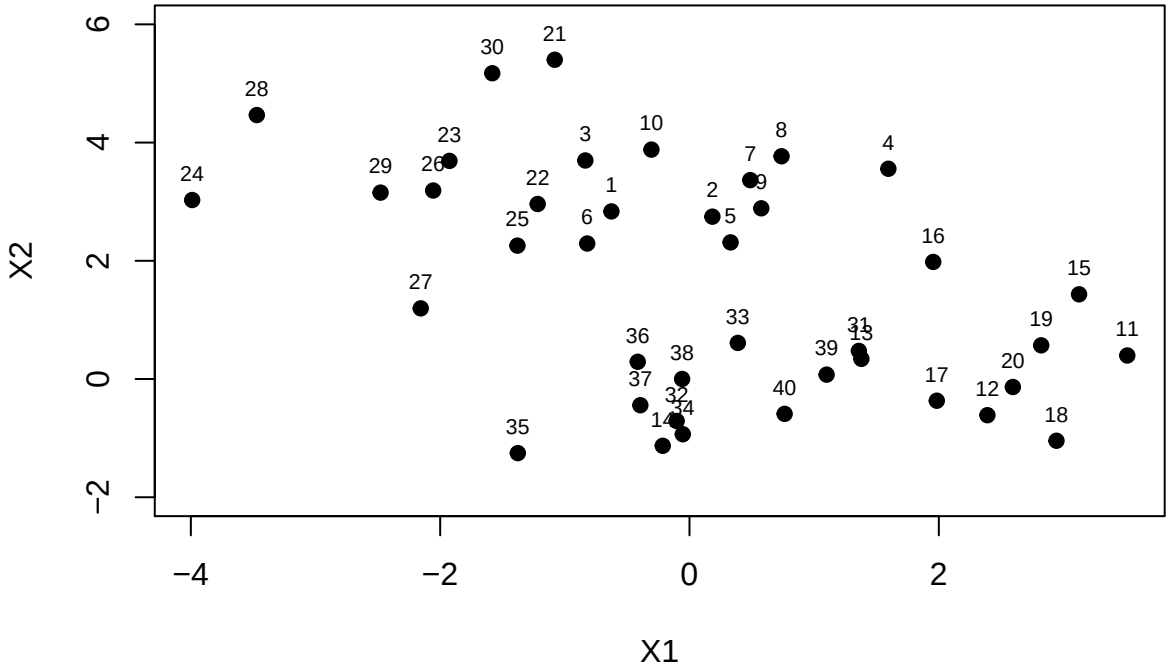
- Start with each observation being its own cluster (we have n clusters);
- Given a similarity measure between pairs of clusters (based on distance), merge the pair of clusters which are most similar (we now have n-1 clusters);
- Continue to merge until we have a single cluster.

Note that the sequence of pairwise mergers can be represented by a rooted **binary tree**. The tree has n leaves corresponding to the start, where we had n clusters (every point) and the root corresponds to the case where all data points are part of a single cluster. The internal nodes of the tree correspond to mergers. The tree is a purely topological object, but we can in addition draw the height of a merging node at the level of combined similarity which was achieved by the merger. The resulting binary tree with scaled height is called a **dendrogram**.

Let us return to the toy example of simulated data from the previous section, saved under `data2` and illustrate the above. The hierarchical clustering is performed as follows:

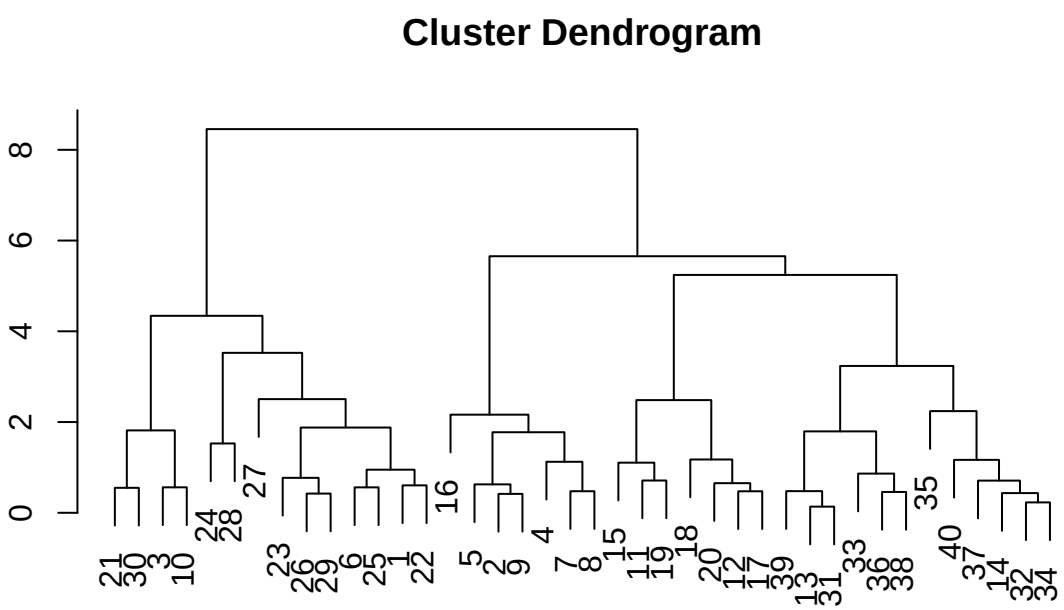
```
clust2 <- hclust(dist(data2))
```

Let us plot again the data, now labelling every data point for i=1,. . . ,20.



We can also plot the dendrogram by simply calling the standard `plot` function on the object created by the `hclust` function:

```
plot(clust2,xlab="", sub="",ylab="")
```



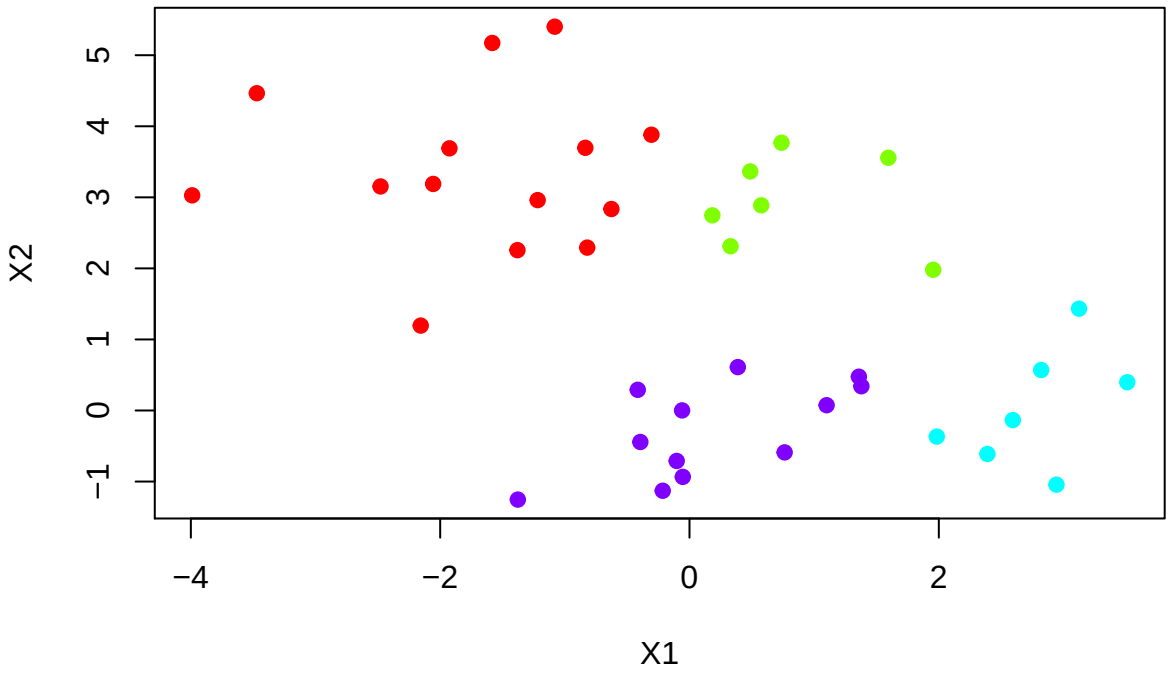
We start reading the dendrogram from the bottom: First points 13 and 31 are merged, then 32 and 34, etc. We see in the scatter plot above that those were the points which were closest to each other.

To obtain a clustering of a given number of clusters K, we can cut the tree at a level where it has K branches. This is done using `cutree`

```
K=4
cutree(clust2,K)
```

```
##  [1] 1 2 1 2 2 1 2 2 2 1 3 3 4 4 3 2 3 3 3 3 1 1 1 1 1 1 1 1 1 1 4 4 4 4 4
## [36] 4 4 4 4 4
```

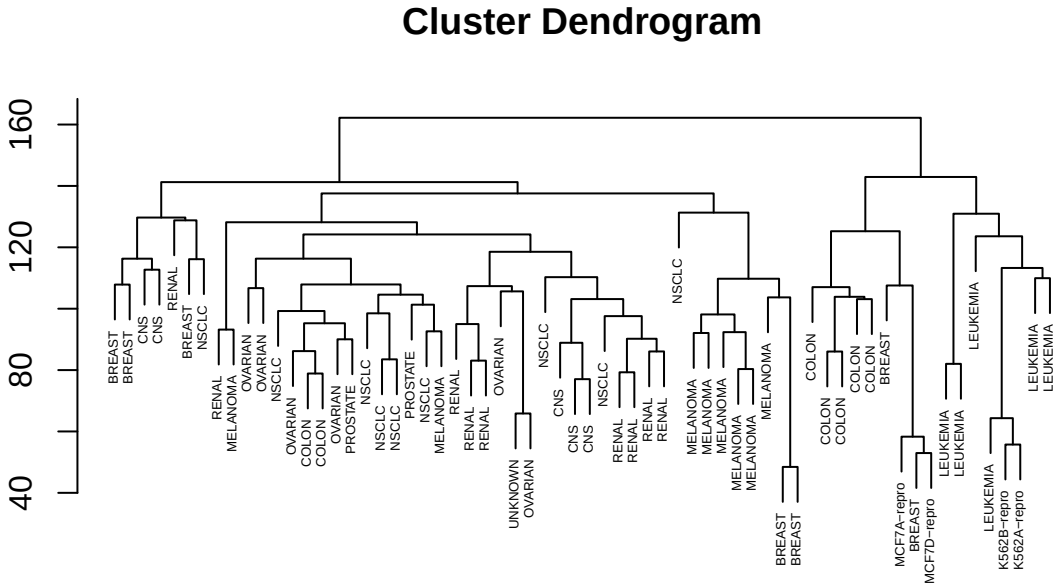
The resulting cluster labels are shown in the figure below. The clustering is slightly different from that obtained using K-means clustering.



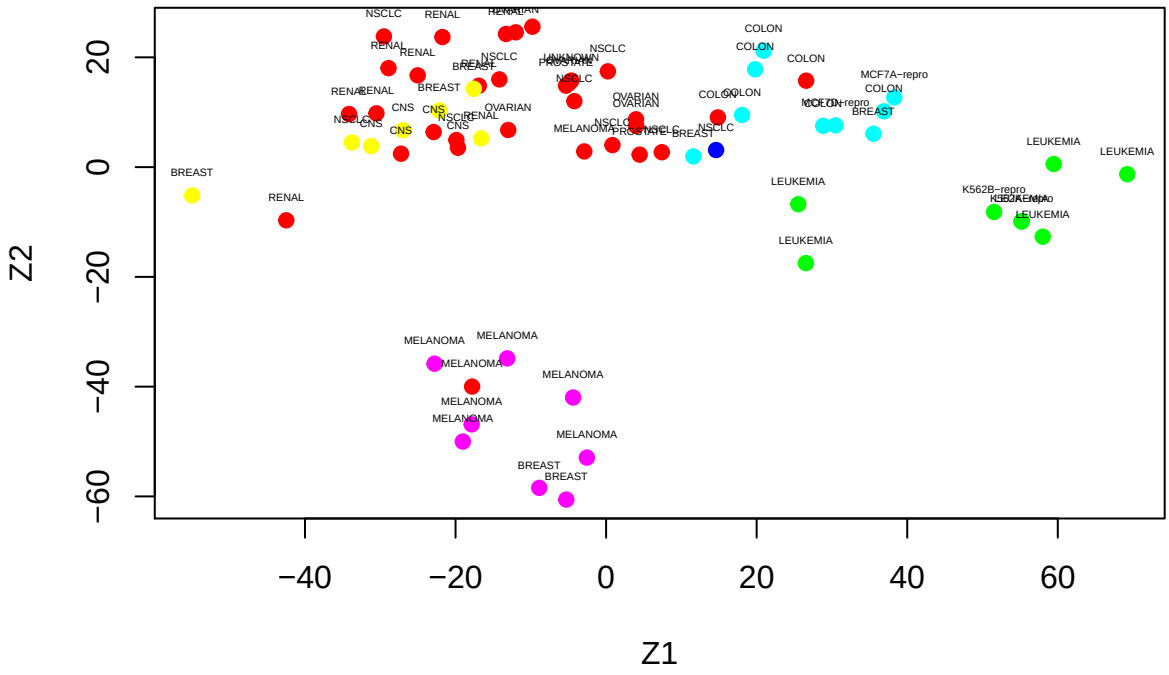
Exercise: Repeat the above analysis and produce the plots in this section. You should also try different similarity measures which are available in the `hclust` function.

Case study: NCI60 data (continuation)

Exercise: Do a hierarchical clustering on the cancer cell line microarray data. Plot the dendrogram and add labels to the leaves, corresponding to the true cancer types `labels=NCI60$labs`. Your plot should look like the one below.



Exercise: Cut the dendrogram such that you have K = 6 clusters. Colour each cluster and plot the data in the two-dimensional representation obtained from the PCA earlier. Also add the true cancer labels to the figure. The result should look like the figure below.



Again, as for the K-means, clustering, cancers of type MELANOMA and LEUKEMIA were grouped well using the clustering.