Machine Learning

Notes Part 6, HSG-MiQEF, Spring 2022

David Preinerstorfer

Last updated on 2022-03-30 13:51:58

Contents

Goals:

In this lecture, we briefly consider a class of problems which are summarized as **unsupervised learning**. We will look at the problems:

- clustering, and
- dimension reduction.

This part of the notes relies on Chapters 9 and 10 of Richter [2019]. Let me also mention (cf. also the course fact sheet) that clustering and dimension reduction are not the focus of this course, mainly because there is another course (MiQEF) that covers these topics in detail: **Multivariate Statistics**.

1 Unsupervised learning

In contrast to supervised learning, we now do no longer have access to training data $(\boldsymbol{X}_i, Y_i)$ for $i = 1, \ldots, n$ drawn i.i.d., but we only have training data $\boldsymbol{X}_i \in \mathbb{R}^p$ for $i = 1, \ldots, n$. That is, we do not observe classes (or outcomes, more generally) corresponding to the feature vector. The goal here is to discover *patterns* in the training data.

Potential applications of clustering algorithms can be found in:

- market research and customer segmentation (discover distinct groups in a customer database to which one can then, e.g., tailor specific products or price reductions),
- biological data (e.g., differentiating between species) and medical imaging (e.g., functional clustering of brain regions),
- search result clustering,
- recommendation engine,
- pattern recognition,
- or social network analysis.

Important applications can also be found in psychological research, where, e.g., one wishes to find similar personality traits or intelligence types.

2 Clustering

The goal of clustering is to detect similarities in a relatively large database. Once one has derived such similarities, one obtains groups of "similar" data-points, which are also referred to as **clusters**. Dividing the data into such partitions of similar data-points is called **clustering**.

Obviously, there are many notions of "similarity". We shall focus on the case where "similarity" of two feature vectors can sensibly be measured by using the (squared) **Euclidean distance** between them (this rules out qualitative features, for which other methods need to be used).

Note the difference to a classification problem in supervised learning: There, we observe for each data point in the training sample a category. New observations are to be classified, according to a decision rule we obtained from the training data. In a clustering problem, the training sample only contains the features (but no classes)! Hence, we can not compare our conclusions to given outcomes Y_i, which is the reason why the learning problem we consider now is called **unsupervised** instead of **supervised**. As a thought experiment, we could still stipulate the existence of *latent* (i.e., unobserved) classes Y_i, which we would like to determine.

2.1 k-means Clustering

Fix an integer $K \geq 2$, i.e., the number of clusters. The goal is to partition the training data $\boldsymbol{X}_i$ $(i = 1, \ldots, n)$ into K clusters in such a way that

- the features within cluster j are "most" equal, i.e., vary minimally.

2.1.1 Optimization problem

To this end, and to formulate a first optimization problem that achieves the above task, call "assignment function" any map $C : \{1, \ldots, n\} \to \{1, \ldots, K\}$. That is, for every observation $i = 1, \ldots, n$ the map C returns the "cluster" to which that observation belongs (according to the given assignment function). Obviously, there are many such functions (n^K to be precise), and we call the set of those functions $\mathcal{C}$.

To every assignment function $C \in \mathcal{C}$, we shall associate

$$W_n(C) := \frac{1}{2} \sum_{k=1}^{K} \frac{1}{n_k(C)} \sum_{i,i':C(i)=C(i')=k} \|\boldsymbol{X}_i - \boldsymbol{X}_{i'}\|^2,$$

where $n_k(C) := \#\{i \in \{1, \ldots, n\} : C(i) = k\}$ denotes the number of observations assigned to cluster k. Note that $W_n(C)$ measures a re-scaled inequality measure of the features in the K clusters. Because we intend to minimize that inequality, we need to solve the optimization problem

$$\min_{C \in \mathcal{C}} W_n(C);$$

call a solution to that problem $\hat{C}_n$, say, **optimal assignment function**, which is a function of the training sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$.

In order to achieve better applicability and interpretability, it pays off to rewrite $W_n(C)$ as

$$W_n(C) = \sum_{i=1}^{K} \sum_{i:C(i)=k} \|\boldsymbol{X}_i - m_k(C)\|^2, \tag{1}$$

where

$$m_k(C) = \frac{1}{n_k(C)} \sum_{i:C(i)=k} \boldsymbol{X}_i;$$

that is $m_k(C)$ denotes the **average** of the features assigned to cluster k. Before we show that this is indeed correct, note that from the representation we can conclude that

- Every assignment function C defines K clusters with cluster means $m_k(C)$.
- The points within cluster k are those to which $m_k(C)$ is the closest cluster average.
- Any new feature vector $\boldsymbol{X}_{n+1}$, say, can meaningfully be assigned to the cluster the average of which is

closest to $\boldsymbol{X}_{n+1}$ in Euclidean distance.

It remains to prove Equation (1). To this end, we write for every k and every i and i' such that $C(i) = C(i') = k$

$$\|\boldsymbol{X}_i - \boldsymbol{X}_{i'}\|^2 = \|(\boldsymbol{X}_i - m_k(C)) + (m_k(C) - \boldsymbol{X}_{i'})\|^2,$$

which we can expand as

$$\|\boldsymbol{X}_i - m_k(C)\|^2 + \|\boldsymbol{X}_{i'} - m_k(C)\|^2 - 2(\boldsymbol{X}_i - m_k(C))'(m_k(C) - \boldsymbol{X}_{i'}).$$

We recall that we *defined*

$$W_n(C) = \frac{1}{2}\sum_{k=1}^{K} \frac{1}{n_k(C)} \sum_{i,i':C(i)=C(i')=k} \|\boldsymbol{X}_i - \boldsymbol{X}_{i'}\|^2.$$

If we now replace each $\|\boldsymbol{X}_i - \boldsymbol{X}_{i'}\|^2$ by the expression in the penultimate display, we arrive at

$$W_n(C) = \frac{1}{2}\sum_{k=1}^{K} \frac{1}{n_k(C)} \sum_{i,i':C(i)=C(i')=k} \left[\|\boldsymbol{X}_i - m_k(C)\|^2 + \|\boldsymbol{X}_{i'} - m_k(C)\|^2 - 2(\boldsymbol{X}_i - m_k(C))'(m_k(C) - \boldsymbol{X}_{i'})\right].$$

Note that for all k we have

$$\sum_{i,i':C(i)=C(i')=k} \|\boldsymbol{X}_i - m_k(C)\|^2 = \sum_{i,i':C(i)=C(i')=k} \|\boldsymbol{X}_{i'} - m_k(C)\|^2 = n_k(C) \sum_{i:C(i)=k} \|\boldsymbol{X}_i - m_k(C)\|^2.$$

Furthermore, observe that for every k we have

$$0 = \sum_{i,i':C(i)=C(i')=k} (\boldsymbol{X}_i - m_k(C)).$$

We hence obtained that

$$W_n(c) = \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\boldsymbol{X}_i - m_k(C)\|^2.$$

2.1.2 k-means iteration

It is not difficult to see (given that we have done such computations already; cf. the first proof in Part 1 of the lecture notes), that $C \in \mathcal{C}$ minimizes $W_n(c)$ if and only if $(C, m_1(C), \ldots, m_K(C))$ minimizes

$$(C, m_1, \ldots, m_K) \mapsto \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\boldsymbol{X}_i - m_i\|^2.$$

Note that the latter problem can be divided in 2 subproblems:

1. For fixed $m_1, \ldots, m_K$, the optimization problem $C \mapsto \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\boldsymbol{X}_i - m_i\|^2$ is solved by $\tilde{C}(i) \in \arg\min_{k=1,\ldots,K} \|\boldsymbol{X}_i - m_k\|^2$; that is assign every $\boldsymbol{X}_i$ to that cluster k the cluster average of which is

closest to $\boldsymbol{X}_i$.

2. For fixed C, the cluster averages $m_K(C)$ minimize $(m_1, \ldots, m_K) \mapsto \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\boldsymbol{X}_i - m_i\|^2$.

This division into sub-problems gives rise to a two-stage algorithm (which can be shown to converge to a local optimum; hence repeating the algorithm with different starting values is recommended):

1. Fix starting values $m_k \in \mathbb{R}^p$ for $k = 1, \ldots, K$.
2. Repeat the following until convergence, or a maximal number of iterations has been reached

- for $i = 1, \ldots, n$ update $C(i) = \arg\min_{k=1,\ldots,K} \|\boldsymbol{X}_i - m_k\|^2$.
- for $k = 1, \ldots, K$ update m_k as the average of all feature vectors $\boldsymbol{X}_i$ for which $C(i) = k$.

The output of this iteration $\hat{C}$, say, is called locally optimal k-means clustering, and the averages $\hat{m}_1, \ldots, \hat{m}_k$a are called the corresponding cluster centers.

A new feature vector $\boldsymbol{X}_{n+1}$, say, is then assigned to cluster

$$\arg \min_{k=1,\ldots,K} \|\boldsymbol{X}_{n+1} - \hat{m}_k\|^2.$$

Note that in every step of the optimization procedure, the value of the assignment function obtained decreases. Since this sequence is lower-bounded by 0, one can show that the iteration converges to a local minimum (cf. Satz 9.11 in Richter [2019]).

Note also that the algorithm can be easily adapted for "similarity" functions other than the squared Euclidean distance.

2.1.3 Example and implementation

Let's implement the above algorithm (with randomly chosen starting values), and see how the iterations proceed in an example. The implementation is mostly done for illustration purposes. For practical applications, it is recommended to use the `kmeans` function in `R` (in which the above iterative algorithm is referred to as **Lloyd's algorithm**; and also other, more elaborated algorithms, are available).

```
km <- function(X, K, max.it = 20){
  n <- dim(X)[1]
  p <- dim(X)[2]

  m_it <- array(NA, dim = c(max.it+1, K, p))
  cla_it <- matrix(NA, nrow = max.it, ncol = n)
```

```
  #generate the starting centers
  #simply by drawing K random feature vectors

  m <- X[c(sample(1:n, K, replace = FALSE)),]
  m_it[1,,] <- m
  cla <- c()

  fu <- function(x){
    diff <- matrix(x, nrow = K, ncol = p, byrow = T)-m
    min(which.min(apply(diff^2, 1, sum)))
    }

  for(r in 1:max.it){
    cla <- apply(X, 1, fu)
    for(k in 1:K){
      m[k,] <- mean(X[cla == k, ])
    }
    m_it[r+1,,] <- m
    cla_it[r,] <- cla
  }
return(list("centers" = m_it, "assign" = cla_it))
}
```

Let's also write a function that plots the first 4 iterations

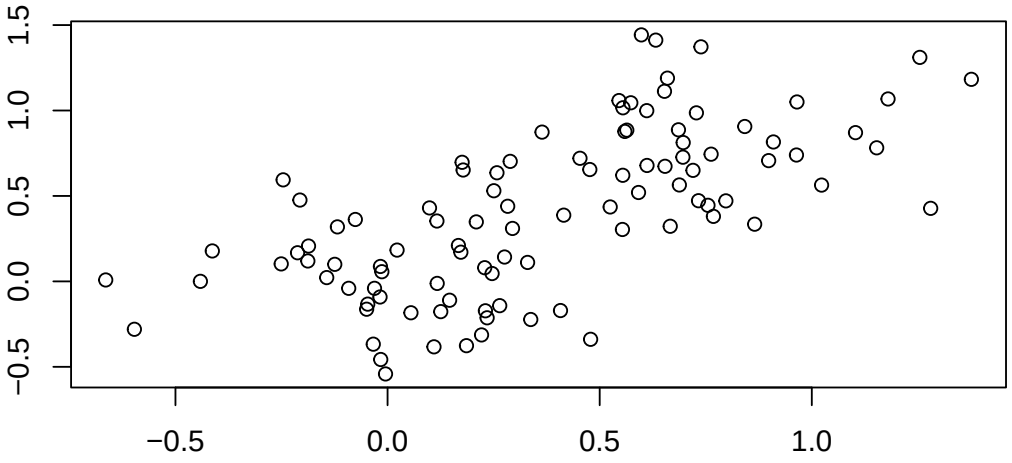
```
plot.km <- function(fit, X){
cla <- fit$assign
cent <- fit$centers

par(mfrow = c(2, 2))
  for(i in 1:4){
    plot(X, xlab = "", ylab = "",
         main = paste("Plot ", i, sep = " "))
```

```
      for(k in 1:K){
       points(X[cla[i,] == k,], col = k+1)
      }
    points(cent[i,,], col = "black", lwd = 2)
  }
}
```

Let's take a look at a dataset for $p = 2$ (i.e., in case there are only 2 features).

```
set.seed(1)
X <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
           matrix(rnorm(100, mean = .75, sd = 0.3), ncol = 2))
plot(X, xlab = "", ylab = "", main = "")
```

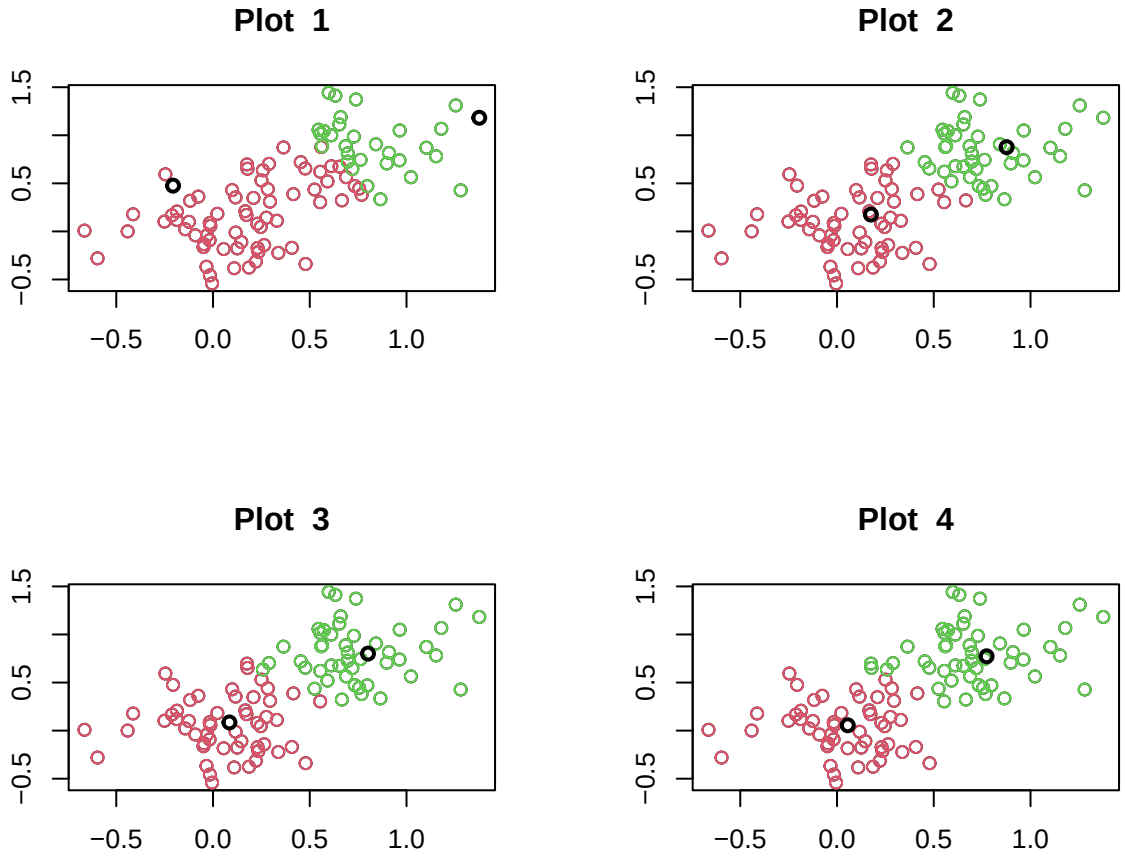


We now apply the k-means algorithm to that dataset with $K = 2$:

```
K <- 2
fit.2 <- km(X, K, 4)
```

We illustrate the four iterations:

```
plot.km(fit.2, X)
```



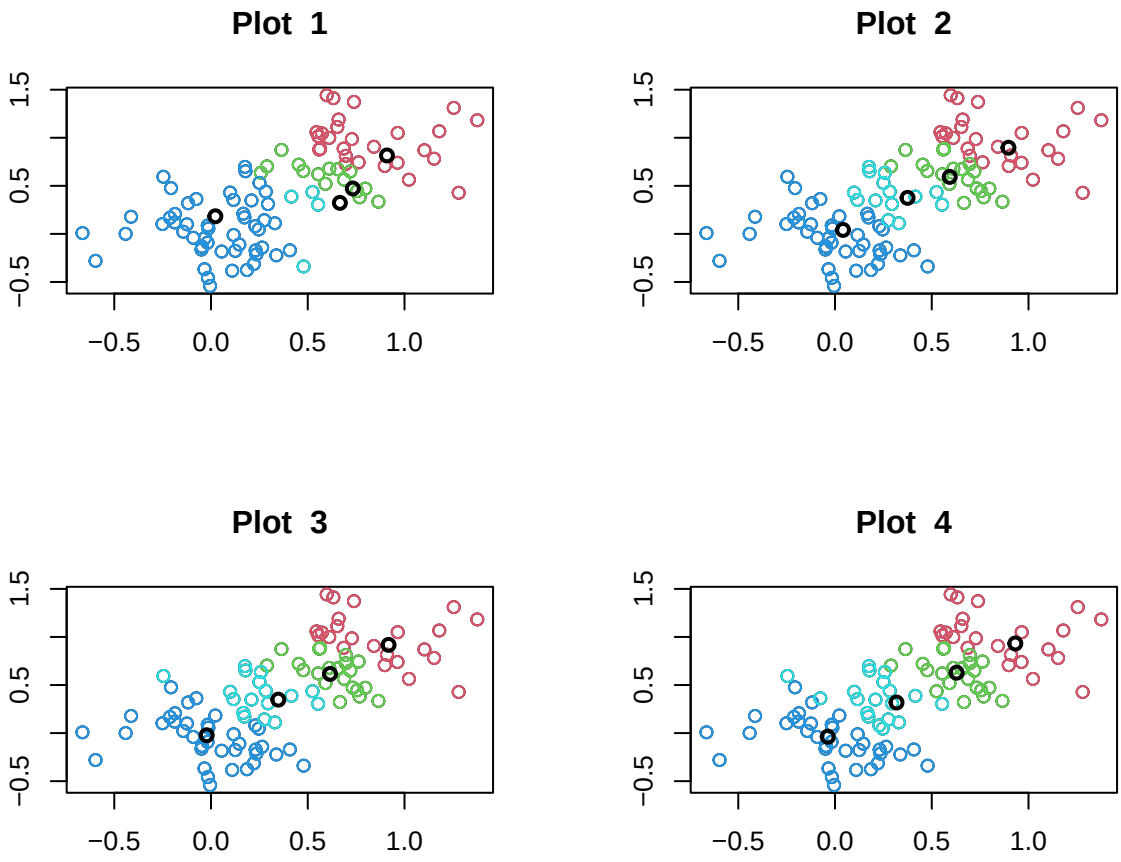
Let's see what happens, if we choose $K = 4$:

```
K <- 4

fit.4 <- km(X, K, 4)
```

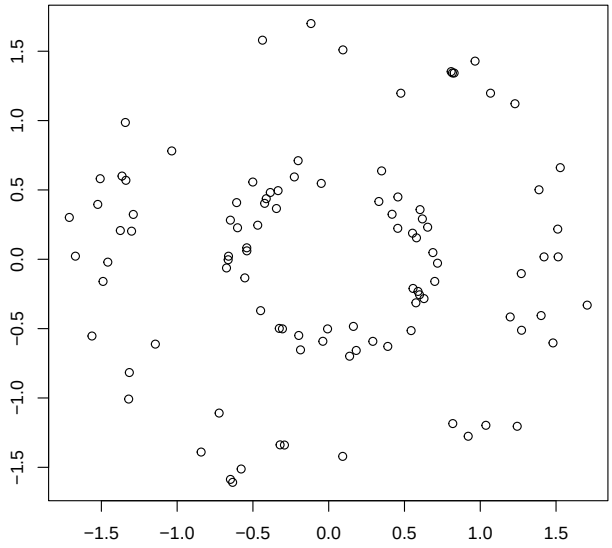
Finally, we illustrate the four iterations:

```
plot.km(fit.4, X)
```



Difficulties obviously arise, in case the data is arranged around two concentric "circles", as can be seen in the following example:

```
v <- runif(100)*2*pi
X2 <- cbind(cos(v), sin(v))
r <- c(runif(50, min = 1/2, max = .75), runif(50, min = 1.25, max = 1.75))
X2 <- diag(r)%*%X2
plot(X2, xlab = "", ylab = "", main = "")
```

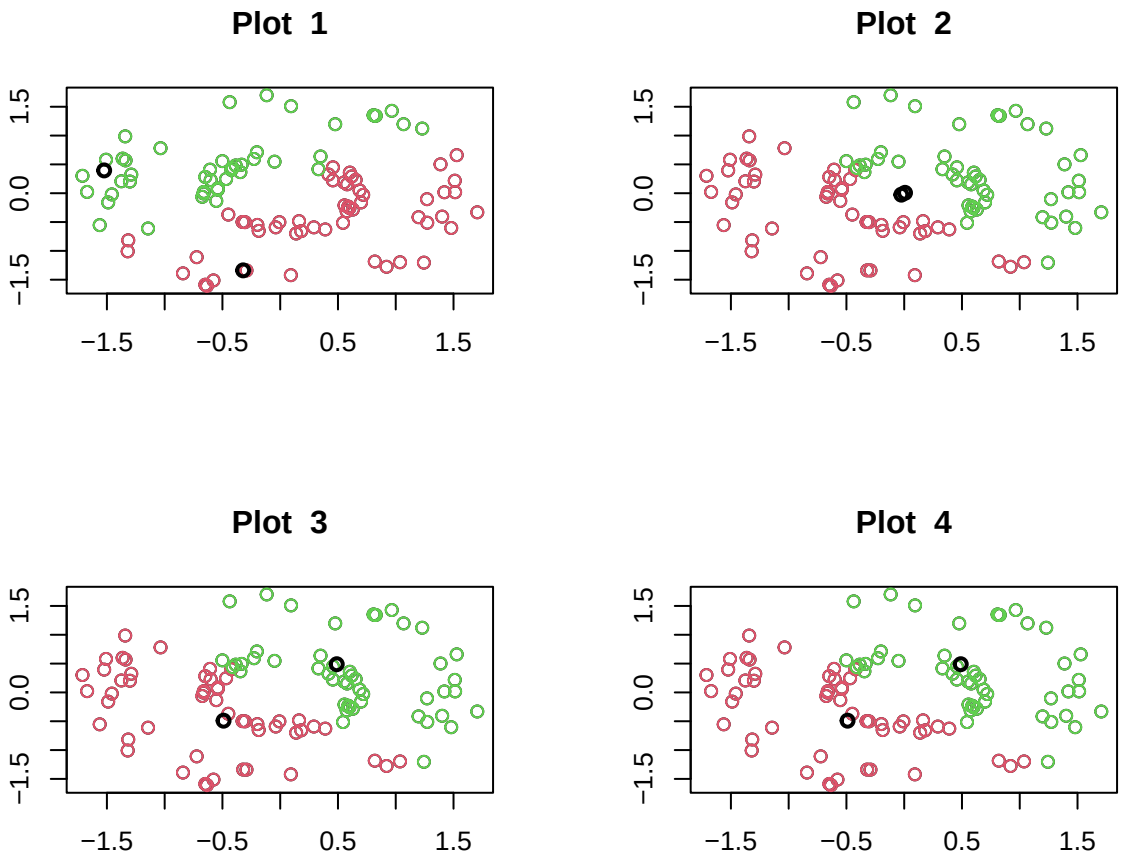



```
K <- 2

fit.22 <- km(X2, K, 4)
```

We illustrate the four iterations:

```
plot.km(fit.22, X2)
```



2.2 Mixture models

The method discussed in the previous section was not based on any distributional assumptions concerning the random vectors $\boldsymbol{X}_i$, but rather was a method that relied on "visual" principles that were then translated into an optimization problem. This resembles the approach we took in supervised learning, for methods such as support vector machines or the kNN classifier, the latter being most similar to k-means clustering in spirit.

In this section, we briefly sketch another approach, which is based on a distributional assumption, and (in this sense) resembles classification methods such as discriminant analysis.

The underlying assumption is that the density f, say, of $\boldsymbol{X}_i$ can be written as

$$f(\boldsymbol{x}) = f_{\boldsymbol{\theta},\boldsymbol{\pi}}(\boldsymbol{x}) = \sum_{i=1}^{K} \pi_k f(\boldsymbol{x}, \boldsymbol{\theta}_k),$$

where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$ and each $\boldsymbol{\theta}_i$ is a "parameter", and $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)'$ is a vector with non-negative

coordinates that sum up to 1. Such models are called **mixture models**. Note that it corresponds to $\boldsymbol{X}_i$ be drawn from one of the densities $f(\boldsymbol{x}, \boldsymbol{\theta}_k)$ each with probability π_k, respectively. That is, we may imagine $\boldsymbol{X}_i$ being drawn from one of K latent classes.

We obtain concrete models by specifying $f(\boldsymbol{x}, \boldsymbol{\theta}_k)$. For example, $\boldsymbol{\theta}_k$ could be the tuple $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ and $f(\boldsymbol{x}, \boldsymbol{\theta}_k)$ could then be a normal distribution with mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$.

These models are parametric, and we can therefore estimate the unknown parameters as well as the mixing weights $\boldsymbol{\pi}$ from data through parametric methods such as maximum likelihood (after potentially reducing the parameter spaces so that the parameters are identified). Therefore, I will not discuss such methods in detail. Let me mention that there is an important algorithm that has been developed for parameter estimation of mixtures: the EM-algorithm. If you are interested in details, please consult Chapter 9.2 in Richter [2019].

3 Dimension reduction

The basic question dimension reduction tries to answer is: if we have to summarize each $\boldsymbol{X}_i$ in $q < p$ coordinates (e.g., $q = 1$), how can we achieve that without losing too much information? Of course, one could just throw away all features $2, 3, \ldots, p$ to reduce the dimension to 1, but then we potentially lose a lot of information.

One typical example where dimension reduction techniques have been applied is in the context of personality questionnaires or intelligence tests. In this situation, one first constructs a questionnaire that intends to measure different aspects of intelligence, say. Then, this questionnaire is given to many participants (representative for a certain population of interest). In a next step, one wants to reduce the dimensionality, i.e., the information contained in *all* answers of a participant to scores on a certain number of (interpretable) sub-scales, to which one can give a certain meaning. Now the question is: how can this be achieved?

Anoter type of example would be a regression problem where there are many regressors, of which we know that there are groups of them that roughly measure the same quantity. It then makes sense to ask whether one can reduce the dimension of the regressors by working with lower-dimensional summary measures.

One method is called **principle component analysis**, PCA for short, which we shall discuss next.

3.1 PCA

In the simplest case, the goal is to summarize the training data $\boldsymbol{X}_i$ for $i = 1, \ldots, n$ as well as possible as points on a straight line

$$g(\alpha) = \boldsymbol{\mu} + \alpha \boldsymbol{v}.$$

That is, we want to summarize the p features in a single number. We impose the condition that $\boldsymbol{v}$ has Euclidean norm equal to 1 (since otherwise α is not well-defined). To reduce dimension we can now choose α, $\boldsymbol{\mu}$ and $\boldsymbol{v}$. Then, to every observation $\boldsymbol{X}_i$, we want to associate a value α_i, corresponding to the point associated to $\boldsymbol{X}_i$ on the line given in the previous display.

The question now is: how to choose $\boldsymbol{\mu}, \boldsymbol{v}$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$?

The answer is that we would choose the line in such a way that we minimize the distance of the points on the line to the points in the training sample. Recall that the goal is to reduce dimension in such a way that as much "information" as possible is preserved.

We would therefore like to solve the optimization problem

$$\min_{\boldsymbol{\alpha}, \boldsymbol{\mu}, \|\boldsymbol{v}\|=1} \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{X}_i - (\boldsymbol{\mu} + \alpha_i \boldsymbol{v})\|^2.$$

It turns out that the solution to this optimization problem is obtained for (cf. Chapter 10.1 of Richter [2019])

- $\hat{\boldsymbol{\mu}}$ the sample average of the $\boldsymbol{X}_i$.
- $\hat{\boldsymbol{v}}$ the eigenvector of unit length corresponding to the **largest** eigenvalue of the sample covariance matrix $\hat{\boldsymbol{\Sigma}} = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{X}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{X}_i - \hat{\boldsymbol{\mu}})'$.
- $\hat{\alpha}_i = \hat{\boldsymbol{v}}'(\boldsymbol{X}_i - \hat{\boldsymbol{\mu}})$, the projection of $\boldsymbol{X}_i$ onto the straight line $\hat{\boldsymbol{\mu}} + \alpha \hat{\boldsymbol{v}}$.

More generally, we can approximate the data by a linear space of some dimension d ($d = 1$ corresponding to a straight line) and solve the optimization problem

$$\min_{\boldsymbol{\alpha}, \boldsymbol{\mu}, \|\boldsymbol{v}_j\|=1} \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{X}_i - (\boldsymbol{\mu} + \sum_{j=1}^{d} \alpha_{ij} \boldsymbol{v}_j)\|^2.$$

Again, one can establish the solution to this optimization problem in closed form (cf. Chapter 10.1 of Richter [2019]):

- $\hat{\boldsymbol{\mu}}$ is again the sample average of the $\boldsymbol{X}_i$.
- $\hat{\boldsymbol{v}}_j$ is the eigenvector of unit length corresponding to j-th **largest** eigenvalue of the sample covariance matrix $\hat{\boldsymbol{\Sigma}} = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{X}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{X}_i - \hat{\boldsymbol{\mu}})'$.
- $\hat{\alpha}_{ij} = \hat{\boldsymbol{v}}_j'(\boldsymbol{X}_i - \hat{\boldsymbol{\mu}})$, the projection of $\boldsymbol{X}_i$ onto the straight line $\hat{\boldsymbol{\mu}} + \alpha \hat{\boldsymbol{v}}_j$.

Essentially, we want to find the d orthogonal directions in p-dimensional space along which the data point-cloud has the largest variability. The other directions are ignored. Note that in case $d = K$, this just amounts to a rotation of the data into a different coordinate system; in this case, we do not lose any information (we just change the representation of the data). The sum of the diagonal entries of $\hat{\mathbf{\Sigma}}$ can be interpreted as the total variance in the data. Note that this sum is just the trace of $\hat{\mathbf{\Sigma}}$, which coincides with the sum of its eigenvalues. In this sense, ,the sum of the d largest eigenvalues divided by the sum of all eigenvalues corresponds to the percentage of variance explained by the d factors. If this percentage is close to 1, d "factors" explain most of the variability in the data (and in this sense: not much information is lost).

Here, we refer to $\hat{\boldsymbol{v}}_j$ as the j-th **principal component**. The j-th largest eigenvalue of $\hat{\mathbf{\Sigma}}$

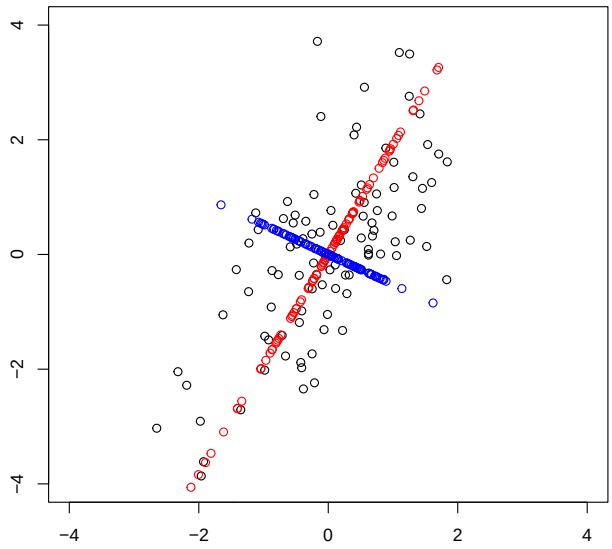
In `R` this can be coded up from scratch quite easily using the function `eigen`, which computes the eigenvalues of a (suitable) matrix. Again, this is done mostly for illustration. For applications, I recommend to use the function `prcomp` (which can also scale the data before applying a PCA).

```
pca <- function(X, K){
  if(K > dim(X)[2]){stop("K should be smaller
                         than the column dimension of X")}
  mu <- apply(X, 2, mean)
  E <- eigen(cov(X))
  V <- E$vectors[,1:K, drop = FALSE]
  A <- (X - matrix(mu, nrow = dim(X)[1], ncol = dim(X)[2], byrow = T))%*%V
  perc <- cumsum(E$values[1:K]/sum(diag(E$values)))
  return(list("mu" = mu, "V" = V, "A" = A, "perc" = perc))
}
```

Let's see what we get for correlated normal data (variances 2 and 1 and covariance 1):

```
Xc <- cbind(rnorm(100), rnorm(100))
Xc <- Xc %*% matrix(c(1, 1, 0, 1), 2, 2, byrow = T)
mp <- matrix(apply(Xc, 2, mean), nrow = 100, ncol = 2, byrow = T)
plot(Xc - mp, xlab = "", ylab = "", main = "", xlim = c(-4, 4), ylim = c(-4, 4))
pcX <- pca(Xc, 2)
PCpts1 <- diag(pcX$A[,1])%*% matrix(t(pcX$V[,1]), nrow = 100, ncol = 2, byrow = T)
PCpts2 <- diag(pcX$A[,2])%*% matrix(t(pcX$V[,2]), nrow = 100, ncol = 2, byrow = T)
points(PCpts1, col = "red")
```

```
points(PCpts2, col = "blue")
```




```
pcX$perc
```

```
## [1] 0.8704527 1.0000000
```

One can now of course generalize the approach by working with nonlinear transformations of the feature vectors (and applying dimension reduction to those). We skip the details. Furthermore, note that in practice it can be advisable to normalize each feature to unit variance before applying the PCA.

References

Stefan Richter. *Statistisches und maschinelles Lernen*. Springer: Berlin, 2019.