Data Mining: Unsupervised Learning

Business Analytics Practice
Winter Term 2015/16
Stefan Feuerriegel

Today's Lecture

Objectives

1. Learning how k-means clustering works
2. Understanding dimensionality reduction via principal component analysis

Outline

Outline

Recap: Supervised vs. Unsupervised Learning

Supervised learning

- Machine learning task of inferring a function from labeled training data
- Training data includes both the input and the desired results
  $\rightarrow$ correct results (target values) are given

Unsupervised learning

- Methods try to find hidden structure in unlabeled data
- The model is not provided with the correct results during the training
- No error or reward signal to evaluate a potential solution
- Examples:
  - Clustering (e. g. by k-means algorithm)
    $\rightarrow$ group into classes only on the basis of their statistical properties
  - Dimensionality reduction (e. g. by principal component analysis)
  - Hidden Markov models with unsupervised learning

Unsupervised Learning

Objective

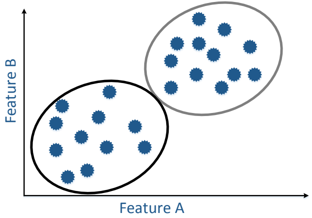
- Find interesting insights in data
- Key metrics can be relationships, main characteristics or similarity of data points
- Usually of exploratory nature as their are no labels

Pros and cons

- Often easy to get unlabeled data
  $\rightarrow$ Labels can be expensive when manual annotations are needed
- Highly subjective as a standardized goal is missing

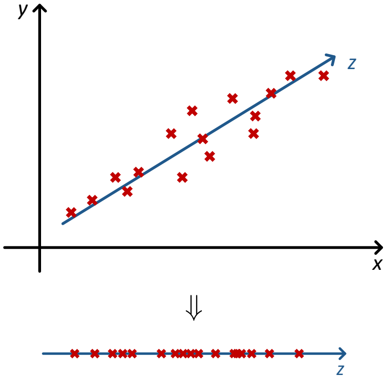
Clustering vs. Dimensionality Reduction

Clustering



- Identifies subgroups of data points with homogeneous characteristics

Dimensionality reduction

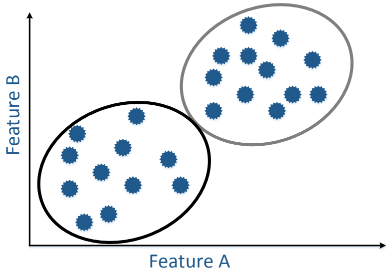


- Calculates the main dimensions across that data points are distributed
- Transforms d...

Outline

k-Means Clustering

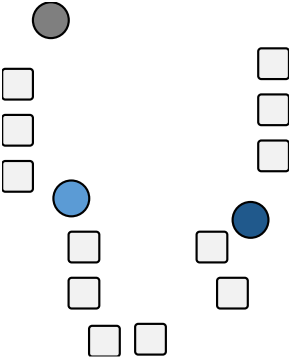
- Partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype for the cluster



- Computationally expensive; instead, we use efficient heuristics
- Default: Euclidean distance as metric and variance as a measure of cluster scatter

Lloyd's Algorithm: Outline

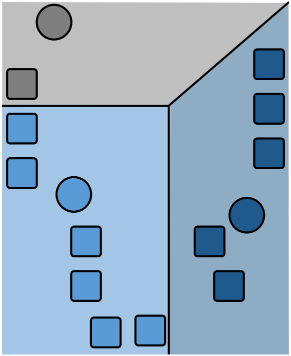
1. Randomly generated k initial "means" (here: $k = 3$)



2. Create k clusters by associating every observation with the nearest mean (colored partitions)
3. Centroid of each of the k clusters becomes the new mean
4. Repeat steps 2 and 3 until convergence

Lloyd's Algorithm: Outline

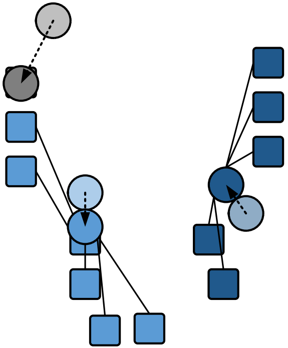
1. Randomly generated k initial "means" (here: $k = 3$)
2. Create k clusters by associating every observation with the nearest mean (colored partitions)



3. Centroid of each of the k clusters becomes the new mean
4. Repeat steps 2 and 3 until convergence

Lloyd's Algorithm: Outline

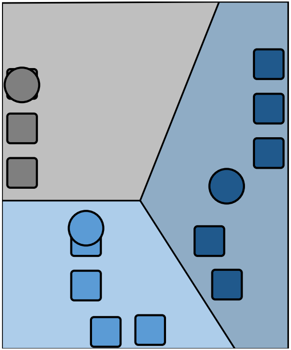
1. Randomly generated k initial "means" (here: $k = 3$)
2. Create k clusters by associating every observation with the nearest mean (colored partitions)
3. Centroid of each of the k clusters becomes the new mean



4. Repeat steps 2 and 3 until convergence

Lloyd's Algorithm: Outline

1. Randomly generated k initial "means" (here: $k = 3$)
2. Create k clusters by associating every observation with the nearest mean (colored partitions)
3. Centroid of each of the k clusters becomes the new mean
4. Repeat steps 2 and 3 until convergence



Lloyd's Algorithm: Pseudocode

1. Initialization
   Choose a set of k means $\mathbf{m}_1^{(1)}, \ldots, \mathbf{m}_k^{(1)}$ randomly
2. Assignment Step
   Assign each observation to the cluster whose mean is closest to it, i.e.

$$S_i^{(t)} = \left\{ \mathbf{x}_p : \left\| \mathbf{x}_p - \mathbf{m}_i^{(t)} \right\| \leq \left\| \mathbf{x}_p - \mathbf{m}_j^{(t)} \right\| \; \forall \; 1 \leq j \leq k \right\}$$

   where each observation is assigned to exactly one cluster, even if it could be is assigned to two or more of them
3. Update Step
   Calculate the new means to be the centroids of the observations in the new clusters

$$\mathbf{m}_i^{(t+1)} = \frac{1}{\left| S_i^{(t)} \right|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

k-Means Clustering in R

- Prepare 2-dimensional sample data

```r
d <- cbind(c(1,2,4,5), c(1,1,3,4))
```

- Call *k*-means via `kmeans(d, k, nstart=n)` with `n` initializations to get cluster means

```r
km <- kmeans(d, 2, nstart=10)
km

## K-means clustering with 2 clusters of sizes 2, 2
##
## Cluster means:
##   [,1] [,2]
## 1  4.5  3.5
## 2  1.5  1.0
##
## Clustering vector:
## [1] 2 2 1 1
##
## Within cluster sum of squares by cluster:
## [1] 1.0 0.5
##  (between_SS / total_SS =  91.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

k-Means Clustering in R

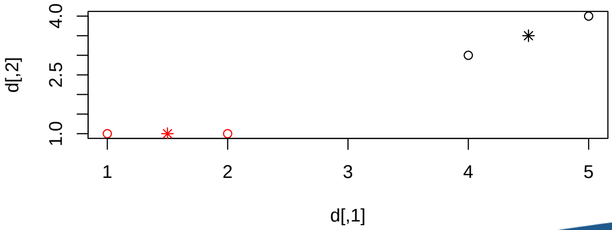
- Calculate within-cluster sum of squares (WCSS) via

```
sum(km$tot.withinss)

## [1] 1.5
```

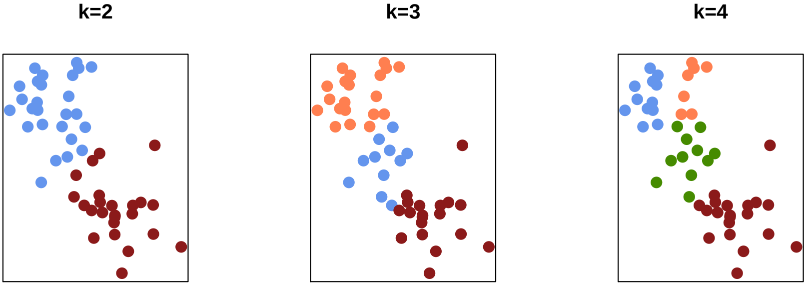
- Plot dataset as circles colored (`col=`) according to calculated cluster
- Add cluster centers `km$centers` as stars (`pch=8`)

```
plot(d, col=km$cluster)
points(km$centers, col=1:nrow(km$centers), pch = 8)
```



Optimal Choice of k

Example: Plots show the results of applying k-means clustering with different values of k



Note: Final results can vary according to random initial means!
$\rightarrow$ In practice, k-means clustering will be performed using multiple random assignments and only the best result is reported

Optimal Choice of k

- Optimal choice of k searches for a balance between maximum compression ($k = 1$) and maximum accuracy ($k = n$)
- Diagnostic checks to determine the number of clusters, such as
  1. Simple rule of thumb sets $k \approx \sqrt{n/2}$
  2. Elbow Method: Plot percent of explained variance vs. number of clusters
  3. Usage of information criteria
  4. ...
- k-means minimizes the within-cluster sum of squares (WCSS)

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{\boldsymbol{x}_j \in S_i} \left\| \boldsymbol{x}_j - \boldsymbol{\mu}_i \right\|^2$$

with clusters $S = \{ S_1, \ldots, S_k \}$ and mean points $\boldsymbol{\mu}_i$ in S_i

Clustering

Research Question

Group countries based on income, literacy, infant mortality and life expectancy (file: `countries.csv`) into three groups accounting for developed, emerging and undeveloped countries.

```r
# Use first column as row names for each observation
countries <- read.csv("countries.csv", header=TRUE, sep=",", row.names=1)
head(countries)
```

```
##            Per.capita.income Literacy Infant.mortality Life.expectancy
## Brazil                 10326     90.0            23.60            75.4
## Germany                39650     99.0             4.08            79.4
## Mozambique               830     38.7            95.90            42.1
## Australia              43163     99.0             4.57            81.2
## China                   5300     90.9            23.00            73.0
## Argentina              13308     97.2            13.40            75.3
```

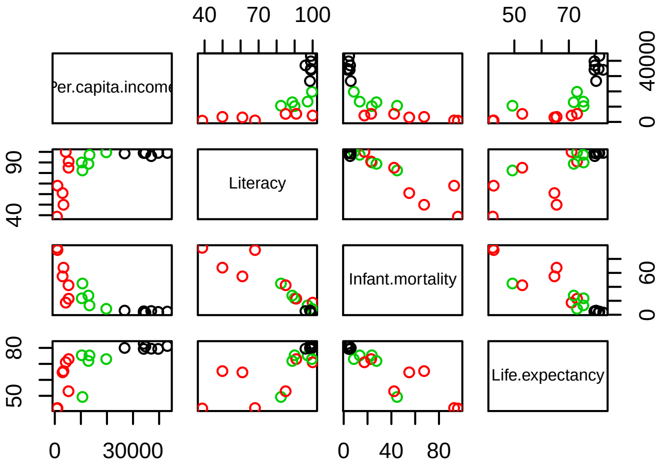
Clustering

```
km <- kmeans(countries, 3, nstart=10)
km
```

```
## K-means clustering with 3 clusters of sizes 7, 7, 5
##
## Cluster means:
##   Per.capita.income Literacy Infant.mortality Life.expectancy
## 1         35642.143    98.50         4.477143        80.42857
## 2          3267.286    70.50        56.251429        58.80000
## 3         13370.400    91.58        23.560000        68.96000
##
## Clustering vector:
##         Brazil        Germany     Mozambique      Australia          China
##              3              1              2              1              2
##      Argentina United Kingdom   South Africa         Zambia        Namibia
##              3              1              3              2              2
##        Georgia       Pakistan          India         Turkey         Sweden
##              2              2              2              3              1
##      Lithuania         Greece          Italy          Japan
##              3              1              1              1
##
## Within cluster sum of squares by cluster:
## [1] 158883600  20109876  57626083
##  (between_SS / total_SS =  94.1 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

Visualizing Results of Clustering

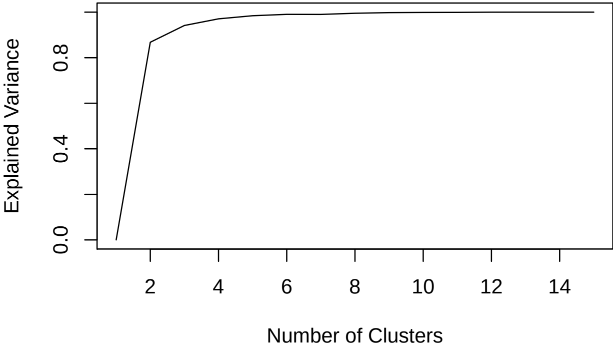
```
plot(countries, col = km$cluster)
```



Elbow Plot to Choose k

Choose k (here: $k = 3$) so that adding another cluster doesn't result in much better modeling of the data

```
ev <- c()
for (i in 1:15) {
  km <- kmeans(countries, i, nstart=10)
  ev[i] <- sum(km$betweenss)/km$totss
}
plot(1:15, ev, type="l", xlab="Number of Clusters", ylab="Explained Variance")
```



Outline

Principal Component Analysis

Motivation

- Large datasets with many variables require extensive computing power
- However, only a small number of variables usually is informative
- High-dimensional data ($\geq$ 4 dimensions) can be difficult to visualize

Principal component analysis (PCA)

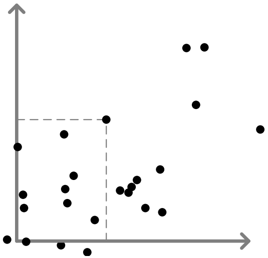
- Finds a low-dimensional representation of data
- Reduces n-dimensional data to k-dimensions with $k \leq n$
- Goal: keep as much of the informative value as possible

Principal Component Analysis

Intuition

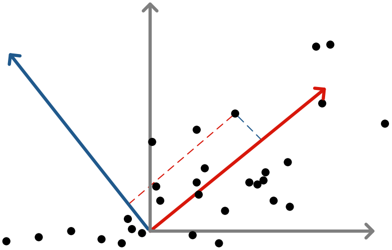
Standard basis

$\boldsymbol{x} = (0.3, 0.5)^T$



Rotated basis

$\boldsymbol{z} = ({\color{red}0.7}, {\color{blue}0.1})^T$

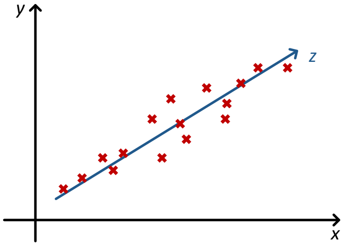


- First principal component is the direction with the largest variance
- Second principal component is orthogonal and in the direction of the largest remaining variance

Principal Component Analysis

Use cases

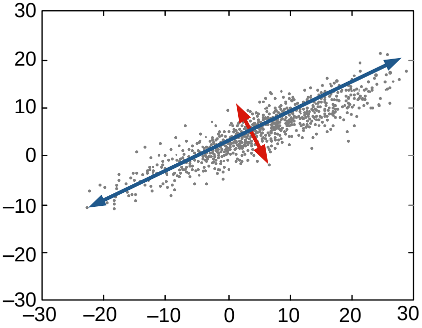
- Principal components can work as input for supervised learning
  $\rightarrow$ especially suited for algorithms with super-linear time complexity in the number of dimensions
- PCA can visualize high-dimensional data with simple graph



$\Rightarrow$

Principal Component Analysis

- Linear combination of uncorrelated variables with maximal variance
  $\rightarrow$ high variance signals high information content
- Data is projected onto orthogonal component vectors so that the projection error is minimized
- Order of directions gives the i-th principal component



Standardizing

- Scaling changes results of PCA $\rightarrow$ standardizing is recommend
- Center variable around mean $\mu = 0$ with standard deviation $\sigma = 1$

Steps

1. Calculate mean and and standard deviation for $\boldsymbol{x} = [x_1, \ldots, x_N]^T$

$$\mu = \frac{1}{N-1} \sum_{i=1}^{N} x_i \qquad \sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2}$$

   Note: R uses internally denominator $N-1$ instead of N

2. Transform variable (built-in via `scale(x)` in R)

$$x_i \leftarrow \frac{x_i - \mu}{\sigma}$$

```r
x <- scale(1:10)
c(mean(x), sd(x))

## [1] 0 1
```

Algorithm

- PCA maps $\boldsymbol{x}_i$ onto a new basis via a linear combination

$$\boldsymbol{z}_i = \phi_{1,i}\, \boldsymbol{x}_1 + \phi_{2,i}\, \boldsymbol{x}_2 + \ldots + \phi_{1,n}\, \boldsymbol{x}_n$$

  with normalization $\sum_{j=1}^{n} \phi_{j,i}^2 = 1$
- $\boldsymbol{z}_i$ is the i-th principal component
- $\phi_{1,i}, \ldots, \phi_{n,i}$ are the loadings of the i-th principal component
- In matrix notation, this gives

$$Z = \Phi X$$

- Geometrically, Φ is a rotation with stretching
  $\rightarrow$ it also spans the directions of the principal components

Algorithm

- If $\boldsymbol{x}_i$ is standardized, it has mean zero and also $\boldsymbol{z}_i$
- Hence, the variance of $\boldsymbol{z}_i$ is

$$\frac{1}{N}\sum_{j=1}^{N} z_{j,i}^2$$

- First loading vector searches a direction to maximize the variance

$$\max_{\phi_{j,1}} \frac{1}{N}\sum_{j=1}^{N} z_{j,i}^2 = \max_{\phi_{j,1}} \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j=1}^{n}\phi_{j,1}x_{i,j}\right]^2 \quad \text{subject to} \quad \sum_{j=1}^{n}\phi_{j,1}^2 = 1$$

- Numerically solved via a singular value decomposition

Singular Value Decomposition

Covariance matrix

- Covariance matrix Σ for the standardized data is given by

$$\Sigma = \frac{1}{N} X^T X \qquad \Leftrightarrow \qquad \Sigma_{ij} = \frac{1}{N} \boldsymbol{x}_i^T \boldsymbol{x}_j$$

- $\Sigma \in \mathbb{R}^{N \times N}$ is symmetric with diagonals being the variance
- Goal: high variance but orthogonality, i. e. zero off-diagonal elements

Singular value decomposition

- Singular value decomposition of square matrix X gives

$$X = V \Sigma V^{-1}$$

- V is a matrix with the eigenvectors of X ($\Rightarrow VV^T = I_N$)
- Σ is a diagonal matrix with the corresponding eigenvalues
- Then $\Phi = V$

PCA in R

- PCA comes with various R packages but also via built-in routines
- Generating sample data

```
set.seed(0)
x <- rnorm(100)
y <- -0.8*x + 0.6*rnorm(100)
d <- cbind(x, y)
```

- Standard deviation of each variable before and after scaling

```
apply(d, 2, sd)

##         x         y
## 0.8826502 0.8546230

d.scaled <- apply(d, 2, scale)
apply(d.scaled, 2, sd)

## x y
## 1 1
```

PCA in R

- Perform PCA with scaling via `prcomp(data, scale=TRUE)`

```
pca <- prcomp(d, scale=TRUE)
```

- Mean and standard deviation used for scaling

```
pca$center  # mean => equals apply(d, 2, mean)

##           x           y
##  0.02266845 -0.04546569

apply(d, 2, mean)

##           x           y
##  0.02266845 -0.04546569

pca$scale   # standard deviation

##         x         y
## 0.8826502 0.8546230
```

PCA in R

- Principal component vectors $\rightarrow$ pick first k columns of interest

```
head(pca$x)

##             PC1         PC2
## [1,] -1.4038205  0.58340965
## [2,]  0.1474487 -0.41157419
## [3,] -2.1955568 -0.10122525
## [4,] -1.7827003  0.21971105
## [5,] -1.1120171 -0.48398399
## [6,]  2.5950741  0.09139112
```

- PCA loadings

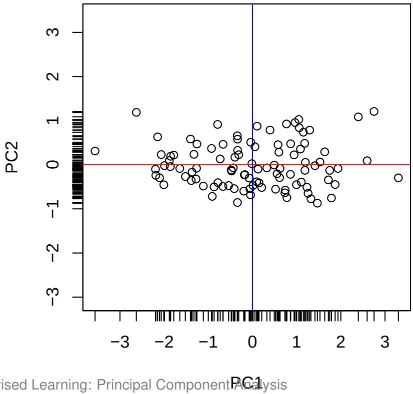
```
pca$rotation

##          PC1       PC2
## x -0.7071068 0.7071068
## y  0.7071068 0.7071068
```

PCA in R

- Visualization of resulting principal component vectors

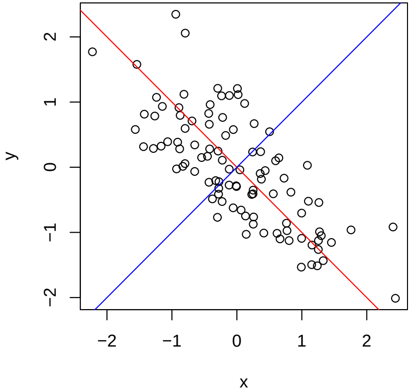
```
plot(pca$x, asp=1) # aspect ratio such that both axes have the same s
box() # reset ticks
axis(1, at=pca$x[, 1], labels=FALSE) # customized ticks
axis(2, at=pca$x[, 2], labels=FALSE)
abline(h=0, col="red")  # 1st principal component
abline(v=0, col="blue") # 2nd principal component
```



PCA in R

- Plot of principal components on original scale in two dimensions

```r
plot(x, y)
rot <- pca$rotation
abline(0, rot[2,1]/rot[1,1], col="red")   # 1st PC
abline(0, rot[1,2]/rot[1,2], col="blue") # 2nd PC
```



PCA in R

- Standard deviation of principal components

```
pca$sdev

## [1] 1.3191770 0.5096784
```

$\rightarrow$ Higher standard deviation in first components, lower in last

- Absolute and proportional variance explained

```
pca$sdev^2  # absolute variance explained by each component

## [1] 1.7402279 0.2597721

pve <- pca$sdev^2 / sum(pca$sdev^2)
pve          # proportion of variance explained

## [1] 0.870114 0.129886
```

- Manual inspection is necessary to identify a suitable k when not explicitly specified beforehand

PCA in R

Case study

- Reduce the dimensionality of the country dataset
- Goal is to retain a large portion of the variance, while still reducing the number of dimensions
- Run PCA for country dataset

```
pca <- prcomp(countries, scale=TRUE)
```

- PCA loadings

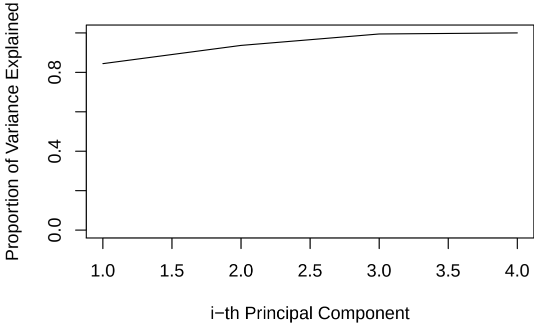
```
pca$rotation

##                          PC1         PC2         PC3         PC4
## Per.capita.income  0.4650236  0.80152688  0.37236742 -0.05148053
## Literacy           0.4943729 -0.54941559  0.50335230 -0.44763206
## Infant.mortality  -0.5346811  0.23163962  0.05703933 -0.81068226
## Life.expectancy    0.5034528  0.04516926 -0.77764097 -0.37385770
```

Proportion of Variance Explained

- Plot with cumulative proportion of variance explained

```
pve <- pca$sdev^2 / sum(pca$sdev^2)
plot(cumsum(pve), xlab="i-th Principal Component",
     ylab="Proportion of Variance Explained",
     type="l", ylim=c(0, 1))
```

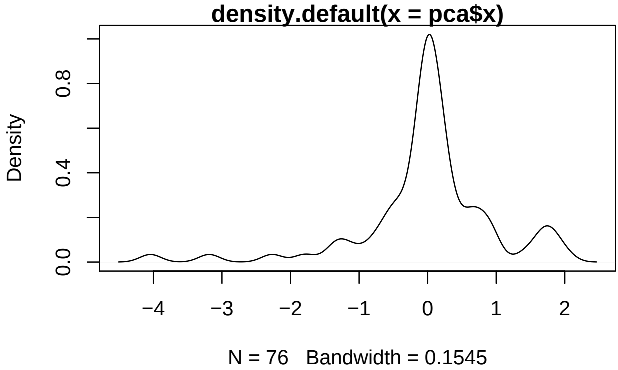


$\rightarrow$ First principal component explains more than 80 % of the variance

PCA Example

- Density estimation reveals subgroups in one dimension

```
plot(density(pca$x))
```



$\rightarrow$ One also observes three groups: a peak, as well as a tail and a leading group

Outline

Summary

- Unsupervised learning usually provides explanatory insights
- *k*-means clustering identifies subsets of similar points
- Elbow plot determines a suitable number of clusters *k*
- PCA reduces dimensions with a minimal amount of information loss

Commands in R

`kmeans(d, k, nstart=n)`	*k*-means clusterin
`prcomp(d, scale=TRUE)`	PCA with scaling
`cumsum(x)`	Cumulative sums
`apply(d, f)`	Apply function `f` to all data points in `d`