Data Science für Start-Ups

Supervised and Unsupervised Learning & Examples

Workshop im Rahmen des DIH SÜD

Data Science and Artificial Intelligence
Institute of Information Management
FH Joanneum – University of Applied Sciences

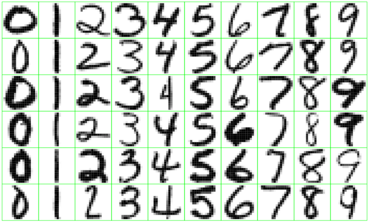
5. Juli 2022

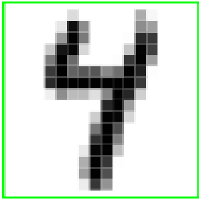
Overview

- Many problems have a special structure – we will see mainly 3 different structures of data (regression/classification and the unsupervised setting).
- In the supervised setting we are primarily interested in a certain quantity y. There are various names for y: response (variable), dependent variable, target (variable), output (variable), outcome, . . .
- Unfortunately, this quantity y is often difficult to measure, e.g. because its measurement is cost-, time- or labour-intensive. In some cases it's even impossible to measure (e.g. tomorrow's stock exchange price (*Börsenkurs*), tomorrow's precipitation in Graz, . . .).
- The idea is to measure one or (typically) more so-called predictors x_j, which are comparably easy/cheap/fast to measure and which can be used to predict y with a so-called prediction model. If we have a single predictor, we will simply call it x, if there are more than 1, we give them the names $x_1, x_2, \ldots$. Alternative names for the x_j are independent variables, inputs, covariates, features, attributes, . . .
- Let's look at some examples . . .

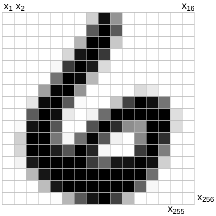
Example – Handwritten Digits

Handwritten digits scanned from U.S. postal envelopes (example from `ESL`). For the human eye it is (in most cases) easy to classify such an image.





The features x_j in our example: we put a 16×16 pixel grid over each handwritten digit and determine the level of *blackness* (ranging from -1 for white to $+1$ for black; *Graustufen*). So each pixel $x_1, x_2, \ldots, x_{256}$ has an associated number in the interval $[-1, +1]$.



Example – Handwritten Digits

In **R** these data are available in the package ElemStatLearn (Book *Elements of Statistical Learning* by Trevor Hastie, Robert Tibshirani and Jerome Friedman) and can be accessed via

```
# package must be installed first
require(ElemStatLearn)
data(zip.train)

# structure of data
class(zip.train)

[1] "matrix" "array"

dim(zip.train)

[1] 7291  257
```

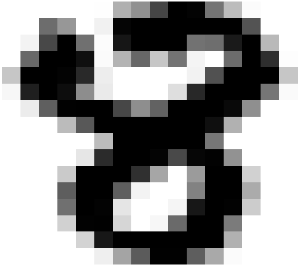
We see that the digit training data are a matrix of dimension 7291×257 (see also the help page for some information). Each of the $n = 7291$ rows represents an object/observation/case/instance (here a case is a single handwritten digit) – first the number $(0$ to $9)$, then the 256 greyscale values.

```
# e.g. let's have a look at the 18th case
zip.train[18, 1:15]

 [1]  8.000 -1.000 -1.000 -1.000 -1.000 -1.000 -0.992 -0.385 -0.143  0.462
[11]  1.000  0.975  0.092 -0.473 -0.968
```

Example – Handwritten Digits

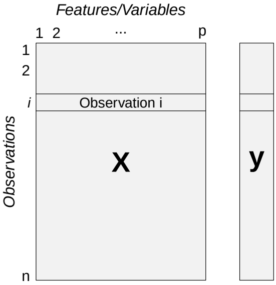
The function `zip2image()` (together with **R**'s `image()` function) can be used for plotting (i.e. the other way round from numerical data in a matrix to images):

```
# plot the previous sign
image(zip2image(zip = zip.train, line = 18), col = gray((256:0)/256),
      xlab = "", ylab = "", xaxt = "n", yaxt = "n", bty = "n")

[1] "digit  8  taken"
```



Supervised Setting – Classification

In supervised learning we have a response y, which we want to model using the predictors $x_1, x_2, \ldots, x_p$. We need a so-called training set with a number of n instances, for which both the x- and y-values are known.



In this example, y represents the class **Zero**, **One**, **Two**, ..., **Nine** (i.e. the response is a categorical and not a numeric variable). So we have a **classification problem**.

Example – Handwritten Digits

We will build e.g. a random forest model for demonstration purposes.

- First we load the **R** packages and the data

```r
# first we load the package(s) and data
require(ElemStatLearn)
data(zip.train)

# package for Random Forest model
require(ranger)
```

- Now we build a classification model with the `ranger()` function

```r
# random forest classification model
rf_model <- ranger(y ~ ., data = zip.train, seed = 123, num.trees = 1000)
```

- Usually we are interested in the performance of our model, i.e. we want to know how accurate the model can predict the digit based on its greyscale image. In a first try we could apply the model to the x-data (i.e. the numeric values of the grayscale pixels) of our training set which gives us a predicted y for each case. We will use the notation $\hat{y}$ for the predicted/estimated value (here a class membership).

```r
# make predictions with the random forest model
preds_training <- predict(rf_model, data = zip.train)
```

Example – Handwritten Digits

- Let's look at the first entries of these predictions:

```
# compare predictions with the truth
head(cbind(observed = zip.train[, "y"],
           predicted = preds_training$predictions), n = 10)
```

```
      observed predicted
 [1,]        7         7
 [2,]        6         6
 [3,]        5         5
 [4,]        8         8
 [5,]        4         4
 [6,]        7         7
 [7,]        4         4
 [8,]        2         2
 [9,]        1         1
[10,]        2         2
```

No classification errors in the first 10 cases ...

Example – Handwritten Digits

A so-called confusion matrix gives a good summary of the model results

```
# confusion matrix
table(true_class = zip.train[, "y"],
      predicted_class = preds_training$predictions)

          predicted_class
true_class    0    1    2    3    4    5    6    7    8    9
         0 1194    0    0    0    0    0    0    0    0    0
         1    0 1005    0    0    0    0    0    0    0    0
         2    0    0  731    0    0    0    0    0    0    0
         3    0    0    0  658    0    0    0    0    0    0
         4    0    0    0    0  652    0    0    0    0    0
         5    0    0    0    0    0  556    0    0    0    0
         6    0    0    0    0    0    0  664    0    0    0
         7    0    0    0    0    0    0    0  645    0    0
         8    0    0    0    0    0    0    0    0  542    0
         9    0    0    0    0    0    0    0    0    0  644
```

Interpretation: There are 658 observations with the true class 3 (sum of all entries in the 3-row) and all of them were correctly classified as a 3.

Are we happy with this result? It seems that we have found a perfect classification model, which always predicts the correct class? What could be the problem, if we evaluate our (classification) model this way?

Example – Handwritten Digits

Building and evaluating (i.e. assessing its performance) a model on the same data set (i.e. with the same observations) is problematic. We get a more realistic estimate of the prediction error, if we apply our model on a new and independent data set (also known as a test set), so far unseen by the model.

In many cases such an independent test set is not available ($\Longrightarrow$ data splitting) – here we are lucky and have such a test set in the **R** object `zip.test`:

```
# load data; very large test set
data(zip.test)
dim(zip.test)

[1] 2007  257

# prepare in the same way as training set
colnames(zip.test) <- c("y", paste("x_", 1:256, sep = ""))
zip.test <- as.data.frame(zip.test)
zip.test[, "y"] <- as.factor(zip.test[, "y"])

# apply model to test set
preds_test <- predict(rf_model, data = zip.test)
```

Example – Handwritten Digits

Also for the test set the true outcomes are known for all cases. So we can look at the confusion matrix

```
# confusion matrix for test set
(cm_test <- table(true_class = zip.test[, "y"],
                  predicted_class = preds_test$predictions))


          predicted_class
true_class   0   1   2   3   4   5   6   7   8   9
         0 353   0   2   0   2   0   1   0   0   1
         1   0 255   0   0   4   0   4   1   0   0
         2   2   0 181   5   2   1   1   1   5   0
         3   0   0   4 149   0  10   0   0   3   0
         4   0   2   4   0 188   0   2   0   0   4
         5   3   0   0   5   1 147   0   0   1   3
         6   0   0   3   0   2   3 160   0   2   0
         7   0   0   1   0   6   0   0 137   2   1
         8   3   0   4   3   0   3   0   0 149   4
         9   0   2   0   0   4   0   0   0   3 168
```

Interpretation: There are 166 images representing the number/class 3 $(4 + 149 + 10 + 3)$ and most (149), but not all were correctly classified. On the other hand, there are 13 $(5 + 5 + 3)$ images representing a number/class other than 3, but which were wrongly classified as 3.

Example – Handwritten Digits

- Assessing the overall quality with the misclassification rate in the test set.

$$\frac{\text{number of misclassified objects}}{\text{total number of objects}}$$

```
# misclassification rate in percent
100 * (1 - sum(diag(cm_test)) / sum(cm_test))

[1] 5.979073
```

gives us about $6\,\%$ misclassification rate. What does this number tell us?

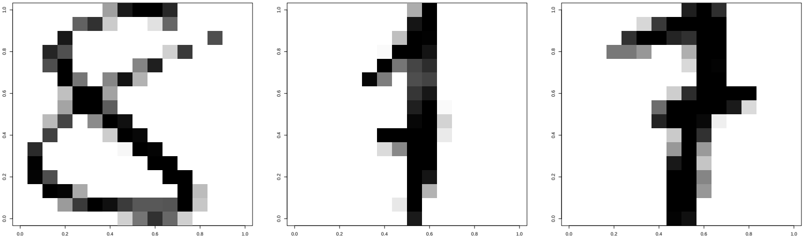
- The contrary measure is the accuracy (proportion of correctly classified elements):

$$\frac{\text{number of correct classification}}{\text{total number of objects}}$$

Example – Handwritten Digits

Some examples of handwritten digits, which were incorrectly classified (in total 120 misclassifications):

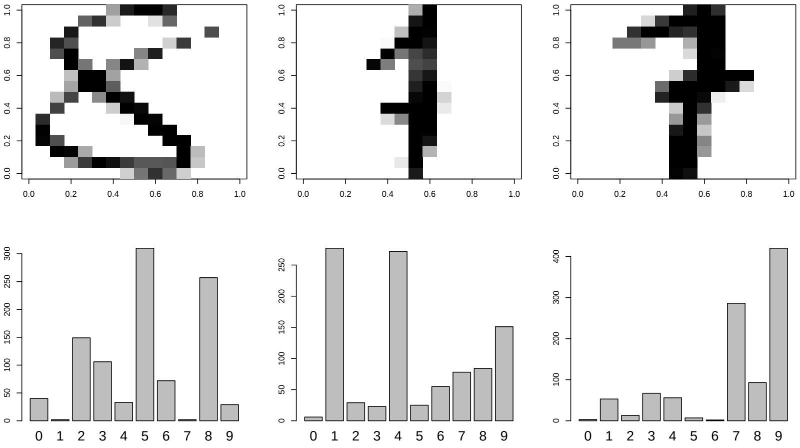
```
[1] "digit  8  taken"
[1] "digit  4  taken"
[1] "digit  7  taken"
```



The above digits shall represent the numbers 8, 4 and 7, but were classified as 5, 1 and 9.

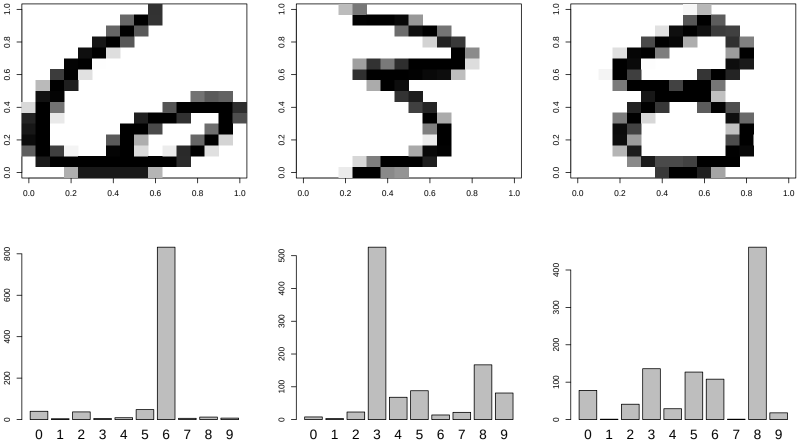
Example – Handwritten Digits

We can even have a closer look – a random forest model consists of a large number (here: 1000, the argument `num.trees` in the `ranger()` call) of trees. Each tree is a classification model on its own. What does each tree predict?



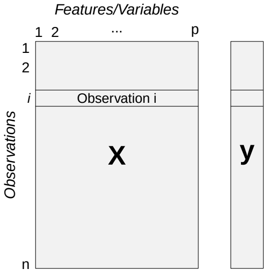
Example – Handwritten Digits

Some of the correctly classified examples ...



Supervised Setting – Classification vs. Regression

- In the previous example we had the following setup



with some predictors x_j (can be numeric or categorical, arranged in columns of $\mathbf{X}$) and a categorical response (classification problem).

- If we have a numeric response variable y, we have a regression problem.

Example – We rent a flat ...

- We want to rent a flat and have an offer for 800 Euro. Is this a fair price?
- Of course the available information is clearly not enough to make a statement – where is the flat (which town, which district), how big is the flat, does it have furniture inside, ...).
- So, again with some more details: the flat has $80\,\text{m}^2$ – is this a fair price?
- Our strategy could be: we take a set of flats, of which we know their size and their price. We make the assumption that *the larger the flat, the more it will cost* (on average).
- **Note**: now we have a numeric quantity y, which we want to model (still it is a supervised learning problem).

Example – We rent a flat . . .

Fortunately, we have an appropriate data set (München, 2015) with $n \approx 3000$.

```
# Einlesen der korrigierten Daten
mieten <- read.table("../../Angewandte Statistik/Daten/Mieten/bearbeitete_Daten/Mietspiegel_Muenchen.csv",
                     header = TRUE, sep = " ")

# erste paar Zeilen
head(mieten, n = 10)

        nm  nmqm wfl rooms     bj                        bez wohngut wohnbest  ww0  zh0 badkach0 badextra kueche
1   608.40 12.67  48     2 1957.5               Untergiesing    nein     nein nein nein       ja     nein   nein
2   780.00 13.00  60     2 1983.0                Bogenhausen      ja     nein nein nein       ja     nein     ja
3   822.60  7.48 110     5 1957.5                Obergiesing    nein     nein nein   ja       ja       ja   nein
4   500.00  8.62  58     2 1957.5          Schwanthalerhoehe    nein     nein nein nein       ja     nein     ja
5   595.00  8.50  70     3 1972.0 Aubing-Lochhausen-Langwied    nein     nein nein nein     nein     nein   nein
6   960.00 11.85  81     3 2006.5          Schwanthalerhoehe    nein     nein nein nein       ja     nein   nein
7  1120.00 11.55  97     3 2000.5                     Hadern      ja     nein nein nein       ja       ja     ja
8   685.00 13.70  50     2 1972.0                Maxvorstadt      ja     nein nein nein     nein     nein     ja
9   767.50 10.81  71     3 1983.0               Untergiesing    nein     nein nein nein       ja     nein   nein
10  565.68  7.44  76     3 1957.5               Untergiesing    nein     nein nein   ja       ja       ja   nein
```

Focus on the different types of variables we have (*Datentypen*). Our target variable is `nm` (Nettomiete).

Example – We rent a flat . . .

- In a very simple analysis, we could take all the flats with exactly $80\,\mathrm{m}^2$ and look at the corresponding rents.

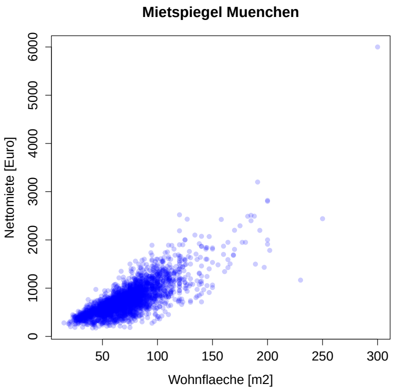
```
# alle Wohnungen mit 80 m^2
summary(mieten[mieten$wfl == 80, "nm"])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  330.0   693.7   792.5   804.3   912.0  1560.0
```

- We see that on average, a flat with $80\,\mathrm{m}^2$ costs 804 Euro, so our 800 Euro seem to be a reasonable price.
- What could we do, if there is no flat with exactly $80\,\mathrm{m}^2$ in our data set? What could we do, if we also want to consider other varialbes/factors, which clearly determine the price?

Example – We rent a flat ...

How does the *Nettomiete* depend on the size of the flat (Variable `wfl` - *Wohnfläche*)?

```
plot(x = mieten$wfl, y = mieten$nm, pch = 19, col = rgb(0,0,1,0.2),
     xlab = "Wohnflaeche [m2]", ylab = "Nettomiete [Euro]", main = "Mietspiegel Muenchen")
```



How would we classify the relationship? What about the outliers (flat with $300\,\mathrm{m}^2$)?

Example – We rent a flat . . .

- We could build a linear regression model of the form

$$\mathsf{nm} = \beta_0 + \beta_1 \cdot \mathsf{wfl} + \beta_2 \cdot \mathsf{bj} + \ldots + \beta_p \cdot \mathsf{bez}$$

- **Note**: It is possible to include qualitative as well as quantitative predictors (variables) in such a model

```r
# wir entfernen den/die Ausreisser
mieten2 <- mieten[mieten$wfl <= 210, ]

# Modell mit 5 Variablen
lin_mod <- lm(nm ~ wfl + rooms + wohngut + badextra + zh0, data = mieten2)
```

Example – We rent a flat . . .

- We might want to ask: how good is our model – we can obtain numerical quantities or use plots.

```
summary(lin_mod)


Call:
lm(formula = nm ~ wfl + rooms + wohngut + badextra + zh0, data = mieten2)

Residuals:
    Min      1Q  Median      3Q     Max
-812.64 -106.72    3.09  103.79 1240.61

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   72.3923    21.4727   3.371 0.000757 ***
wfl           11.7537     0.2601  45.188  < 2e-16 ***
rooms        -63.0117     6.6273  -9.508  < 2e-16 ***
wohngutnein  -75.8218     7.3161 -10.364  < 2e-16 ***
badextranein -60.8802    11.0944  -5.487 4.41e-08 ***
zh0nein      125.4160    13.9939   8.962  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 192.3 on 3056 degrees of freedom
Multiple R-squared:  0.6466,Adjusted R-squared:  0.6461
F-statistic:  1118 on 5 and 3056 DF,  p-value: < 2.2e-16
```

Example – We rent a flat . . .

- Our model will make predictions $\hat{y}$ and we know the actual values y (observed).
- From a good model we will expect that predicted and observed will be close. A measure for the average prediction error is the root mean squared error:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- In our case it is

```
sqrt(mean((mieten2$nm - predict(lin_mod))^2))
```

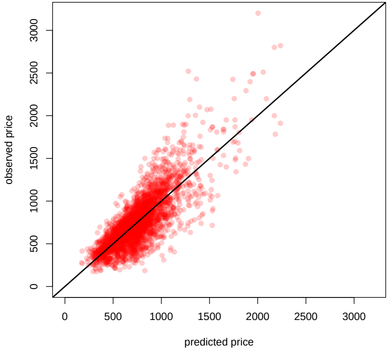
```
[1] 192.1558
```

So we can predict the *Nettomiete* with an average accuracy of ≈ 190 Euro.

Example – We rent a flat . . .

We can also plot the observed values versus the predicted values.

```
plot(x = predict(lin_mod), y = mieten2$nm, pch = 19, col = rgb(1,0,0,0.2),
     xlim = c(0, 3200), ylim = c(0, 3200), xlab = "predicted price", ylab = "observed price")
abline(a = 0, b = 1, lwd = 2)
```



How can we use such a model and which data do we need?

PCA with **R** – wines data

In many situations we just have some data $\mathbf{X}$, but no corresponding $\mathbf{y}$ we want to predict. Let's look at an example – the wines data set (contained in the **R** package kohonen):

```
# load the package/data
require(kohonen)
data(wines)

# dimension of data
dim(wines)

[1] 177  13

# what variables do we have
colnames(wines)

 [1] "alcohol"          "malic acid"        "ash"               "ash alkalinity"    "magnesium"
 [6] "tot. phenols"     "flavonoids"        "non-flav. phenols" "proanth"           "col. int."
[11] "col. hue"         "OD ratio"          "proline"
```

PCA with **R** – wines data

```r
# die ersten Zeilen
head(wines[, 1:10])


     alcohol malic acid  ash ash alkalinity magnesium tot. phenols flavonoids non-flav. phenols proanth col. int.
[1,]   13.20       1.78 2.14            11.2       100         2.65       2.76              0.26    1.28      4.38
[2,]   13.16       2.36 2.67            18.6       101         2.80       3.24              0.30    2.81      5.68
[3,]   14.37       1.95 2.50            16.8       113         3.85       3.49              0.24    2.18      7.80
[4,]   13.24       2.59 2.87            21.0       118         2.80       2.69              0.39    1.82      4.32
[5,]   14.20       1.76 2.45            15.2       112         3.27       3.39              0.34    1.97      6.75
[6,]   14.39       1.87 2.45            14.6        96         2.50       2.52              0.30    1.98      5.25
```

Unsupervised Learning

Questions and problems we might have regarding such data:

- Are there any groups/clusters among the data. We might define a group as chemically similar objects (which poses the next question: what does *chemically similar* mean?)
- Are there any outlying observations (*outliers*) not fitting to any of the (eventually) discovered groups?
- How can we visualize such data? What might be a problem with univariate or bivariate plots (such as histograms/boxplots or 2D scatterplots)?
- Assuming that we have found some groups in the data and we have a completely new observation – to which group does this observation belong to?
- ...

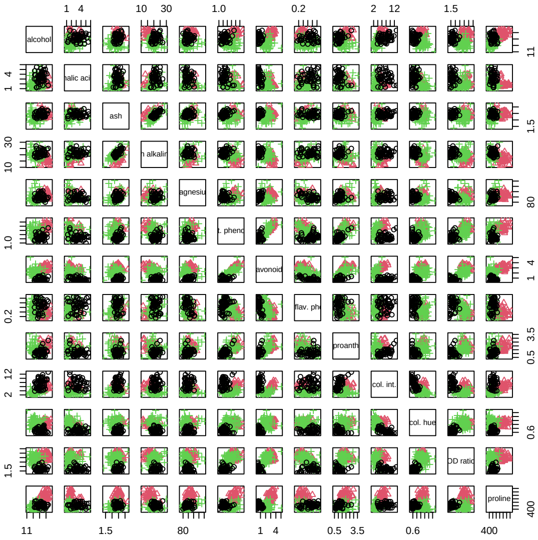
Unsupervised Learning – `wines` data

- For $n = 177$ wines (objects) the data frame contains the results of chemical analyses. Wines were grown in the same region in Italy (Piedmont), but originate from 3 different cultivars (German: *Sorte*) – `Barolo`, `Grignolino` and `Barbera`. They are given in the object `vintages`:

```
# cultivars of wine
head(vintages)

[1] Barolo Barolo Barolo Barolo Barolo Barolo
Levels: Barbera Barolo Grignolino

# how many observations from each cultivar
table(vintages)

vintages
   Barbera     Barolo Grignolino
        48         58         71
```

- We will not use this qualitative (factor) variable for calculating the `PCA` (just for e.g. coloring the data points).

Unsupervised Learning – `wines` data



Unsupervised Learning – `wines` data

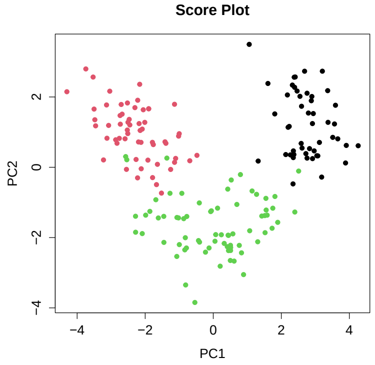
- We perform a `PCA` (Principal Component Analysis) with the `wines` data set (`x = wines`).

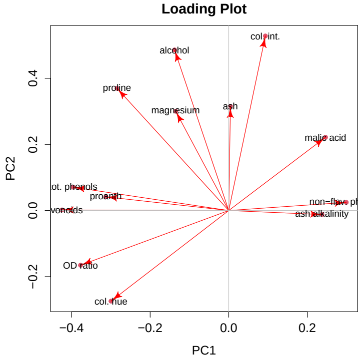
```
# PCA with wines data
pca_wines <- prcomp(x = wines, center = TRUE, scale. = TRUE, retx = TRUE)
```

- What happens is that the high-dimensional data are projected onto a lower-dimensional space (which is more accessible, i.e. it can be plotted).

PCA with R

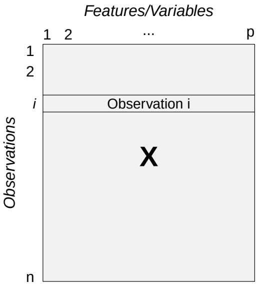
Score- and loading plots can be obtained with the corresponding matrices in the items `x` (scores) and `rotation` (loadings), e.g. a plot of the scores/loadings of `PC2` versus `PC1` (code on the next slide):





Unsupervised Learning

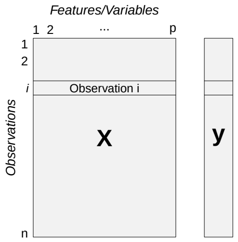
The general setup in unsupervised situations: $\mathbf{X}$ of dimension $n \times p$. p variables $(x_1, x_2, \ldots, x_p)$ measured on n objects.



Regression, Classification, Clustering

Supervised

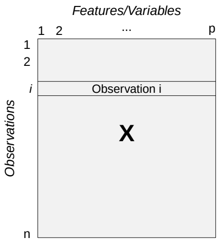
- predict a response y with predictors x_j
- classification: y is qualitative (categorical)
- regression: y is quantitative (numeric)



MLR, PCR, PLS, Lasso, Ridge Regression, Elastic Net, Trees, Random Forests, . . . (regression) and LDA, QDA, kNN, SVM, . . .

Unsupervised

- discover interesting structure in data
- no y to predict
- often part of EDA (exploratory data analysis)



PCA, MDS, Factor Analysis, Kohonen maps, Hierarchical clustering, model based clustering, kmeans, . . .

DIHSÜD

FFG
Forschung wirkt.

Das Land Steiermark

LAND KÄRNTEN

FH JOANNEUM
University of Applied Sciences