THE BIG-R BOOK

FROM DATA SCIENCE TO LEARNING MACHINES AND BIG DATA

— PART 05—

Dr. Philippe J.S. De Brouwer

last compiled: September 1, 2021
Version 0.1.1

THE BIG R-BOOK:
From Data Science to Big Data and Learning Machines

♡— PART 05: Modelling —♡

These slides are to be used in with the book – for best experience, teachers will read the book *before* using the slides and students have access to the book and the code.

part 05: Modelling

↓

chapter 21:

Regression Models

PART 05: MODELLING

↓

CHAPTER 21: REGRESSION MODELS

↓

SECTION 1:

Linear Regression

With a linear regression we try to estimate an unknown variable, y, (also “dependent variable”) based on a known variable, x, (also “independent variable”) and some constants (a and b). Its form is

$$y = ax + b$$

```r
library(MASS)

# Explore the data by plotting it:
plot(survey$Height, survey$Wr.Hnd)
```

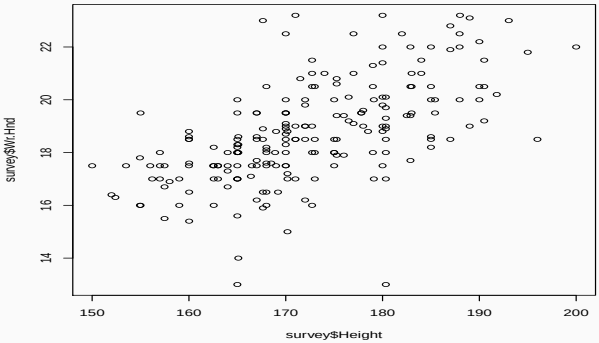


Figure 1: A scatter-plot generated by the line “plot(survey$Height, survey$Wr.Hnd).”

```r
# Create the model:
lm1 <- lm (formula = Wr.Hnd ~ Height, data = survey)
summary(lm1)
##
## Call:
## lm(formula = Wr.Hnd ~ Height, data = survey)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.6698 -0.7914 -0.0051  0.9147  4.8020
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.23013    1.85412  -0.663    0.508
## Height       0.11589    0.01074  10.792   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.525 on 206 degrees of freedom
##   (29 observations deleted due to missingness)
## Multiple R-squared:  0.3612,Adjusted R-squared:  0.3581
## F-statistic: 116.5 on 1 and 206 DF,  p-value: < 2.2e-16
```

```r
# Create predictions:
h <- data.frame(Height = 150:200)
Wr.lm <- predict(lm1, h)

# Show the results:
plot(survey$Height, survey$Wr.Hnd,col="red")
lines(t(h),Wr.lm,col="blue",lwd = 3)
```

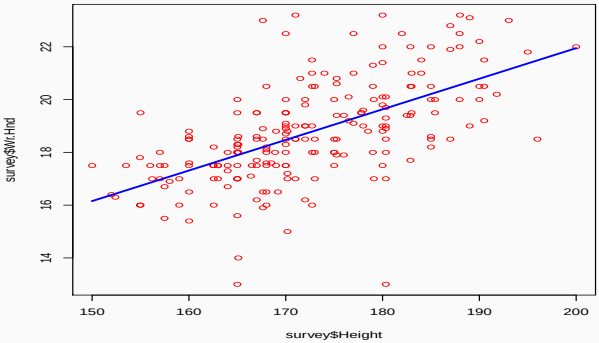


Figure 2: A plot visualizing the linear regression model (the data in red and the regression in blue).

```r
# Or use the function abline()
plot(survey$Height, survey$Wr.Hnd,col = "red",
     main = "Hand span in function of Height",
     abline(lm(survey$Wr.Hnd ~ survey$Height ),
            col='blue',lwd = 3),
     cex = 1.3,pch = 16,
     xlab = "Height",ylab ="Hand span")
```

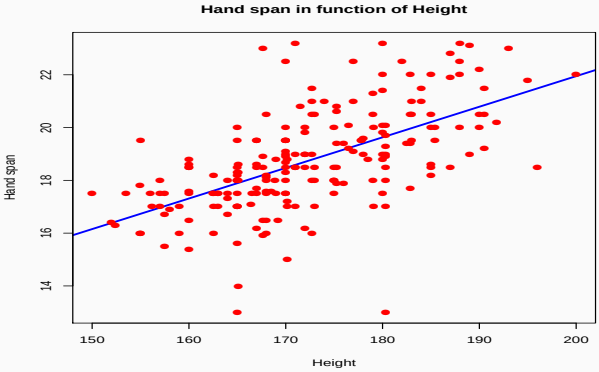


Figure 3: Using the function abline() and cleaning up the titles.

Question #1 – Build a linear model

Consider the data set `mtcars` from the library MASS. Make a linear regression of the fuel consumption in function of the parameter that according to you has the most explanatory power. Study the residuals. What is your conclusion?

PART 05: MODELLING

↓

CHAPTER 21: REGRESSION MODELS

↓

SECTION 2:

Multiple Linear Regression

Multiple regression is a relationship between more than two known variables (independent variables) to predict one variable (dependent variable). The generic form of the model is:

$$y = b + a_1x_1 + a_2x_2 + \cdots + a_nx_n$$

In R, the `lm()` function will handle this too. All we need to do is update the parameter `formula`:

```
# We use mtcars from the library MASS
model <- lm(mpg ~ disp + hp + wt, data = mtcars)
print(model)
##
## Call:
## lm(formula = mpg ~ disp + hp + wt, data = mtcars)
##
## Coefficients:
## (Intercept)         disp           hp           wt
##   37.105505    -0.000937    -0.031157    -3.800891
```

Note also that all coefficients and intercept can be accessed via the function `coef()`:

```
# Accessing the coefficients
intercept <- coef(model)[1]
a_disp    <- coef(model)[2]
a_hp      <- coef(model)[3]
a_wt      <- coef(model)[4]

paste('MPG =', intercept, '+', a_disp, 'x disp +',
     a_hp,'x hp +', a_wt, 'x wt')
## [1] "MPG = 37.1055052690318 + -0.000937009081489667 x disp + -0.0311565508299456 x hp + -3.80089058263761 x wt"
```

```r
# This allows us to manually predict the fuel consumption
# e.g. for the Mazda Rx4
2.23 + a_disp * 160 + a_hp * 110 + a_wt * 2.62
##      disp
## -11.30548
```

Question #2 – Build a multiple linear regression

Consider the data set `mtcars` from the library `MASS`. Make a linear regression that predicts the fuel consumption of a car. Make sure to include only significant variables and remember that the significance of a variable depends on the other variables in the model.

Definition 1 (Poisson Regression)

The general form of the Poisson Regression is

$$\log(y) = b + a_1 x_1 + a_2 x_2 + b_n x_n$$

with:

- y: the predicted variable (aka response variable, independent variable, or unknown variable)
- a and b are the numeric coefficients.
- x is the known variabl, aka the predictor variable, or independent variable.

The Poisson Regression can be handled by the function `glm()` in R, its general form is as follows.

Function use for glm()

```
glm(formula, data, family)
```

where:

- `formula` is the symbolic representation the relationship between the variables,
- `data` is the dataset giving the values of these variables,
- `family` is R object to specify the details of the model and for the Poisson Regression is value is “Poisson”.

Example i

Consider a simple example, where we want to check if we can estimate the number of cylinders of a car based on its horse power and weight, using the dataset `mtcars`

Example ii

```
m <- glm(cyl ~ hp + wt, data = mtcars, family = "poisson")
summary(m)
##
## Call:
## glm(formula = cyl ~ hp + wt, family = "poisson", data = mtcars)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.59240  -0.31647  -0.00394   0.29820   0.68731
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1.064836   0.257317   4.138  3.5e-05 ***
## hp          0.002220   0.001264   1.756    0.079 .
## wt          0.124722   0.090127   1.384    0.166
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 16.5743  on 31  degrees of freedom
## Residual deviance:  4.1923  on 29  degrees of freedom
## AIC: 126.85
##
## Number of Fisher Scoring iterations: 4
```

Example iii

Weight does not seem to be relevant, so we drop it and try again (only using horse power):

```
m <- glm(cyl ~ hp, data = mtcars, family = "poisson")
summary(m)
```

```
##
## Call:
## glm(formula = cyl ~ hp, family = "poisson", data = mtcars)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.97955  -0.30748  -0.03387   0.28155   0.73433
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 1.3225669  0.1739422   7.603 2.88e-14 ***
## hp          0.0032367  0.0009761   3.316 0.000913 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 16.5743  on 31  degrees of freedom
## Residual deviance:  6.0878  on 30  degrees of freedom
## AIC: 126.75
##
## Number of Fisher Scoring iterations: 4
```

Example iv

Syntax of Non-Linear Regression

Function use for nls()

`nls(formula, data, start)` with

1. `formula` a non-linear model formula including variables and parameters,
2. `data` the data-frame used to optimize the model,
3. `start` a named list or named numeric vector of starting estimates.

Example for nls() i

```
# Consider observations for dt = d0 + v0 t + 1/2 a t^2
t  <- c(1,2,3,4,5,1.5,2.5,3.5,4.5,1)
dt <- c(8.1,24.9,52,89.2,136.1,15.0,37.0,60.0,111.0,8)

# Plot these values.
plot(t,  dt, xlab = "time", ylab = "distance")

# Take the assumed values and fit into the model.
model <- nls(dt ~ d0 + v0 * t + 1/2 * a * t^2,
             start = list(d0 = 1,v0 = 3,a = 10))

# Plot the model curve
simulation.data <- data.frame(t = seq(min(t),max(t),len = 100))
lines(simulation.data$t,predict(model,
      newdata = simulation.data), col = "red", lwd = 3)
```

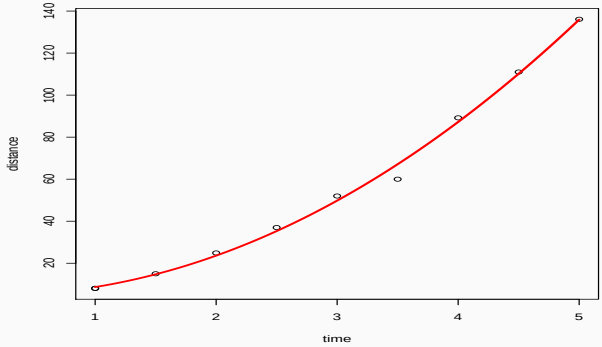


Figure 4: The results of the non-linear regression with nls(). This plot indicates that there is one outlier and you might want to rerun the model without this observation.

The model seems to fit quite well the data. As usual, we can extract more information from the model object via the functions `summary()` and/or `print()`.

Example for nls() iv

```
# Learn about the model:
summary(model)                # the summary
##
## Formula: dt ~ d0 + v0 * t + 1/2 * a * t^2
##
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## d0    4.981      4.660   1.069    0.321
## v0   -1.925      3.732  -0.516    0.622
## a    11.245      1.269   8.861 4.72e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.056 on 7 degrees of freedom
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 1.822e-07

print(sum(residuals(model)^2))# squared sum of residuals
## [1] 65.39269

print(confint(model))         # confidence intervals
##          2.5%     97.5%
## d0  -6.038315 15.999559
## v0 -10.749091  6.899734
## a    8.244167 14.245927
```

PART 05: MODELLING

↓

CHAPTER 21: REGRESSION MODELS

↓

SECTION 3:

Performance of Regression Models

Mean Square Error (MSE)

Definition 2 (Mean Square Error (MSE))

The means square error is the average residual variance. The following is a predictor:

$$\text{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{k=1}^{N} (y_k - \hat{y})^2$$

Definition 3 (*R*-squared)

R-squared is the the proportion of the variance in the dependent variable that is predictable from the independent variable(s). We can calculate *R*-squared as:

$$R^2 = 1 - \frac{\sum_{k=1}^{N} (y_k - \hat{y})^2}{\sum_{k=1}^{N} (y_k - \bar{y})^2}$$

with $\hat{y_k}$ the estimate for observation y_k based on our model, and $\bar{y_k}$ the mean of all observations y_k.

Example

```r
m <- lm(data = mtcars, formula = mpg ~ wt)
summary(m)
##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
## wt           -5.3445     0.5591  -9.559 1.29e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528,Adjusted R-squared:  0.7446
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10

summary(m)$r.squared
## [1] 0.7528328
```

Question #3 – Find a better model

Use the dataset `mtcars` (from the library MASS), and try to find the model that best explains the consumption (mpg).

Definition 4 (Mean average deviation (MAD))

$$\text{MAD}(y, \hat{y}) := \frac{1}{N} \sum_{k=1}^{N} |y_k - \hat{y}|$$

part 05: Modelling

↓

chapter 22:

Classification Models

PART 05: MODELLING

↓

CHAPTER 22: CLASSIFICATION MODELS

↓

SECTION 1:

Logistic Regression

Definition 5 (– Generalised logistic regression)

A logistic regression, is a regression of the log-odds:

$$\ln\left\{\frac{P[Y=1|X]}{P[Y=0|X]}\right\} = \alpha + \sum_{n=1}^{N} f_n(X_n)$$

with $X = (X_1, X_2, \ldots, X_N)$ the set of prognostic factors.

Definition 6 (– Additive logistic regression)

Assuming a linear model for the f_n such that , the probability that $Y = 1$ is modelled as:

$$y = \frac{1}{1 + e^{-(b + a_1 x_1 + a_2 x_2 + a_3 x_3 + \cdots)}}$$

This regression can be fitted with the function `glm()`, that we encountered earlier.

```
# Consider the relation between the hours studied and passing
# an exam (1) or failing it (0):

# First prepare the data:
hours <- c(0,0.50, 0.75, 1.00, 1.25, 1.50, 1.75,
           1.75, 2.00, 2.25, 2.50, 2.75, 3.00, 3.25,
           3.50, 4.00, 4.25, 4.50, 4.75, 5.00, 5.50)
pass  <- c(0,0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
           1, 0, 1, 1, 1, 1, 1, 1)
d <- data.frame(cbind(hours,pass))

# Then fit the model:
m <- glm(formula = pass ~ hours, family = binomial,
         data = d)
```

```r
# Visualize the results:
plot(hours, pass, col = "red", pch = 23, bg = "grey",
     xlab = 'Hours studied',
     ylab = 'Passed exam (1) or not (0)')
pred <- 1 / (1+ exp(-(coef(m)[1] + hours * coef(m)[2])))
lines(hours, pred, col = "blue", lwd = 4)
```

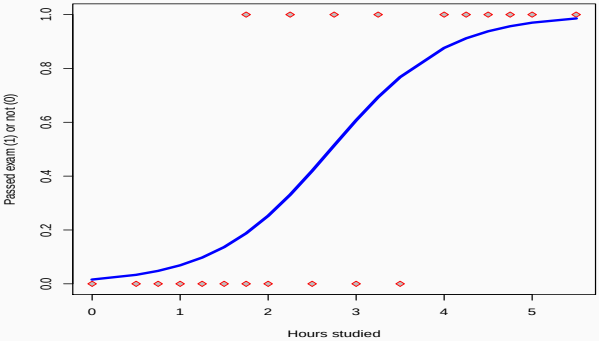


Figure 5: The grey diamonds with red border are the data-points (not passed is 0 and passed is 1) and the blue line represents the logistic regression model (or the probability to succeed the exam in function of the hours studied.

PART 05: MODELLING

↓

CHAPTER 22: CLASSIFICATION MODELS

↓

SECTION 2:

Performance of Binary Classification Models

The Example for this Section i

In the following sections we will use the dataset from the package `titanic`. This is data of the passengers on the RMS Titanic, that sunk in 1929 in the Northern Atlantic Ocean after a collision with an iceberg.

The data can be unlocked as follows:

```
# if necessary: install.packages('titanic')
library(titanic)

# This provides a.o. two datasets titanic_train and titanic_test.
# We will work further with the training-dataset.
t <- titanic_train
colnames(t)
##  [1] "PassengerId" "Survived"    "Pclass"      "Name"
##  [5] "Sex"         "Age"         "SibSp"       "Parch"
##  [9] "Ticket"      "Fare"        "Cabin"       "Embarked"
```

Fitting a Logistic Regression on the Titanic data

```
# fit provide a simple model
m <- glm(data    = t,
         formula = Survived ~ Pclass + Sex + Pclass * Sex + Age + SibSp,
         family  = binomial)
summary(m)
##
## Call:
## glm(formula = Survived ~ Pclass + Sex + Pclass * Sex + Age +
##     SibSp, family = binomial, data = t)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3507  -0.6574  -0.4438   0.4532   2.3450
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     8.487528   0.996601   8.516  < 2e-16 ***
## Pclass         -2.429192   0.330221  -7.356 1.89e-13 ***
## Sexmale        -6.162294   0.929696  -6.628 3.40e-11 ***
## Age            -0.046830   0.008603  -5.443 5.24e-08 ***
## SibSp          -0.354855   0.120373  -2.948   0.0032 **
## Pclass:Sexmale  1.462084   0.349338   4.185 2.85e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 964.52  on 713  degrees of freedom
```

Useful Concepts for the Confusion Matrix

The following are useful measures for how good a classification model fits its data:

- *Accuracy:* The proportion of predictions that were correctly identified.
- *Precision* (or positive predictive value): The proportion of positive cases that correct.
- *Negative predictive value:* The proportion of negative cases that were correctly identified.
- *Sensitivity* or Recall: The proportion of actual positive cases which are correctly identified.
- *Specificity:* The proportion of actual negative cases which are correctly identified.

Some Acronyms for the Confusion Matrix

Let us use the following definitions:

- Objective concepts (depends only on the data):
  - *P:* The number of positive observations ($y = 1$)
  - *N:* The number of negative observations ($y = 0$)
- Model dependent definitions:
  - True positive (TP) the positive observations ($y = 1$) that are by the model correctly classified as positive;
  - False positive (FP) the negative observations ($y = 0$) that are by the model incorrectly classified as positive – this is a false alarm (Type I error);
  - True negative (TN) the negative observations ($y = 0$) that are by the model correctly classified as negative;
  - False negative (FN) the positive observations ($y = 1$) that are by the model incorrectly classified as negative – miss (Type II error).

The Definition of the Confusion Matrix

	Observed pos.	**Observed neg.**	
Pred. pos.	TP	FP	Pos.pred.val = $\frac{\text{TP}}{\text{TP}+\text{FP}}$
Pred. neg.	FN	TN	Neg.pred.val = $\frac{\text{TN}}{\text{FN}+\text{TN}}$
	Sensitivity $= \frac{\text{TP}}{\text{TP}+\text{FN}}$ $= \frac{\text{TP}}{\text{TP}+\text{FN}}$	Specificity $= \frac{\text{TN}}{\text{FP}+\text{TN}}$ $= \frac{\text{TN}}{\text{FP}+\text{TN}}$	Accuracy $= \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FN}+\text{FP}+\text{TN}}$ $= \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FN}+\text{FP}+\text{TN}}$

Table 1: The confusion matrix, where "pred." refers to the predictions made by the model, "pred." stands for "predicted," and the words "positive" and "negative" are shortened to three letters.

The Confusion Matrix in R

```r
# We build further on the model m.

# Predict scores between 0 and 1 (odds):
t2 <- t[complete.cases(t),]
predicScore <- predict(object=m,type="response", newdat = t2)

# Introduce a cut-off level above which we assume survival:
predic <- ifelse(predicScore > 0.7, 1, 0)

# The confusion matrix is one line, the headings 2:
confusion_matrix <- table(predic, t2$Survived)
rownames(confusion_matrix) <- c("predicted_death",
                                "predicted_survival")
colnames(confusion_matrix) <- c("observed_death",
                                "observed_survival")

# Display the result:
print(confusion_matrix)
##
## predic               observed_death observed_survival
##   predicted_death              414               134
##   predicted_survival            10               156
```

- **TPR** = True Positive Rate = sensitivity = recall = hit rate = probability of detection

$$\mathrm{TPR} = \frac{\mathrm{TP}}{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

- **FPR** = False Positive Rate = fallout = 1 - Specificity

$$\mathrm{FPR} = \frac{\mathrm{FP}}{N} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

- **TNR** = specificity = selectivity = true negative rate

$$\mathrm{TNR} = \frac{\mathrm{TN}}{N} = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{FPR}$$

- **FNR** = false negative rate = miss rate

$$\mathrm{FNR} = \frac{\mathrm{FN}}{P} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{TPR}$$

- **Precision** = positive predictive value = PPV

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

- **Negative predictive value** = NPV

$$\mathit{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}}$$

- **ACC** = accuracy

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{N + P} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

- **F_1 score** = harmonic mean of precision and sensitivity

$$\mathrm{F}_1 = \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\,\mathrm{TP}}{2\,\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Definitions for the ROC Curve

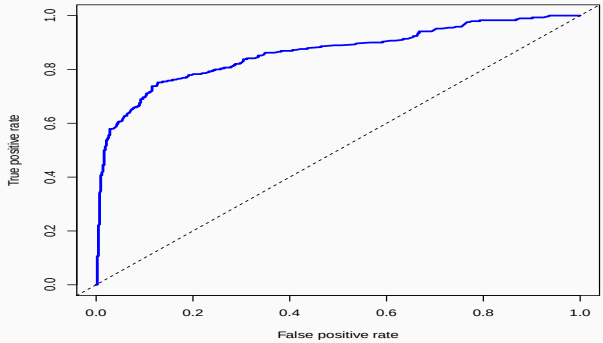
The ROC curve is formed by plotting the true positive rate (TPR) against the false positive rate (FPR) at various cut-off levels.[1] Formally, the ROC curve is the interpolated curve made of points whose coordinates are functions of the threshold: threshold $= \theta \in \mathbb{R}$, here $\theta \in [0, 1]$

$$\mathrm{ROC}_x(\theta) = \mathrm{FPR}(\theta) = \frac{\mathrm{FP}(\theta)}{\mathrm{FP}(\theta) + \mathrm{TN}(\theta)} = \frac{\mathrm{FP}(\theta)}{\#N}$$

$$\mathrm{ROC}_y(\theta) = \mathrm{TPR}(\theta) = \frac{\mathrm{TP}(\theta)}{\mathrm{FN}(\theta) + \mathrm{TP}(\theta)} = \frac{\mathrm{FP}(\theta)}{\#P} = 1 - \frac{\mathrm{FN}(\theta)}{\#P} = 1 - \mathrm{FNR}(\theta)$$

Visualising the ROC Curve in Base R

```
library(ROCR)
# Re-use the model m and the dataset t2:
pred <- prediction(predict(m, type = "response"), t2$Survived)

# Visualize the ROC curve:
plot(performance(pred, "tpr", "fpr"), col="blue", lwd = 3)
abline(0, 1, lty = 2)
```



Note: The Performance Object is an S4 Object

This object will know how it can be plotted (or rather “the function plot will dispactch th the relvant method”). If necessary, then it can be converted to an a data frame as follows:

```
S4_perf <- performance(pred, "tpr", "fpr")
df <- data.frame(
   x = S4_perf@x.values,
   y = S4_perf@y.values,
   a = S4_perf@alpha.values
   )
colnames(df) <- c(S4_perf@x.name, S4_perf@y.name, S4_perf@alpha.name)
head(df)
##   False positive rate True positive rate    Cutoff
## 1         0.000000000        0.000000000       Inf
## 2         0.002358491        0.000000000 0.9963516
## 3         0.002358491        0.003448276 0.9953019
## 4         0.002358491        0.013793103 0.9950778
## 5         0.002358491        0.017241379 0.9945971
## 6         0.002358491        0.024137931 0.9943395
```

In a final report, it might be desirable to use the power of ggplot2 consistently. In the following code we illustrate how this a ROC curve can be obtained in ggplot2.[2] The plot is in Figure 7 on slide 53.

```
library(ggplot2)
p <- ggplot(data=df,
            aes(x = `False positive rate`, y = `True positive rate`)) +
            geom_line(lwd=2, col='blue')  +
            # The next lines add the shading:
            aes(x = `False positive rate`, ymin = 0,
                ymax = `True positive rate`) +
            geom_ribbon(, alpha=.5)
p
```

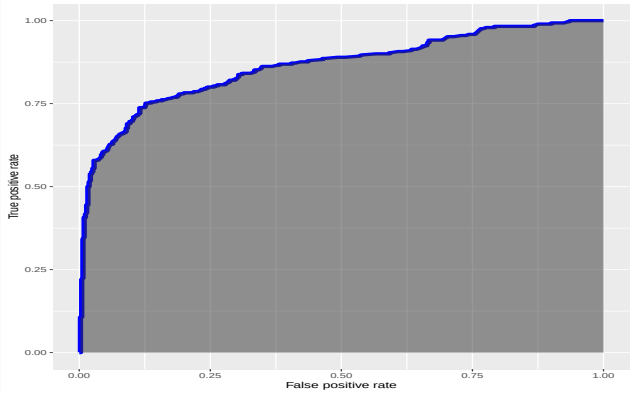


Figure 7: The ROC curve plotted with ggplot2.

The performance object can also provide the accuracy of the model, and this can be plotted as follows – note that the plot is in Figure 8.

```
# Plotting the accuracy (in function of the cut-off)
plot(performance(pred, "acc"), col="blue", lwd = 3)
```

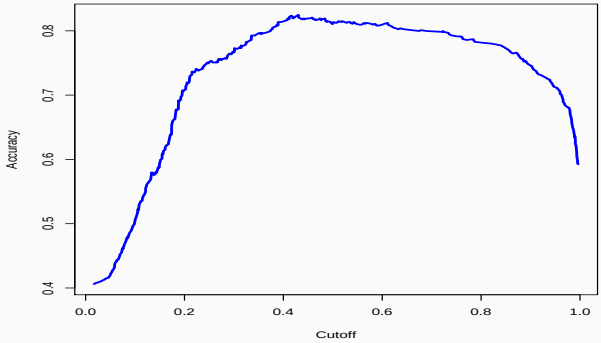


Figure 8: A plot of the accuracy in function of the cut-off (threshold) level.

```r
# Assuming that we have the predictions in the prediction object:
plot(performance(pred, "tpr", "fpr"), col = "blue", lwd = 4)
abline(0, 1, lty = 2, lwd = 3)
x <- c(0.3, 0.1, 0.8)
y <- c(0.5, 0.9, 0.3)
text(x, y, labels = LETTERS[1:3], font = 2, cex = 3)
```

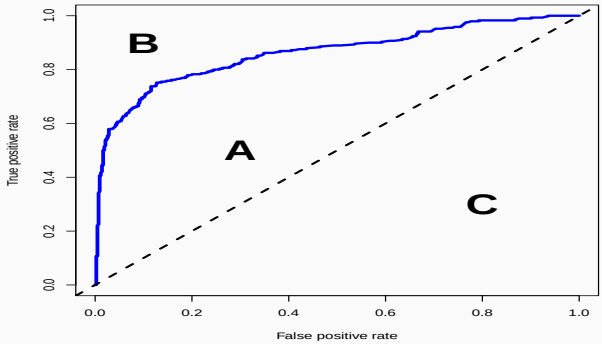


Figure 9: The area under the curve (AUC) is the area A plus the area C. In next section we characterise the Gini coeffient, which equals area A divided by area C.

```
# Note: instead you can also call the function text() three times:
# text(x = 0.3, y = 0.5, labels = "A", font = 2, cex = 3)
# text(x = 0.1, y = 0.9, labels = "B", font = 2, cex = 3)
# text(x = 0.8, y = 0.3, labels = "C", font = 2, cex = 3)
```

The AUC in R

In R, the AUC in R is provided by the `performance()` function of `ROCR` and stored in the performance object. It is an S4 object, and hence we can extract the information as follows.

```
AUC <- attr(performance(pred, "auc"), "y.values")[[1]]
AUC
## [1] 0.8615241
```

The Gini Coefficient in R

In R, extracting the Gini coefficient from the performance object is trivial, given the AUC that we calculated before. In fact, we can use the AUC to obtain the Gini:

```
paste("the Gini is:",round(2 * AUC - 1, 2))
## [1] "the Gini is: 0.72"
```

The Definition of KS

The Kolmogorov-Smirnov (KS) test is another measure that aims to summarize the power of a model in one parameter. In general, the KS is the largest distance between two cumulative distribution functions:

$$\text{KS} = \sup |F_1(x) - F_2(x)|$$

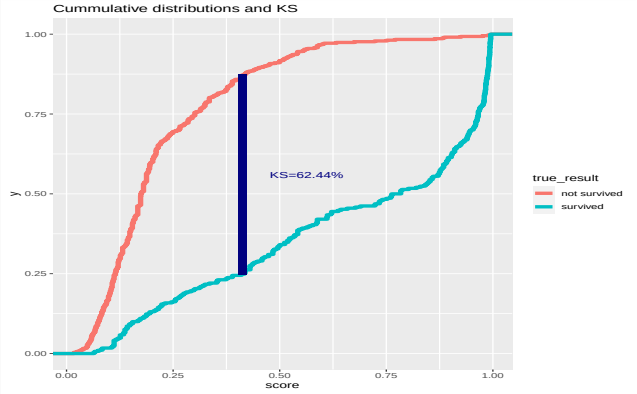


Figure 10: The KS as the maximum distance between the cumulative distributions of the positive and negative observations.

Calculating the KS in R

The package `stats` from base R provides the functions `ks.test()` to calculate the KS.

```
pred <- prediction(predict(m,type="response"), t2$Survived)
ks.test(attr(pred,"predictions")[[1]],
        t2$Survived,
        alternative = 'greater')
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  attr(pred, "predictions")[[1]] and t2$Survived
## D^+ = 0.40616, p-value < 2.2e-16
## alternative hypothesis: the CDF of x lies above that of y
```

As you can see in the aforementioned code, this does not work in some cases. Fortunately, it is easy to construct an alternative:

```
perf <- performance(pred, "tpr", "fpr")
ks <- max(attr(perf,'y.values')[[1]] - attr(perf,'x.values')[[1]])
ks
## [1] 0.6243656

# Note: the following line yields the same outcome
ks <- max(perf@y.values[[1]] - perf@x.values[[1]])
ks
## [1] 0.6243656
```

Naive Function to find the Optimal Cutoff i

```r
# get_best_cutoff
# Finds a cutof for the score so that sensitivity and specificity
# are optimal.
# Arguments
#   fpr    -- numeric vector -- false positive rate
#   tpr    -- numeric vector -- true positive rate
#   cutoff -- numeric vector -- the associated cutoff values
# Returns:
#   the cutoff value (numeric)
get_best_cutoff <- function(fpr, tpr, cutoff){
        cst <- (fpr - 0)^2 + (tpr - 1)^2
        idx = which(cst == min(cst))
        c(sensitivity = tpr[[idx]],
          specificity = 1 - fpr[[idx]],
          cutoff = cutoff[[idx]])
    }

# opt_cut_off
# Wrapper for get_best_cutoff. Finds a cutof for the score so that
# sensitivity and specificity are optimal.
# Arguments:
#    perf -- performance object (ROCR package)
#    pred -- prediction object (ROCR package)
# Returns:
#   The optimal cutoff value (numeric)
opt_cut_off = function(perf, pred){
    mapply(FUN=get_best_cutoff,
           perf@x.values,
           perf@y.values
```

Naive Function to find the Optimal Cutoff ii

We can now test the function as follows:

```
opt_cut_off(perf, pred)
##                   [,1]
## sensitivity 0.7517241
## specificity 0.8726415
## cutoff      0.4161801
```

The Optimal Cutoff in case of Dissimilar Costs for FPs and FNs i

```r
# We introduce cost.fp to be understood as a the cost of a
# false positive, expressed as a multiple of the cost of a
# false negative.

# get_best_cutoff
# Finds a cutof for the score so that sensitivity and specificity
# are optimal.
# Arguments
#   fpr     -- numeric vector -- false positive rate
#   tpr     -- numeric vector -- true positive rate
#   cutoff  -- numeric vector -- the associated cutoff values
#   cost.fp -- numeric        -- cost of false positive divided
#                                by the cost of a false negative
#                                (default = 1)
# Returns:
#   the cutoff value (numeric)
get_best_cutoff <- function(fpr, tpr, cutoff, cost.fp = 1){
        cst <- (cost.fp * fpr - 0)^2 + (tpr - 1)^2
        idx = which(cst == min(cst))
        c(sensitivity = tpr[[idx]],
          specificity = 1 - fpr[[idx]],
          cutoff = cutoff[[idx]])
    }
```

The Optimal Cutoff in case of Dissimilar Costs for FPs and FNs ii

```
# opt_cut_off
# Wrapper for get_best_cutoff. Finds a cutof for the score so that
# sensitivity and specificity are optimal.
# Arguments:
#    perf    -- performance object (ROCR package)
#    pred    -- prediction object (ROCR package)
#    cost.fp -- numeric -- cost of false positive divided by the
#                          cost of a false negative (default = 1)
# Returns:
#   The optimal cutoff value (numeric)
opt_cut_off = function(perf, pred, cost.fp = 1){
    mapply(FUN=get_best_cutoff,
           perf@x.values,
           perf@y.values,
           pred@cutoffs,
           cost.fp)
   }
```

The Optimal Cutoff in case of Dissimilar Costs for FPs and FNs iii

When false positives are more (or less) expensive than false negatives, then we can use our funtion as follows:

```
# Test the function:
opt_cut_off(perf, pred, cost.fp = 5)
##                   [,1]
## sensitivity 0.5793103
## specificity 0.9716981
## cutoff      0.6108004
```

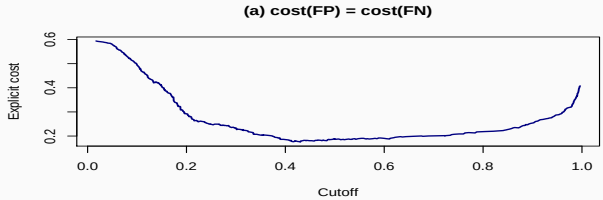
Using ROCR with Dissimilar Costs for FPs and FNs

```
# e.g. cost.fp = 1 x cost.fn
perf_cst1 <- performance(pred, "cost", cost.fp = 1)
str(perf_cst1) # the cost is in the y-values
## Formal class 'performance' [package "ROCR"] with 6 slots
##   ..@ x.name      : chr "Cutoff"
##   ..@ y.name      : chr "Explicit cost"
##   ..@ alpha.name  : chr "none"
##   ..@ x.values    :List of 1
##   .. ..$ : Named num [1:410] Inf 0.996 0.995 0.995 0.995 ...
##   .. .. ..- attr(*, "names")= chr [1:410] "" "298" "690" "854" ...
##   ..@ y.values    :List of 1
##   .. ..$ : num [1:410] 0.406 0.408 0.406 0.402 0.401 ...
##   ..@ alpha.values: list()

# the optimal cut-off is then the same as in previous code sample
pred@cutoffs[[1]][which.min(perf_cst1@y.values[[1]])]
##       738
## 0.4302302

# e.g. cost.fp = 5 x cost.fn
perf_cst2 <- performance(pred, "cost", cost.fp = 5)

# the optimal cut-off is now:
pred@cutoffs[[1]][which.min(perf_cst2@y.values[[1]])]
##       306
## 0.7231593
```

Plotting Cost Information i

```
par(mfrow=c(2,1))
plot(perf_cst1, lwd=2, col='navy', main='(a) cost(FP) = cost(FN)')
plot(perf_cst2, lwd=2, col='navy', main='(b) cost(FP) = 5 x cost(FN)')
```

Plotting Cost Information ii



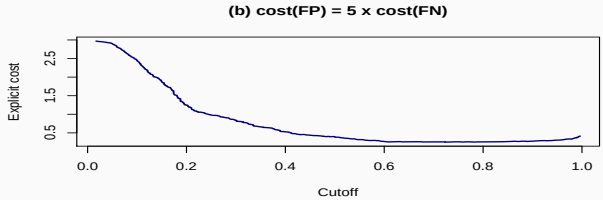


Figure 11: The cost functions compared different cost structures. In plot (a), we plotted the cost function when the cost of a false positive is equal to the cost of a false negative. In plot (b), a false positive costs five times more than a false negative (valid for a loan in a bank).

```
par(mfrow=c(1,1))
```

part 05: Modelling

↓

chapter 23:

Learning Machines

- *Supervised learning:* The algorithm will learn from provided results (e.g. we have data of good and bad credit customers)
- *Unsupervised learning:* The algorithm groups observations according to a given criteria (e.g. the algorithm classifies customers according to profitability without being told what good or bad is).
- *Reinforced learning:* The algorithm learns from outcomes: rather than being told what is good or bad, the system will get something like a cost-function (e.g. the result of a treatment, the result of a chess game, or the relative return of a portfolio of investments in a competitive stock market). Another way of defining reinforced learning is that in this case, the environment rather than the teacher provides the right outcomes.

PART 05: MODELLING

↓

CHAPTER 23: LEARNING MACHINES

↓

SECTION 1:

Decision Tree

$$\hat{y} = \hat{f}(x) = \sum_{n=1}^{N} \alpha_n I\{x \in R_n\}$$

with $x = (x_1, \ldots, x_m)$ and $I\{b\}$ the identity function so that $I\{b\} := \begin{cases} 1 & \text{if } b \\ 0 & \text{if } !b \end{cases}$

Visual representation of the decision tree

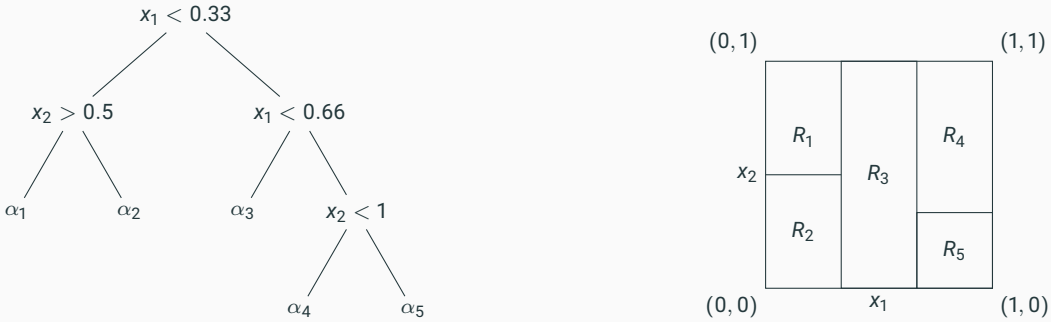


Figure 12: An example of the decision tree on fake data represented in two ways: on the left the decision tree and on the right the regions R_i that can be identified in the (x_1, x_2)-plane.

1. goodness of fit: SS: $\sum (y_i - f(x_i))^2$
2. estimate in each region R_i: $\hat{y}_i = avg(y_i | x_i \in R_i)$
3. best split: $\min_{j,s} \left[\min_{y_1} \sum_{x_i \in R_1(j,s)} (y_i - y_1)^2 + min_{y_1} \sum_{x_i \in R_2(j,s)} (y_i - y_2)^2 \right]$
4. For any pair (j, s) we can solve the minimizations with average as estimator: $\begin{cases} \hat{y_1} = avg[y_i | x_i \in R_1(j,s)] \\ \hat{y_2} = avg[y_i | x_i \in R_2(j,s)] \end{cases}$

The idea is to minimize the "cost of complexity function" for a given pruning parameter α. The cost function is defined as

$$C_\alpha(T) := \sum_{n=1}^{|E_T|} \mathrm{SE}_n(T) + \alpha|T| \tag{1}$$

This is the sum of squares in each end-note plus α times the size of the tree. $|T|$ is the number of terminal nodes in the sub-tree T (T is a subtree to T_0 if T has only nodes of T_0), $|E_T|$ is the number of end-nodes in the tree T and $\mathrm{SE}_n(T)$ is the sum of squares in the end-node n for the tree T. The square errors in node n (or in region R_n) also equals:

$$\begin{aligned}
\mathrm{SE}_n(T) &= N_n \, \mathrm{MSE}_n(T) \\
&= N_n \, \frac{1}{N_n} \sum_{x_i \in R_n}^{N_n} (y_i - \hat{y}_n)^2 \\
&= \sum_{x_i \in R_n}^{N_n} (y_i - \hat{y}_n)^2
\end{aligned}$$

with $\hat{y}_n$ the average of all y_i in the region n as explained previously.

In case the values y_i do not come from a numerical function but are rather a nominal or ordinal scale,[3] it is no longer possible to use MSE as a measure of fitness for the model. In that case, we can use the average number of matches with class c:

$$\hat{p}_{n,c} := \frac{1}{N_c} \sum_{x_i \in R_n} I\{y_i = c\} \tag{2}$$

The class c that has the highest proportion $\hat{p}_{n,c}$, is defined as $argmax_c(\hat{p}_{m,k})$. This is the value that we will assign in that node. The node impurity then can be calculated by one of the following:

$$\text{Gini index} = \sum_{c \neq \tilde{c}} \hat{p}_{n,c} \hat{p}_{n,\tilde{c}} \tag{3}$$

$$= \sum_{c=1}^{C} \hat{p}_{n,c}(1 - \hat{p}_{n,c}) \tag{4}$$

$$\text{Cross-entropy or deviance} = -\sum_{c=1}^{C} \hat{p}_{n,c} \log_2(\hat{p}_{n,c}) \tag{5}$$

$$\text{Misclassification error} = \frac{1}{N_n} \sum_{x_i \in R_n} I\{y_i = c\} \tag{6}$$

$$= 1 - \hat{p} \tag{7}$$

While largely covered by the explanation above, it is worth to take a few minutes and study the particular case where the output variable is binary: true or false, good or bad, 0 or 1. This is not only a very important case, but it also allows us to make the parallel with information theory.

Binary classifications are important cases in everyday practice: good or bad credit risk, sick or not, death or alive, etc.

The mechanism to fit the tree works exactly the same. From all attributes, choose that one that classifies the results the best. Split the dataset according to the value that bests separates the goods from the bads.

We need a way to tell what is a good split. This can be done by selecting the attribute that has the most information value. The information – measured in bits – of outcomes x_i with probabilities P_i is

$$I(P_1, \ldots, P_N) = -\sum_{i=1}^{N} P_i \log_2(P_i)$$

Which in the case of two possible outcomes (G the number of "good" observations and B the number of "bad" observations) reduces to

$$I\left(\frac{G}{G+B}, \frac{B}{G+B}\right) = -\frac{G}{G+B}\log_2\left(\frac{G}{G+B}\right) - \frac{B}{G+B}\log_2\left(\frac{B}{G+B}\right)$$

1. **Loss matrix**
2. **Missing values**
3. **Linear combination splits**
4. **Link with ANOVA:** An alternative way to understand the ideal stopping point is using the ANOVA approach. The impurity in a node can be thought of as the MSE in that node.

$$\text{MSE} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

with y_i the value of the i^{th} observation and $\bar{y}$ the average of all observations.
This node impurity can also be thought of as in ANOVA analyses.

$$\frac{\frac{\text{SS}_{\text{between}}}{B-1}}{\frac{\text{SS}_{\text{within}}}{n-B}} \sim F_{n-B,B-1}$$

with

$$\begin{cases} \text{SS}_{\text{between}} & = n_b \sum_{b=1}^{B} (\bar{y}_b - \bar{y})^2 \\ \text{SS}_{\text{within}} & = \sum_{b=1}^{B} \sum_{i=1}^{n_b} (\bar{y}_{bi} - \bar{y})^2 \end{cases}$$

with B the number of branches, n_b the number of observations in branch b, $y_b i$ the value of observation bi.
Now, optimal stopping can be determined by using measures of fit and relevance as in a linear regression

Issues

1. **Over-fitting:** this is one of the most important issues with decision trees. It should never be used without appropriate validation methods such as cross validation or random forest approach before an effort to prune the tree.
2. **Categorical predictor values**
3. **Instability**
4. **Difficulties to capture additive relationships**
5. **Stepwise predictions**

Function use for rpart()

```
rpart(formula, data, weights, subset, na.action = na.rpart,
      method=c('class','anova'), model = FALSE,
      x = FALSE, y = TRUE, parms, control, cost, ...)
```

with the most important parameters:

- `data:` the data-frame containing the columns to be used in `formula`.
- `formula:` am R-formula of the form `y ~ x1 + x1 + ...` – note that the plus signs do not really symbolise the addition here, but only indicate which columns to choose.
- `weights:` optional case weights.
- `subset:` optional expression that indicates which section of the data should be used.
- `na.action:` optional information on what to do with missing values. The default is `na.rpart`, which means that all rows with `y` missing will be deleted, but any x_i can be missing.
- `method:` optional method such as “anova,” “poisson,” “class” (for classification tree), or “exp”. If it is missing, a reasonably guess will be made, based on the nature of `y`.

As usual, more information is in the documentation of the function and the package.

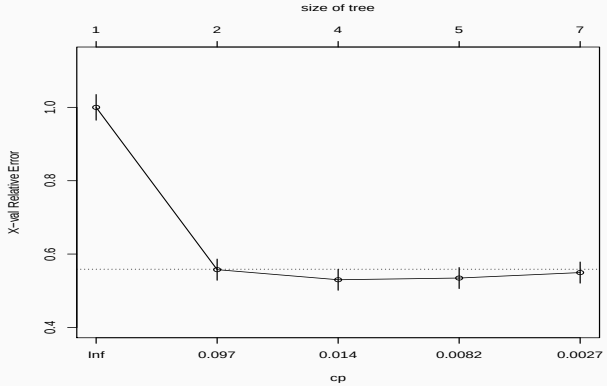
Example of a Classification Tree with `rpart` i

```
## example of a regression tree with rpart on the dataset of the Titanic
##
library(rpart)
titanic <- read.csv("../../data/titanic3.csv")
frm     <- survived ~ pclass + sex + sibsp + parch + embarked + age
t0      <- rpart(frm, data=titanic, na.action = na.rpart,
  method="class",
  parms = list(prior = c(0.6,0.4)),
  #weights=c(...), # each observation (row) can be weighted
  control = rpart.control(
  minsplit       = 50,  # minimum nbr. of observations required for split
  minbucket      = 20,  # minimum nbr. of observations in a terminal node
  cp             = 0.001,# complexity parameter set to a small value
                         # this will grow a large (over-fit) tree
  maxcompete     = 4,   # nbr. of competitor splits retained in output
  maxsurrogate   = 5,   # nbr. of surrogate splits retained in output
  usesurrogate   = 2,   # how to use surrogates in the splitting process
  xval           = 7,   # nbr. of cross validations
  surrogatestyle = 0,   # controls the selection of a best surrogate
  maxdepth       = 6)   # maximum depth of any node of the final tree
  )
```

Example of a Classification Tree with rpart ii

```
# Show details about the tree t0:
printcp(t0)
##
## Classification tree:
## rpart(formula = frm, data = titanic, na.action = na.rpart, method = "class",
##     parms = list(prior = c(0.6, 0.4)), control = rpart.control(minsplit = 50,
##         minbucket = 20, cp = 0.001, maxcompete = 4, maxsurrogate = 5,
##         usesurrogate = 2, xval = 7, surrogatestyle = 0, maxdepth = 6))
##
## Variables actually used in tree construction:
## [1] age      embarked pclass   sex      sibsp
##
## Root node error: 523.6/1309 = 0.4
##
## n= 1309
##
##          CP nsplit rel error  xerror     xstd
## 1 0.4425241      0   1.00000 1.00000 0.035158
## 2 0.0213115      1   0.55748 0.55748 0.029038
## 3 0.0092089      3   0.51485 0.52998 0.028819
## 4 0.0073337      4   0.50564 0.53462 0.028806
## 5 0.0010000      6   0.49098 0.54952 0.028945

# Plot the error in function of the complexity parameter
plotcp(t0)
```

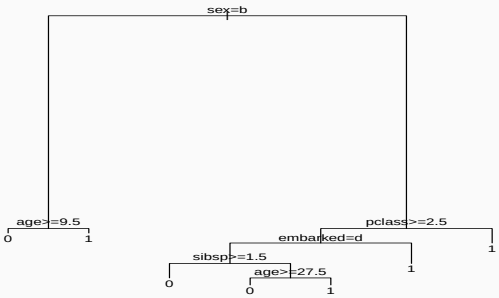
Example of a Classification Tree with `rpart` iii



Example of a Classification Tree with `rpart` iv

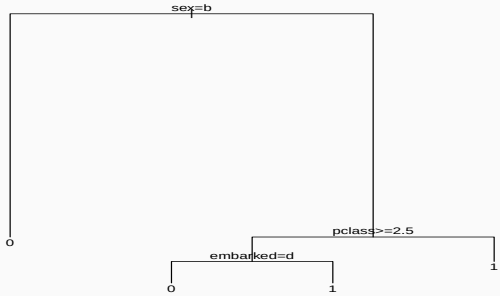
```
# print(t0) # to avoid too long output we commented this out
# summary(t0)

# Plot the original decisions tree
plot(t0)


text(t0)
```

Example of a Classification Tree with rpart v




```r
# Prune the tree:
t1 <- prune(t0, cp=0.01)
plot(t1); text(t1)
```

Example of a Classification Tree with `rpart` vi



Visualizing the tree with `rpart.plot` i

```r
# plot the tree with rpart.plot
library(rpart.plot)
prp(t0, type = 5, extra = 8, box.palette = "auto",
    yesno = 1, yes.text="survived",no.text="dead"
    )
```

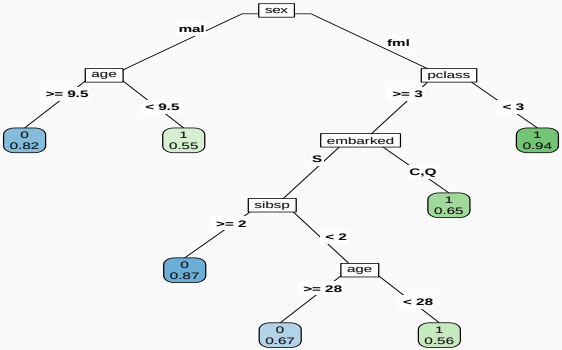


Figure 13: The decision tree represented by the function `prp()` from the package `rpart.plot`. This plot not only looks more elegant, but it is also more informative and less simplified. For example the top node "sex" has now two clear options in which descriptions we can recognize the words male and female, and the words are on the branches, so there is no confustion possible which is left and which right.

Example of a regression tree with rpart i

```
# Example of a regression tree with rpart on the dataset mtcars

# The libraries should be loaded by now:
library(rpart); library(MASS); library (rpart.plot)

# Fit the tree:
t <- rpart(mpg ~ cyl + disp + hp + drat + wt + qsec + am + gear,
 data=mtcars, na.action = na.rpart,
 method     = "anova",
 control    = rpart.control(
   minsplit       = 10,  # minimum nbr. of observations required for split
   minbucket      = 20/3,# minimum nbr. of observations in a terminal node
                         # the default = minsplit/3
   cp             = 0.01,# complexity parameter set to a very small value
                         # his will grow a large (over-fit) tree
   maxcompete     = 4,   # nbr. of competitor splits retained in output
   maxsurrogate   = 5,   # nbr. of surrogate splits retained in output
   usesurrogate   = 2,   # how to use surrogates in the splitting process
   xval           = 7,   # nbr. of cross validations
   surrogatestyle = 0,   # controls the selection of a best surrogate
   maxdepth       = 30   # maximum depth of any node of the final tree
   )
 )

# Investigate the complexity parameter dependence:
printcp(t)
##
## Regression tree:
```

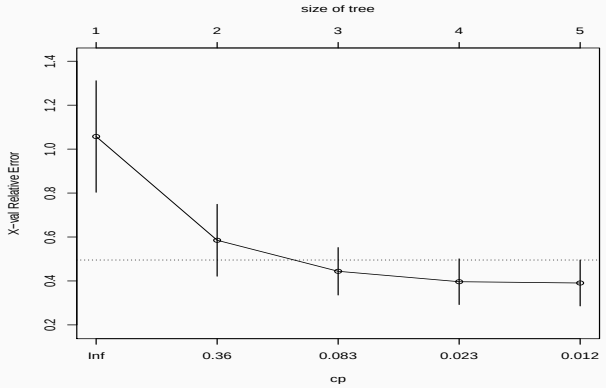


Figure 14: The plot of the complexity parameter (cp) via the function `plotcp()`

Example of a regression tree with `rpart` iii

```
# Print the tree:
print(t)
## n= 32
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 32 1126.04700 20.09062
##    2) wt>=2.26 26  346.56650 17.78846
##      4) cyl>=7 14   85.20000 15.10000
##        8) hp>=192.5 7   28.82857 13.41429 *
##        9) hp< 192.5 7   16.58857 16.78571 *
##      5) cyl< 7 12   42.12250 20.92500
##       10) disp>=153.35 6   12.67500 19.75000 *
##       11) disp< 153.35 6   12.88000 22.10000 *
##    3) wt< 2.26 6   44.55333 30.06667 *

summary(t)
## Call:
## rpart(formula = mpg ~ cyl + disp + hp + drat + wt + qsec + am +
##     gear, data = mtcars, na.action = na.rpart, method = "anova",
##     control = rpart.control(minsplit = 10, minbucket = 20/3,
##         cp = 0.01, maxcompete = 4, maxsurrogate = 5, usesurrogate = 2,
##         xval = 7, surrogatestyle = 0, maxdepth = 30))
##   n= 32
##
##           CP nsplit rel error    xerror      xstd
## 1 0.65266121      0 1.0000000 1.0574288 0.2539755
```

PART 05: MODELLING

↓

CHAPTER 23: LEARNING MACHINES

↓

SECTION 2:

Random Forest

To fit a random forest in R, we can rely on the package `randomforest`:

```
library(randomForest)
```

```r
head(mtcars)
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1

mtcars$l <- NULL  # remove our variable
frm      <- mpg ~ cyl + disp + hp + drat + wt + qsec + am + gear
set.seed(1879)

# Fit the random forest:
forestCars   = randomForest(frm, data = mtcars)

# Show an overview:
print(forestCars)
##
## Call:
##  randomForest(formula = frm, data = mtcars)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 6.001878
##                     % Var explained: 82.94

# Plot the random forest overview:
plot(forestCars)
```

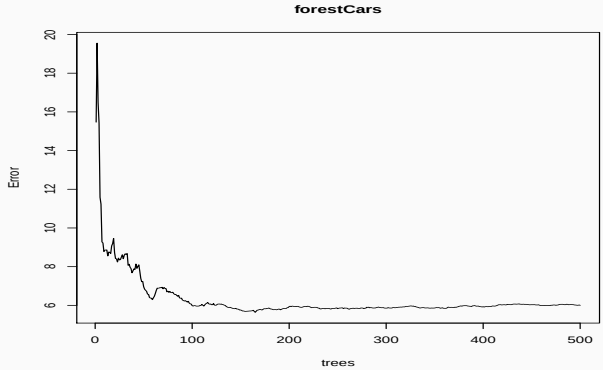


Figure 16: The plot of a randomForest object shows how the model improves in function of the number of trees used.

```r
# Show the summary of fit:
summary(forestCars)
##                 Length Class  Mode
## call              3    -none- call
## type              1    -none- character
## predicted        32    -none- numeric
## mse             500    -none- numeric
## rsq             500    -none- numeric
## oob.times        32    -none- numeric
## importance        8    -none- numeric
## importanceSD      0    -none- NULL
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           11    -none- list
## coefs             0    -none- NULL
## y                32    -none- numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call

# visualization of the RF:
getTree(forestCars, 1, labelVar=TRUE)
##    left daughter right daughter split var split point status
## 1              2              3      disp     192.500     -3
## 2              4              5       cyl       5.000     -3
## 3              6              7       cyl       7.000     -3
## 4              8              9      gear       3.500     -3
## 5              0              0      <NA>       0.000     -1
## 6              0              0      <NA>       0.000     -1
```

PART 05: MODELLING

↓

CHAPTER 23: LEARNING MACHINES

↓

SECTION 3:

Artificial Neural Networks (ANNs)

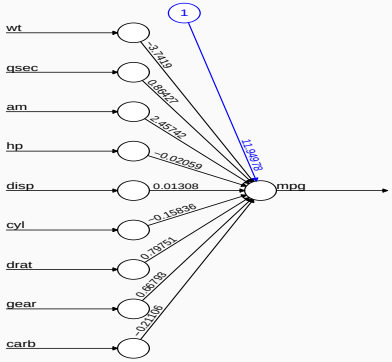


Figure 21: A logistic regression is actually a neural network with one neuron. Each variable contributes to a sigmoid function in one node, and if that one node gets loadings over a critical threshold, then we predict 1, otherwise 0. The intercept is the "1" in a circle. The numbers on the arrows are the loadings for each variable.

```r
#install.packages("neuralnet") # Do only once.

# Load the library neuralnet:
library(neuralnet)

# Fit the aNN with 2 hidden layers that have resp. 3 and 2 neurons:
# (neuralnet does not accept a formula wit a dot as in 'y ~ .' )
nn1 <- neuralnet(mpg ~ wt + qsec + am + hp + disp + cyl + drat +
                       gear + carb,
                 data = mtcars, hidden = c(3,2),
                 linear.output = TRUE)
```

```r
plot(nn1, rep = "best", information = FALSE);
```

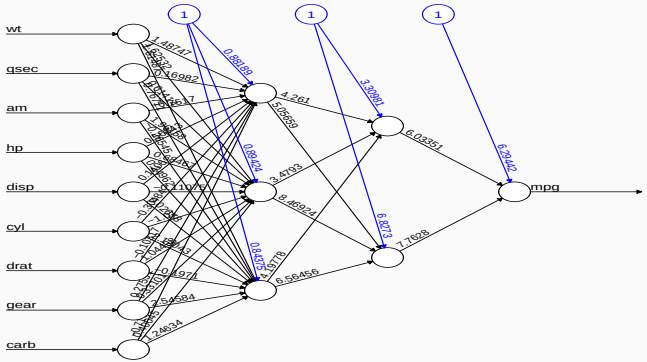


Figure 22: A simple neural net fitted to the dataset of mtcars, predicting the miles per gallon (mpg). In this example we predict the fuel consumption of a car based on some other values in the dataset t mtcars.

Using a different dataset

```
# Get the data about crimes in Boston:
library(MASS)
d <- Boston
```

Step 1: Missing Data

```
# Inspect if there is missing data:
apply(d, 2, function(x) sum(is.na(x)))
##    crim      zn   indus    chas     nox      rm     age     dis
##       0       0       0       0       0       0       0       0
##     rad     tax ptratio   black   lstat    medv
##       0       0       0       0       0       0

# There are no missing values.
```

Step 2: Split the Data in Test and Training Set

```r
set.seed(1877) # set the seed for the random generator
idx.train <- sample(1:nrow(d), round(0.75 * nrow(d)))
d.train   <- d[idx.train,]
d.test    <- d[-idx.train,]
```

Step 3: Fit a Challenger Model

```r
# Fit the linear model, no default for family, so use 'gaussian':
lm.fit <- glm(medv ~ ., data = d.train)
summary(lm.fit)
##
## Call:
## glm(formula = medv ~ ., data = d.train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -14.2361   -2.7610   -0.5274    1.7500   24.3261
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  43.951765   6.183072   7.108 6.19e-12 ***
## crim         -0.115996   0.044113  -2.630  0.00891 **
## zn            0.049986   0.015809   3.162  0.00170 **
## indus        -0.017726   0.073447  -0.241  0.80942
## chas          2.022221   1.054440   1.918  0.05591 .
## nox         -19.073462   4.377995  -4.357 1.72e-05 ***
## rm            3.259283   0.496699   6.562 1.82e-10 ***
## age           0.010649   0.015858   0.671  0.50234
## dis          -1.688850   0.240451  -7.024 1.06e-11 ***
## rad           0.335786   0.080535   4.169 3.82e-05 ***
## tax          -0.012459   0.004593  -2.713  0.00699 **
## ptratio      -1.056385   0.151795  -6.959 1.59e-11 ***
## black         0.008201   0.003229   2.539  0.01151 *
## lstat        -0.573025   0.060766  -9.430  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Step 4: Rescale the Data and Split into Training and Testing Set

```r
# Store the maxima and minima:
d.maxs <- apply(d, 2, max)
d.mins <- apply(d, 2, min)

# Rescale the data:
d.sc <- as.data.frame(scale(d, center = d.mins,
                               scale  = d.maxs - d.mins))

# Split the data in training and testing set:
d.train.sc <- d.sc[idx.train,]
d.test.sc  <- d.sc[-idx.train,]
```

Step 5: Train the ANN on the Training Set

Finally, we are ready to train the ANN. This is straightforward:

```
library(neuralnet)

# Since the shorthand notation y~. does not work in the
# neuralnet() function we have to replicate it:
nm  <- names(d.train.sc)
frm <- as.formula(paste("medv ~", paste(nm[!nm %in% "medv"], collapse = " + ")))

nn2 <- neuralnet(frm, data = d.train.sc, hidden = c(7,5,5),
                 linear.output = T)
```

```
plot(nn2, rep = "best", information = FALSE,
     show.weights = FALSE)
```

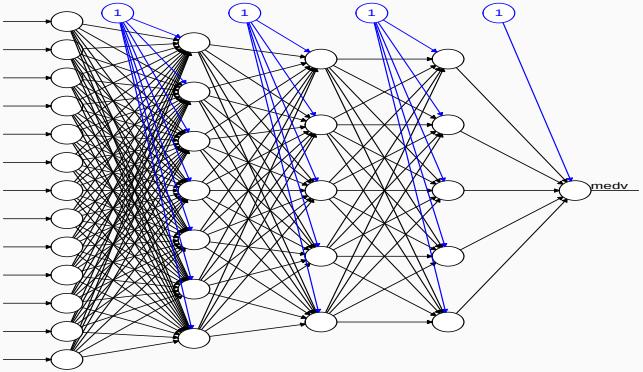


Figure 23: A visualisation of the ANN. Note that we left out the weights, because there would be too many. With 13 variables, and three layers of respectively 7, 5, and 5 neurons, we have $13 \times 7 + 7 \times 5 + 5 \times 5 + 5 + 7 + 5 + 5 + 1 = 174$ parameters.

Step 6: Test the Model on the Test Data

```
# Our independent variable 'medv' is the 14th column, so:
pr.nn2 <- compute(nn2,d.test.sc[,1:13])

# Rescale back to original span:
pr.nn2 <- pr.nn2$net.result*(max(d$medv)-min(d$medv))+min(d$medv)
test.r <- (d.test.sc$medv)*(max(d$medv)-min(d$medv))+min(d$medv)

# Calculate the MSE:
MSE.nn2 <- sum((test.r - pr.nn2)^2)/nrow(d.test.sc)
print(paste(MSE.lm,MSE.nn2))
## [1] "21.7744962283853 10.641222207598"
```

```
plot  (d.test$medv,pr.nn2,col='red',
       main='Observed vs predicted NN',
       pch=18,cex=0.7)
points(d.test$medv,pr.lm,col='blue',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend=c('NN','LM'),pch=18,
       col=c('red','blue'))
```

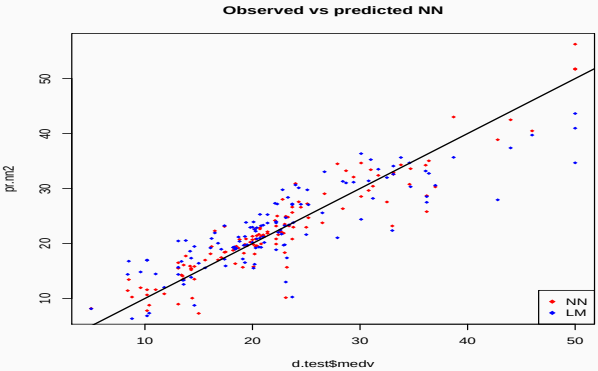


Figure 24: A visualisation of the performance of the ANN compared to the linear regression model with both models in one plot.

To execute the *k*-fold cross validation for the linear model, we use the function `cv.glm()` from the package `boot`. Below is the code for the 10 fold cross validation MSE for the linear model:

```
library(boot)
set.seed(1875)
lm.fit <- glm(medv ~ ., data = d)

# The estimate of prediction error is now here:
cv.glm(d, lm.fit, K = 10)$delta[1]
## [1] 23.78659
```

Cross Validation of the ANN

```
# Reminders:
d   <- Boston
nm  <- names(d)
frm <- as.formula(paste("medv ~", paste(nm[!nm %in% "medv"],
                        collapse = " + ")))
# Store the maxima and minima:
d.maxs <- apply(d, 2, max)
d.mins <- apply(d, 2, min)

# Rescale the data:
d.sc <- as.data.frame(scale(d, center = d.mins,
                               scale  = d.maxs - d.mins))

# Set parameters:
set.seed(1873)
cv.error <- NULL  # Initiate to append later
k        <- 10    # The number of repetitions

# This code might be slow, so you can add a progress bar as follows:
#library(plyr)
#pbar <- create_progress_bar('text')
#pbar$init(k)

# In k-fold cross validation, we must take care to select each
# observation just once in the testing set. This is made easy
# with modelr:
library(modelr)
kFoldXval <- crossv_kfold(data = d.sc, k = 10, id = '.id')
```

PART 05: MODELLING

↓

CHAPTER 23: LEARNING MACHINES

↓

SECTION 4:

Support Vector Machine

The idea behind support vector machines (SVM) is to find a hyperplane that best separates the data in the known classes. The idea is to find a hyperplane that maximises the distance between the groups.

The problem is in essence a linear set of equations to be solved, and it will fit a hyperplane, which would be a straight line for two dimensional data.

Obviously, if the separation is not linear, this method will not work well. The solution to this issue is known as the "kernel trick." We add a variable that is a suitable combination of he two variables (for example if one group appears to be centred in the 2D plane, then we could us $z = x^2 + y^2$ as third variable). Then we solve the SVM method as before (but with three variables instead of two), and find a hyperplane (flat surface) in a 3D space span by (x, y, z). This will allow for a much better separation of the data in many cases.

Function use for svm()

```
svm(formula, data, subset, na.action = na.omit, scale = TRUE,
    type = NULL, kernel = 'radial', degree = 3,
    gamma = if (is.vector(x)) 1 else 1 / ncol(x), coef0 = 0,
    cost = 1, nu = 0.5,  class.weights = NULL, cachesize = 40,
    tolerance = 0.001, epsilon = 0.1, shrinking = TRUE,
    cross = 0, probability = FALSE, fitted = TRUE, ...)
```

Most parameters work very similar to other models such as `lm`, `glm`, etc. For example `data` and `formula` do not need much explanation anymore. The variable `type`, however, is an interesting one and it is quite specific for the SVM model:

1. `C-classification`: The default type of the dependent variable is a factor object;
2. `nu-classification`: Alternative classification – the parameter ν is used to determine the number of support vectors that should be kept in the solution (relative to the size of the dataset), this method will use the parameter ϵ for the optimization, but it is automatically set;
3. `one-classification`: Allows to detect outliers and can be used when only one class is available (say only cars with four cylinders and it allows to detect "unusual cars with four cylinders");
4. `eps-regression`: The default regression type, ϵ regression allows to set the parameter ϵ, the amount of error the model can have so that anything larger than ϵ is penalized in proportion to C, the regularization parameter;

Another important parameter is `kernel`. This parameter allows us to select which kernel should be used. The following options are possible:

1. Linear: `t(u)*v`
2. Polynomial: `(gamma*t(u)*v + coef0)^degree`
3. Radial basis: `exp(-gamma*|u-v|^2)`
4. Sigmoid: `tanh(gamma*u'*v + coef0)`

When used, the parameters `gamma`, `coef0`, and `degree` can be provided to the function if one wants to over-ride the defaults.

Note – Optimisation types

Excluding the `one-classification`, there are two types of optimization: ν and ϵ and there are two types of target variables and hence we have regression and classification. In the `svm()` function both `C` and `eps` are used to refer to the same mechanism.

An Example of Regression SVM in R with `e1071`

Here is a simple example, based on the dataset `mtcars`:

```
library(e1071)
svmCars1 <- svm(cyl ~ ., data = mtcars)
summary(svmCars1)
##
## Call:
## svm(formula = cyl ~ ., data = mtcars)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.1
##     epsilon:  0.1
##
##
## Number of Support Vectors:  17
```

An Example of Classification SVM in R with e1071

Below we illustrate how classification SVM model can be fitted:

```
# split mtcars in two subsets (not necessary but easier later):
x <- subset(mtcars, select = -cyl)
y <- mtcars$cyl

# fit the model again as a classification model:
svmCars2 <- svm(cyl ~ ., data = mtcars, type = 'C-classification')

# create predictions
pred <- predict(svmCars2, x)

# show the confusion matrix:
table(pred, y)
##     y
## pred  4  6  8
##    4 11  0  0
##    6  0  7  0
##    8  0  0 14
```

Optimising the SVM

```
svmTune <- tune(svm, train.x=x, train.y=y, kernel = "radial",
                ranges =  list(cost = 10^(-1:2), gamma = c(.5, 1, 2)))

print(svmTune)
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.906572
```

After you have found the optimal parameters, you can run the model again and specify the desired parameters and compare the performance (e.g. with the confusion matrix).

PART 05: MODELLING

↓

CHAPTER 23: LEARNING MACHINES

↓

SECTION 5:

Unsupervised Learning and Clustering

Clustering methods identify sets of similar objects – referred to as “clusters” – in a multivariate data set. The most common types of clustering include

1. partitioning methods,
2. hierarchical clustering,
3. fuzzy clustering,
4. density-based clustering, and
5. model-based clustering.

Given a set of observations $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ (where each observations x_i is a n-dimensional vector), k-means clustering aims to minimize the variance for k (where $k \leq n$) sets – or clusters, C_i henceforth – between the mean of the set and the members of that group $\mathbf{C} = \{C_1, C_2, \ldots, C_k\}$. So the goal of k-means clustering becomes to find

$$\text{argmin}_{\mathbf{C}} \sum_{i=1}^{k} \sum_{x\ in C_i} ||\mathbf{x} - \mu_\mathbf{i}||$$

The standard algorithm start from randomly taking k different observations as initial centre for the k clusters. Each observation is then assigned to the cluster whose centre is the "closest." The distance is usually expressed as the Euclidian distance between that observation and the centroid of the cluster.

Then we calculate again the centre of each cluster[4] and the process is repeated: each observation is now allocated to the cluster that has the centroid closest to the observation. This step is then repeated till there are no changes in the cluster allocations in consecutive steps.

In this section, we will use the dataset `mtcars` that is – by now – well known. The dataset is usually loaded when R starts, and if that is not the case you can find it in the package `datasets`.

First, we have a look at the data `mtcars` and choose weight and fuel consumption as variables of interest for our analysis. Along the way, we introduce you to the package `ggrepel` that is handy to pull labels away from each other. We use this because we want to plot the name of the car next to each dot in order to get some understanding of what is going on.

Most of those things can be obtained with `ggplot2` alone.[5] The output is in Figure 25 on slide 127.

```
library(ggplot2)
library(ggrepel)  # provides geom_label_repel()
ggplot(mtcars, aes(wt, mpg, color = factor(cyl))) +
       geom_point(size = 5) +
       geom_label_repel(aes(label = rownames(mtcars)),
                  box.padding   = 0.2,
                  point.padding = 0.25,
                  segment.color = 'grey60')
```

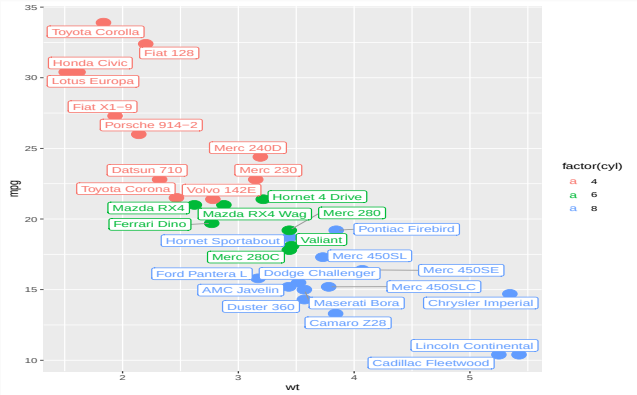
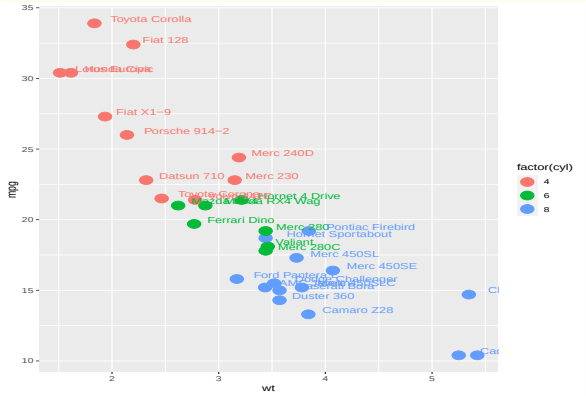


Figure 25: The cars in the dataset mtcars with fuel consumption plotted in function of weight and coloured by the number of cylinders.

Note – Elegant labels

Compare Figure 25 on slide 127 with the result that we could get from adding to our plot text via the function `geom_text()`:

```
ggplot(mtcars, aes(wt, mpg, color = factor(cyl))) +
       geom_point(size = 5) +
       geom_text(aes(label = rownames(mtcars)),
                 hjust = -0.2, vjust = -0.2)
```



It also works, but `geom_label()` and `geom_label_repel` do a lot of heavy lifting: putting a frame around the text, uncluttering the labels, and even adding a small line between the box and the dot if the distance gets too big.

Plotting the cars in the `(wt, mpg)` plane we notice a certain – almost linear – relation and colouring the dots according to the number of cylinders we might be able to imagine some possible groups.

PCA before clustering i

PCA before clustering ii

```
# Normalize the whole mtcars dataset:
d <- data.frame(matrix(NA, nrow = nrow(mtcars), ncol = 1))
d <- d[,-1]  # d is an empty data frame with 32 rows
for (k in 1:ncol(mtcars)) {
rng <- range(mtcars[, k], na.rm = TRUE)
d[, k]  <- (mtcars[, k]  - rng[1])  / rng[2]
}
colnames(d) <- colnames(mtcars)
rownames(d) <- rownames(mtcars)

# The PCA analysis:
pca1 <- prcomp(d)
summary(pca1)
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6
## Standard deviation     0.6960 0.4871 0.20255 0.13916 0.09207 0.07719
## Proportion of Variance 0.5993 0.2935 0.05076 0.02396 0.01049 0.00737
## Cumulative Proportion  0.5993 0.8929 0.94365 0.96761 0.97810 0.98547
##                            PC7     PC8     PC9    PC10    PC11
## Standard deviation     0.06203 0.05801 0.05112 0.03642 0.02432
## Proportion of Variance 0.00476 0.00416 0.00323 0.00164 0.00073
## Cumulative Proportion  0.99023 0.99439 0.99763 0.99927 1.00000

# Note also:
class(pca1)
## [1] "prcomp"
```

We see that the first two components explain about 90% of the variance. This means that for most applications

Fuzzy Clustering: the Idea

One such algorithm is called “fuzzy clustering” – also referred to as soft clustering or “soft k-means.” It works as follows:

1. Decide on the number of clusters, k.
2. Each observation has a coefficient w_{ij} (the degree of x_i being in a cluster j) for each cluster — in the first step assign those coefficients randomly.
3. Calculate the centroid of each cluster as

$$c_j = \frac{\sum_i w_{ij}(x_i)^m x_i}{\sum_i w_{ij}(x_i)^m}$$

   where m is the parameter that controls how fuzzy the cluster will be. Higher m values will result in a more fuzzy cluster. This parameter is also referred to as the “hyper-parameter”
4. For each observation calculate again the weights with the updated centroids.

$$w_{ij} = \frac{1}{\sum_l^k \left(\frac{||x_i - c_i||}{x_i - c_l} \right)^{\frac{2}{m-1}}}$$

5. Repeat from step 3, until the algorithm has coefficients that do not change more than a given small value ϵ, the sensitivity threshold

```r
library(tidyverse)  # provides if_else
library(ggplot2)    # 2D plotting
library(ggfortify)
library(cluster)    # provides fanny (the fuzzy clustering)
library(ggrepel)    # provides geom_label_repel (de-clutter labels)


carCluster <- fanny(d, 4)
my_colors <- if_else(carCluster$cluster == 1, "coral",
               if_else(carCluster$cluster == 2, "darkolivegreen3",
               if_else(carCluster$cluster == 3, "cyan3",
                 "darkorchid1")))

# Autoplot with visualization of 4 clusters
autoplot(carCluster, label=FALSE, frame=TRUE,  frame.type='norm',
         shape=16,
         loadings=TRUE,  loadings.colour = 'blue',
         loadings.label = TRUE, loadings.label.size = 5,
         loadings.label.vjust = 1.2, loadings.label.hjust = 1.3) +
       geom_point(size = 5, alpha = 0.7, colour = my_colors) +
       geom_label_repel(aes(label = rownames(mtcars)),
                  box.padding   = 0.2,
                  point.padding = 0.25,
                  segment.color = 'grey40') +
                  theme_classic()
```

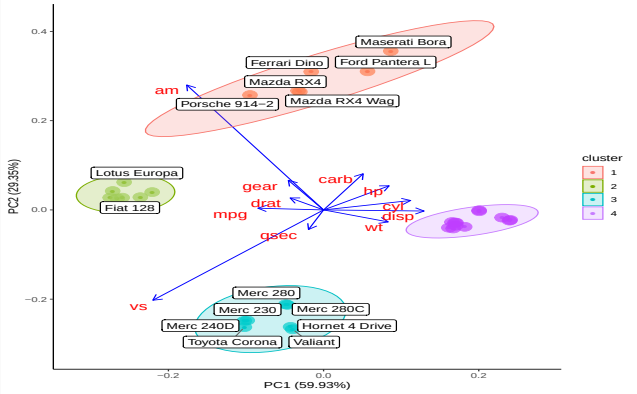


Figure 28: A plot with `autoplot()`, enhanced with `ggrepel` of the fuzzy clustering for the dataset mtcars.

Hierarchical Clustering: the Idea

Hierarchical clustering is a particularly useful approach that provides a lot of insight and does not require to define a number of clusters to be provided by the user. Ultimately, we get a tree-based representation of all observations in our dataset, which is also known as the dendrogram. This means that we can use the dendrogram itself to make an educated guess on where to separate the dendrogram and hence how much and at what level we make clusters.

The R code to compute and visualize hierarchical clustering is below, and the plot resulting from it is in Figure 29 on slide 138:

```
# Compute hierarchical clustering
library(tidyverse)
cars_hc <- mtcars                       %>%
           scale                        %>% # scale the data
           dist(method = "euclidean")   %>% # dissimilarity matrix
           hclust(method = "ward.D2")       # hierachical clustering

plot(cars_hc)
```

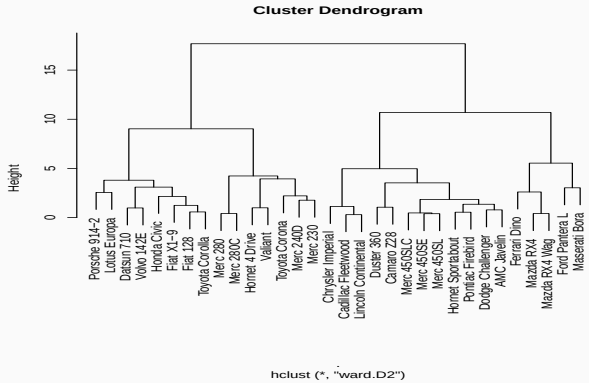


Figure 29: A hierarchical cluster for the dataset mtcars.

part 05: Modelling

↓

chapter 24:

Towards a Tidy Modelling Cycle with modelr

Loading the Packages and Show the Example

The package `model` provides a layer around R's base functions that allows not only to work with models using the pipe `%>%` command, but also provides some functions that are more intuitive to work with. `modelr` is not part of the core-tidyverse, so, we need to load it separately.

```
library(tidyverse)
library(modelr)
```

```
d   <- mtcars
lm1 <- lm(mpg ~ wt + cyl, data = d)
```

PART 05: MODELLING

↓

CHAPTER 24: TOWARDS A TIDY MODELLING CYCLE WITH MODELR

↓

SECTION 1:

Adding predictions

Function use for add_predictions()

```
add_predictions(data, model, var = "pred", type = NULL)
```

Adds predictions to a dataset for a given model, the predictions are added in a column named by the variable `pred`.

```
library(modelr)

# Use the data defined above:
d1 <- d %>% add_predictions(lm1)

# d1 has now an extra column "pred"
head(d1)
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
##                       pred
## Mazda RX4         22.27914
## Mazda RX4 Wag     21.46545
## Datsun 710        26.25203
```

PART 05: MODELLING

↓

CHAPTER 24: TOWARDS A TIDY MODELLING CYCLE WITH MODELR

↓

SECTION 2:

Adding Residuals

Adding residuals

Function use for add_residuals()

```
add_residuals(data, model, var = "resid")
```

Adds residuals to a given dataset for a given model. The new column is named by the parameter `var`.

```
d2 <- d1 %>% add_residuals(lm1)

# d2 has now an extra column "resid"
head(d2)
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
##                       pred      resid
## Mazda RX4         22.27914 -1.2791447
## Mazda RX4 Wag     21.46545 -0.4654468
## Datsun 710        26.25203 -3.4520262
## Hornet 4 Drive    20.38052  1.0194838
## Hornet Sportabout 16.64696  2.0530424
## Valiant           19.59873 -1.4987281
```

PART 05: MODELLING

↓

CHAPTER 24: TOWARDS A TIDY MODELLING CYCLE WITH MODELR

↓

SECTION 3:

Bootstrapping Data

Function use for bootstrap()

```
bootstrap(data, n, id = ".id")
```

Generates *n* bootstrap replicates (dataset build from random draws – with replacement – of observations from the source data) of the dataset data.

The following code illustrates how bootstrapping can be used to generate a set of estimates for relevant coefficients for a linear model, and then tidies up the results for further use.

```r
set.seed(1872)   # make sure that results can be replicated
library(modelr)  # provides bootstrap
library(purrr)   # provides map, map_df, etc.
library(ggplot2) # provides ggplot
d    <- mtcars
boot <- bootstrap(d, 10)

# Now, we can leverage tidyverse functions such as map to create
# multiple models on the 10 datasets
models <- map(boot$strap, ~ lm(mpg ~ wt + cyl, data = .))

# The function tidy of broom (also tidyverse) allows to create a
# dataset based on the list of models. Broom is not loaded, because
# it also provides a function bootstrap().
tidied <- map_df(models, broom::tidy, .id = "id")
```

Now that we have a tidy tibble of results, we can for example visualise the results in order to study how stable the model is. The histogram of the estimates of coefficients is shown in Figure 30 on slide 149 with the following code.

Bootstrapping data iii

```
# Visualize the results with ggplot2:
p <- ggplot(tidied, aes(estimate)) +
     geom_histogram(bins = 5, col = 'red', fill='khaki3',
                    alpha = 0.5) +
     ylab('Count') +
     xlab('Estimate of the coefficient in the plot-title') +
     facet_grid(. ~ term, scales = "free")
p
```

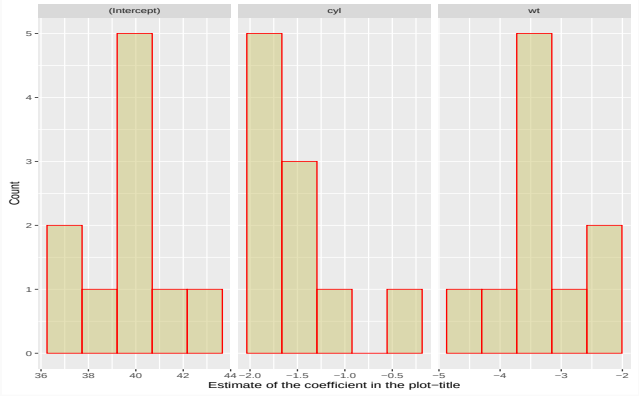


Figure 30: The results of the bootstrap exercise: a set of estimates for each coefficient.

PART 05: MODELLING

↓

CHAPTER 24: TOWARDS A TIDY MODELLING CYCLE WITH MODELR

↓

SECTION 4:

Other Functions of modelr

part 05: Modelling

↓

chapter 25:

Model Validation

PART 05: MODELLING

↓

CHAPTER 25: MODEL VALIDATION

↓

SECTION 1:

Model Quality Measures

Model Quality Measures with `modelr`

```r
# load modelr:
library(modelr)

# Fit a model:
lm1 <- lm(mpg ~ wt + qsec + am, data = mtcars)

# MSE (mean square error):
mse(lm1, mtcars)
## [1] 5.290185

# RMSE (root mean square error):
rmse(lm1, mtcars)
## [1] 2.30004

# MAD (mean absolute error):
mae(lm1, mtcars)
## [1] 1.931954

# Quantiles of absolute error:
qae(lm1, mtcars)
##         5%       25%       50%       75%       95%
## 0.3794271 0.9657082 1.4923568 2.8170045 4.3435305

# R-square (variance of predictions divided by the variance of the
# response variable):
rsquare(lm1, mtcars)
## [1] 0.8496636
```

Cross Validations with `modelr`

```r
set.seed(1871)

# Split the data:
rs  <- mtcars  %>%
       resample_partition(c(train = 0.6, test = 0.4))

# Train the model on the training set:
lm2 <- lm(mpg ~ wt + qsec + am, data = rs$train)

# Compare the RMSE on the training set with the testing set:
rmse(lm2, rs$train); rmse(lm2, rs$test)
## [1] 2.354864
## [1] 2.559619

# Note that this can alos be done with the pipe operator:
lm2 %>% rmse(rs$train)
## [1] 2.354864

lm2 %>% rmse(rs$test)
## [1] 2.559619
```

PART 05: MODELLING

↓

CHAPTER 25: MODEL VALIDATION

↓

SECTION 2:

Predictions and Residuals

Add Predictions and Residuals with `modelr`

```r
# Fit the model:
lm1 <- lm(mpg ~ wt + qsec + am, data = mtcars)

# Add the predictions and residuals:
df <- mtcars               %>%
      add_predictions(lm1) %>%
      add_residuals(lm1)

# The predictions are now available in $pred:
head(df$pred)
## [1] 22.47046 22.15825 26.28107 20.85744 17.00959 20.85409

# The residuals are now available in $resid:
head(df$resid)
## [1] -1.4704610 -1.1582487 -3.4810670  0.5425557  1.6904131 -2.7540920

# It is now easy to do something with those predictions and
# residuals, e.g. the following 3 lines all do the same:
sum((df$pred - df$mpg)^2) / nrow(mtcars)
## [1] 5.290185

sum((df$resid)^2) / nrow(mtcars)
## [1] 5.290185

mse(lm1, mtcars)  # Check if this yields the same
## [1] 5.290185
```

PART 05: MODELLING

↓

CHAPTER 25: MODEL VALIDATION

↓

SECTION 3:

Bootstrapping

The function “sample()” takes a sample from data

Function use for sample()

`sample(x, size, replace = FALSE, prob = NULL)` with

- `x:` either a vector of one or more elements from which to choose, or a positive integer.
- `size:` the number of items to select from x
- `replace:` set to TRUE if sampling is to be done with replacement
- `prob:` a vector of probability weights for obtaining the elements of the vector being sampled

Example: Sampling the SP500 data i

```r
# Create the sample:
SP500_sample <- sample(SP500,size=100)

# Change plotting to 4 plots in one output:
par(mfrow=c(2,2))

# The histogram of the complete dataset:
hist(SP500,main="(a) Histogram of all data",fr=FALSE,
     breaks=c(-9:5),ylim=c(0,0.4))

# The histogram of the sample:
hist(SP500_sample,main="(b) Histogram of the sample",
     fr=FALSE,breaks=c(-9:5),ylim=c(0,0.4))

# The boxplot of the complete dataset:
boxplot(SP500,main="(c) Boxplot of all data",ylim=c(-9,5))

# The boxplot of the complete sample:
boxplot(SP500_sample,main="(c) Boxplot of the sample",
        ylim=c(-9,5))
```

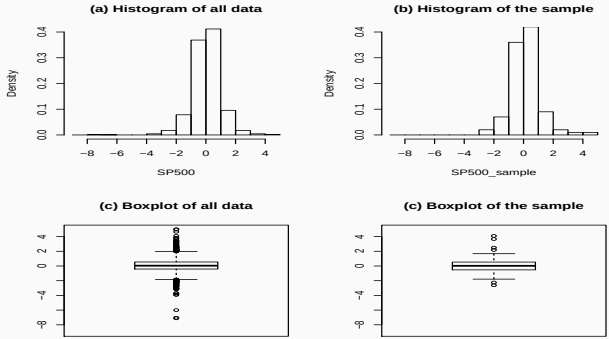


Figure 31: Bootstrapping the returns of the S&P500 index.

Example: Sampling the SP500 data iii

```
# Reset the plot parameters:
par(mfrow=c(1,1))
```

In base R, the sample is a dataset itself and it can be addressed as any other dataset:

```
mean(SP500)
## [1] 0.04575267

mean(SP500_sample)
## [1] 0.07596865

sd(SP500)
## [1] 0.9477464

sd(SP500_sample)
## [1] 0.9967802
```

The function `bootstrap()` works as follows:

```
# Bootstrap generates a number of re-ordered datasets
boot <- bootstrap(mtcars, 3)
# The datasets are now in boot$strap[[n]]
# with n between 1 and 3

# e.g. the 3rd set is addressed as follows:
class(boot$strap[[3]])
## [1] "resample"

nrow(boot$strap[[3]])
## [1] 32

mean(as.data.frame(boot$strap[[3]])$mpg)
## [1] 18.94687

# It is also possible to coerce the selections into a data-frame:
df <- as.data.frame(boot$strap[[3]])
class(df)
## [1] "data.frame"
```

Bootstrapping with `modelr`: an Example i

```r
set.seed(1871)
library(purrr)  # to use the function map()
boot <- bootstrap(mtcars, 150)

lmodels <- map(boot$strap, ~ lm(mpg ~ wt + hp + am:vs, data = .))

# The function tidy of broom turns a model object in a tibble:
df_mods <- map_df(lmodels, broom::tidy, .id = "id")

# Create the plots of histograms of estimates for the coefficients:
par(mfrow=c(2,2))
hist(subset(df_mods, term == "wt")$estimate, col="khaki3",
     main = '(a) wt', xlab = 'estimate for wt')
hist(subset(df_mods, term == "hp")$estimate, col="khaki3",
     main = '(b) hp', xlab = 'estimate for hp')
hist(subset(df_mods, term == "am:vs")$estimate, col="khaki3",
     main = '(c) am:vs', xlab = 'estimate for am:vs')
hist(subset(df_mods, term == "(Intercept)")$estimate, col="khaki3",
     main = '(d) intercept', xlab = 'estimate for the intercept')
```

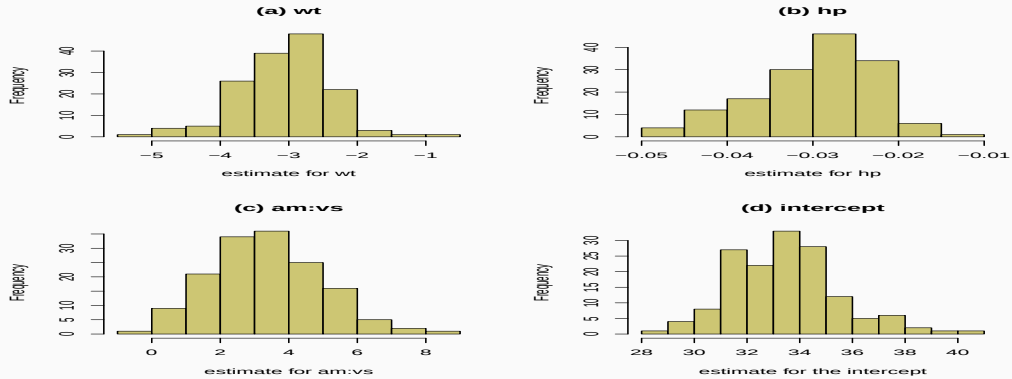


Figure 32: The histograms of the different coefficients of the linear regression model predicting the mpg in the dataset mtcars. We show (a) Estimate for wt., (b) Estimate for hp., (c) Estimate for am:vs., and (d) Estimate for the intercept.

```
par(mfrow=c(1,1))
```

PART 05: MODELLING

↓

CHAPTER 25: MODEL VALIDATION

↓

SECTION 4:

Cross-Validation

Elementary Cross Validation in Base R

```
d <- mtcars                # get data
set.seed(1871)             # set the seed for the random generator
idx.train <- sample(1:nrow(d),round(0.75*nrow(d)))
d.train <- d[idx.train,]   # positive matches for training set
d.test  <- d[-idx.train,]  # the opposite to the testing set
```

```
set.seed(1870)
sample_cars <- mtcars %>%
               resample(sample(1:nrow(mtcars),5)) # random 5 cars

# This is a resample object (indexes shown, not data):
sample_cars
## <resample [5 x 11]> 14, 25, 32, 16, 20


# Turn it into data:
as.data.frame(sample_cars)
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4
##                     carb
## Merc 450SLC            3
## Pontiac Firebird       2
## Volvo 142E             2
## Lincoln Continental    4
## Toyota Corolla         1


# or into a tibble
as_tibble(sample_cars)
## # A tibble: 5 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
```

```
library(modelr)
rs <- mtcars  %>%
      resample_partition(c(train = 0.6, test = 0.4))

# address the datasets with: as.data.frame(rs$train)
#                            as.data.frame(rs$test)

# Check execution:
lapply(rs, nrow)
## $train
## [1] 19
##
## $test
## [1] 13
```

Now, that we have a training and test dataset, we have all the tools necessary. The standard workflow now becomes simply the following:

```
# 0. Store training and test dataset for further use (optional):
d_train  <- as.data.frame(rs$train)
d_test   <- as.data.frame(rs$test)

# 1. Fit the model on the training dataset:
lm1      <- lm(mpg ~ wt + hp + am:vs, data = rs$train)

# 2. Calculate the desired performance measure (e.g.
# root mean square error (rmse)):
rmse_trn <- lm1 %>% rmse(rs$train)
rmse_tst <- lm1 %>% rmse(rs$test)
print(rmse_trn)
## [1] 2.014614

print(rmse_tst)
## [1] 2.990294
```

We were using a performance measure that was readily available via the function `rmse()`, but if we want to calculate another risk measure, we might need the residuals and/or predictions first. Below, we calculate the same risk measure without using the function `rmse()`. Note that step one is the same as in the aforementioned code.

```r
# 2. Add predictions and residuals:
x_trn  <- add_predictions(d_train, model = lm1) %>%
          add_residuals(model = lm1)
x_tst  <- add_predictions(d_test,  model = lm1) %>%
          add_residuals(model = lm1)


# 3. Calculate the desired risk metrics (via the residuals):
RMSE_trn  <- sqrt(sum(x_trn$resid^2) / nrow(d_train))
RMSE_tst  <- sqrt(sum(x_tst$resid^2) / nrow(d_test))
print(RMSE_trn)
## [1] 2.014614

print(RMSE_tst)
## [1] 2.990294
```

```r
# Monte Carlo cross validation
cv_mc <- crossv_mc(data = mtcars, # the dataset to split
           n = 50,      # n random partitions train and test
           test = 0.25, # validation set is 25%
           id = ".id")  # unique identifier for each model

# Example of use:

# Access the 2nd test dataset:
d <- data.frame(cv_mc$test[2])

# Access mpg in that data frame:
data.frame(cv_mc$test[2])$mpg
## [1] 16.4 10.4 30.4 19.2 27.3 26.0 15.8 19.7 15.0

# More cryptic notations are possible to obtain the same:
mtcars[cv_mc[[2]][[2]][2]$idx,1]
## [1] 16.4 10.4 30.4 19.2 27.3 26.0 15.8 19.7 15.0
```

Monte Carlo Cross Validation in the tidyverse: An Example i

```
set.seed(1868)
library(modelr)     # sample functions
library(purrr)      # to use the function map()

cv_mc <- crossv_mc(mtcars, n = 50, test = 0.40)
mods  <- map(cv_mc$train, ~ lm(mpg ~ wt + hp + am:vs, data = .))
RMSE  <- map2_dbl(mods, cv_mc$test, rmse)
hist(RMSE, col="khaki3")
```

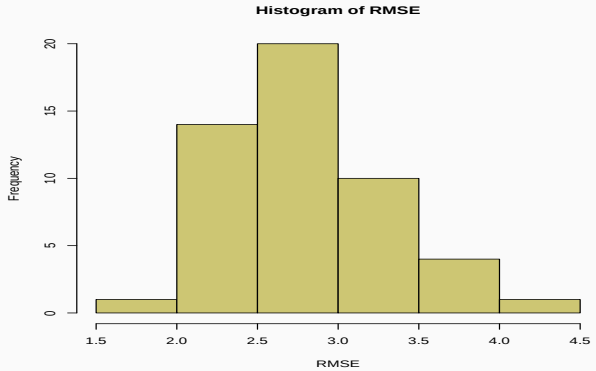


Figure 33: The histogram of the RMSE for a Monte Carlo cross validation on the dataset mtcars.

The function `crossv_kfold` of `modelr` will prepare the selections as for each run as follows.

```
library(modelr)
# k-fold cross validation
cv_k  <- crossv_kfold(data = mtcars,
           k = 5,      # number of folds
           id = ".id") # unique identifier for each
```

Each observation of the 32 will now appear once in one test dataset:

```
cv_k$test
## $`1`
## <resample [7 x 11]> 1, 6, 14, 15, 23, 26, 32
##
## $`2`
## <resample [7 x 11]> 3, 10, 12, 17, 18, 22, 28
##
## $`3`
## <resample [6 x 11]> 2, 4, 9, 19, 20, 25
##
## $`4`
## <resample [6 x 11]> 5, 8, 13, 24, 27, 29
##
## $`5`
## <resample [6 x 11]> 7, 11, 16, 21, 30, 31
```

```
set.seed(1868)
library(modelr)
library(magrittr)  # to access the %T>% pipe
crossv <- mtcars                                          %>%
          crossv_kfold(k = 5)
RMSE <- crossv                                            %$%
        map(train, ~ lm(mpg ~ wt + hp + am:vs, data = .)) %>%
        map2_dbl(crossv$test, rmse)                       %T>%
        hist(col = "khaki3", main ="Histogram of RMSE",
             xlab = "RMSE")
```

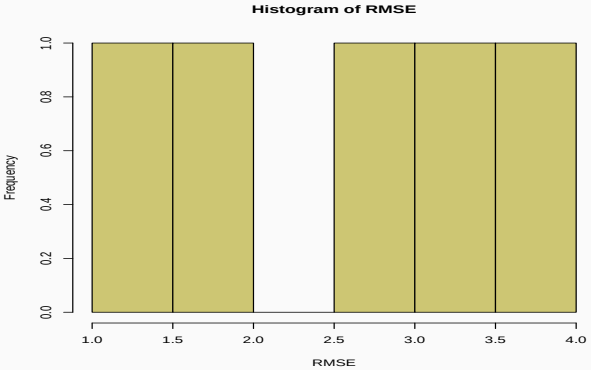


Figure 34: Histogram of the RMSE based on a 5-fold cross validation. The histogram indeed shows that there were 5 observations. Note the significant spread of RMSE: the largest one is about four times the smallest.

PART 05: MODELLING

↓

CHAPTER 25: MODEL VALIDATION

↓

SECTION 5:

Validation in a Broader Perspective

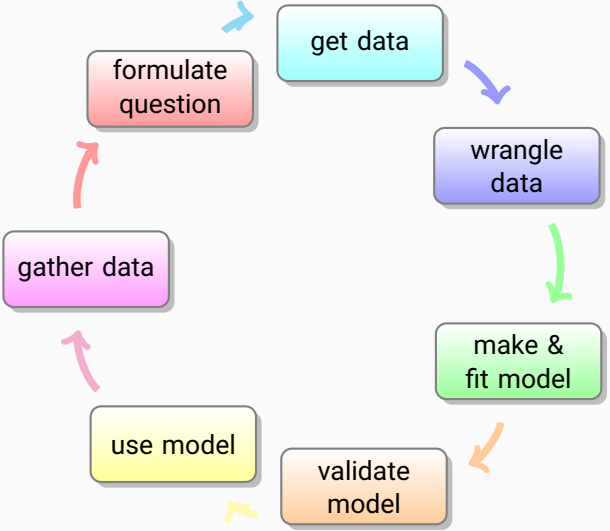


Figure 35: The life cycle of a model: a model is an integrated part of business and focus of continuous improvement. Note how using a model will collect more data and lead to improvement of the model itself.

part 05: Modelling

↓

chapter 26:

Labs

PART 05: MODELLING

↓

CHAPTER 26: LABS

↓

SECTION 1:

Financial Analysis with quantmod

```r
# Install quantmod:
if(!any(grepl("quantmod", installed.packages()))){
    install.packages("quantmod")}

# Load the library:
library(quantmod)
```

Now, we are ready to use `quantmod`. For example, we can start downloading some data with the function `getSymbols()`:

```r
# Download historic data of the Google share price:
getSymbols("GOOG", src = "yahoo")          # get Google's history
## [1] "GOOG"

getSymbols(c("GS", "GOOG"), src = "yahoo") # to load more than one
## [1] "GS"   "GOOG"
```

What type of data does quantmod provide?

The function `stockSymbols()` can provide a list of symbols that are quoted on Amex, Nasdaq, and NSYE.

```
stockList <- stockSymbols()  # get all symbols
nrow(stockList)     # number of symbols
## [1] 11083

colnames(stockList) # information in this list
##  [1] "Symbol"           "Name"             "LastSale"
##  [4] "MarketCap"        "IPOyear"          "Sector"
##  [7] "Industry"         "Exchange"         "Test.Issue"
## [10] "Round.Lot.Size"   "ETF"              "Market.Category"
## [13] "Financial.Status" "Next.Shares"      "ACT.Symbol"
## [16] "CQS.Symbol"       "NASDAQ.Symbol"
```

```
getSymbols("HSBC",src="yahoo") #get HSBC's data from Yahoo
## [1] "HSBC"
```

```
# 1. The bar chart:
barChart(HSBC)
```

Figure 36: Demonstration of the barChart() function of the package quantmod.

```
# 2. The line chart:
lineChart(HSBC)
```

Figure 37: Demonstration of the lineChart() function of the package quandmod.

```
# Note: the lineChart is also the default that yields the same as plot(HSBC)

# 3. The candle chart:
candleChart(HSBC, subset='last 1 years',theme="white",
            multi.col=TRUE)
```

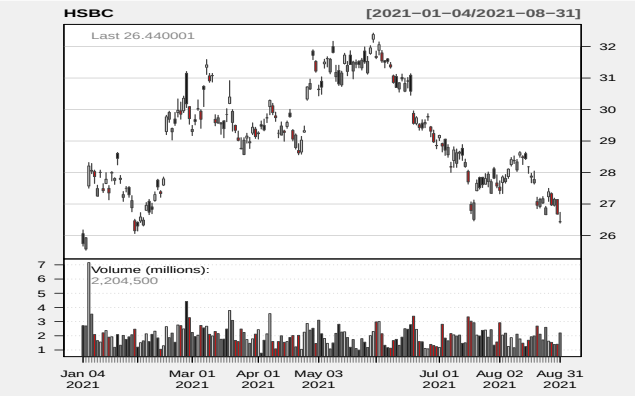


Figure 38: Demonstration of the candleChart() function of the package quantmod.

```
myxtsdata["2008-01-01/2010-12-31"]  # between 2 date-stamps

# All data before or after a certain time-stamp:
xtsdata["/2007"]  # from start of data until end of 2007
xtsdata["2009/"]  # from 2009 until the end of the data

# Select the data between different hours:
xtsdata["T07:15/T09:45"]
```

Subsetting by Time and Date

```
HSBC['2017']    #returns HSBC's OHLC data for 2017
HSBC['2017-08'] #returns HSBC's OHLC data for August 2017
HSBC['2017-06::2018-01-15'] # from June 2017 to Jan 15 2018

HSBC['::']     # returns all data
HSBC['2017::'] # returns all data in HSBC, from 2017 onward
my.selection <- c('2017-01','2017-03','2017-11')
HSBC[my.selection]
```

Aggregating to a different time scale

```
periodicity(HSBC)
unclass(periodicity(HSBC))
to.weekly(HSBC)
to.monthly(HSBC)
periodicity(to.monthly(HSBC))
ndays(HSBC); nweeks(HSBC); nyears(HSBC)
```

```
endpoints(HSBC,on="years")
##  [1]    0  251  504  756 1008 1260 1510 1762 2014 2266 2518 2769 3020
## [14] 3272 3525 3692

# Find the maximum closing price each year:
apply.yearly(HSBC,FUN=function(x) {max(Cl(x)) } )
##             [,1]
## 2007-12-31 99.52
## 2008-12-31 87.67
## 2009-12-31 63.95
## 2010-12-31 59.32
## 2011-12-30 58.99
## 2012-12-31 53.07
## 2013-12-31 58.61
## 2014-12-31 55.96
## 2015-12-31 50.17
## 2016-12-30 42.96
## 2017-12-29 51.66
## 2018-12-31 55.62
## 2019-12-31 44.70
## 2020-12-31 39.37
## 2021-08-31 32.38

# The same thing - only more general:
subHSBC <- HSBC['2012::']
period.apply(subHSBC,endpoints(subHSBC,on='years'), FUN=function(x) {max(Cl(x))} )
##             [,1]
## 2012-12-31 53.07
```

```
seriesHi(HSBC)
##            HSBC.Open HSBC.High HSBC.Low HSBC.Close HSBC.Volume
## 2007-10-31     98.92     99.52    98.05      99.52     1457900
##            HSBC.Adjusted
## 2007-10-31      49.39895

has.Cl(HSBC)
## [1] TRUE

tail(Cl(HSBC))
##            HSBC.Close
## 2021-08-24      26.88
## 2021-08-25      27.39
## 2021-08-26      26.97
## 2021-08-27      27.15
## 2021-08-30      26.68
## 2021-08-31      26.44
```

There are even functions that will calculate differences, for example:

- `OpCl()`: daily percent change open to close
- `OpOp()`: daily open to open change

- `HiCl()`: the percent change from high to close

These functions rely on the following that are also available to use:

- `Lag()`: gets the previous value in the series
- `Next()`: gets the next value in the series
- `Delt()`: returns the change (delta) from two prices

```
Lag(Cl(HSBC))
Lag(Cl(HSBC), c(1, 5, 10)) # One, five and ten period lags
Next(OpCl(HSBC))

# Open to close one, two and three-day lags:
Delt(Op(HSBC),Cl(HSBC),k=1:3)
```

Period Returns i

```
dailyReturn(HSBC)
weeklyReturn(HSBC)
monthlyReturn(HSBC)
quarterlyReturn(HSBC)
yearlyReturn(HSBC)
allReturns(HSBC)     # all previous returns
```

Consider the following naive model:

```
# First, we create a quantmod object.
# At this point, we do not need to load data.

setSymbolLookup(SPY = 'yahoo', VXN = list(name = '^VIX', src = 'yahoo'))

qmModel <- specifyModel(Next(OpCl(SPY)) ~ OpCl(SPY) + Cl(VIX))
head(modelData(qmModel))
##            Next.OpCl.SPY      OpCl.SPY  Cl.VIX
## 2014-12-04  0.0006254149  0.0005782548 28447.7
## 2014-12-05 -0.0043851339  0.0006254149 26056.5
## 2014-12-08  0.0102755104 -0.0043851339 23582.8
## 2014-12-09 -0.0133553492  0.0102755104 21274.0
## 2014-12-10  0.0015204875 -0.0133553492 19295.0
## 2014-12-11 -0.0086360048  0.0015204875 17728.3
```

First, we import the data and plot the linechart for the symbol in Figure 39 on slide 197:

```
getSymbols('HSBC',src='yahoo') #google doesn't carry the adjusted price
## [1] "HSBC"

lineChart(HSBC)
```

Figure 39: The evolution of the HSBC share for the last ten years.

The line-chart shows that the behaviour of the stock is very different in the period after the crisis. Therefore, we decide to consider only data after 2010.

```
HSBC.tmp   <- HSBC["2010/"]          #see: subsetting for xts objects
```

The next step is to divide our data in a training dataset and a test-dataset.

```
# use 70% of the data to train the model:
n          <- floor(nrow(HSBC.tmp) * 0.7)
HSBC.train <- HSBC.tmp[1:n]               # training data
HSBC.test  <- HSBC[(n+1):nrow(HSBC.tmp)]  # test-data
# head(HSBC.train)
```

Till now we used the functionality of `quantmod` to pull in data, but the function `specifyModel()` allows us to prepare automatically the data for modelling: it will align the next opening price with the explaining variables. Further, `modelData()` allows to make sure the data is up-to-date.

```
# Making sure that whenever we re-run this the latest data is pulled in:
m.qm.tr <- specifyModel(Next(Op(HSBC.train)) ~ Ad(HSBC.train)
          + Hi(HSBC.train) - Lo(HSBC.train) + Vo(HSBC.train))

D <- modelData(m.qm.tr)
```

We decide to create an additional variable that is the difference between the high and low prices of the previous day.

```
# Add the additional column:
D$diff.HSBC <- D$Hi.HSBC.train - D$Lo.HSBC.train

# Note that the last value is NA:
tail(D, n = 3L)
##            Next.Op.HSBC.train Ad.HSBC.train Hi.HSBC.train
## 2018-02-28              49.93      42.10609         50.39
## 2018-03-01              49.14      42.08063         50.00
## 2018-03-02                 NA      41.97883         49.52
##            Lo.HSBC.train Vo.HSBC.train diff.HSBC
## 2018-02-28         49.60       1902700  0.790001
## 2018-03-01         49.27       2673600  0.730000
## 2018-03-02         48.93       2283700  0.590000

# Since the last value is NA, let us remove it:
D <- D[-nrow(D),]
```

The column names of the data inherit the full name of the dataset. This is not practical since the names will be different in the training set and in the test-set. So we rename them before making the model.

```
colnames(D) <- c("Next.Op","Ad","Hi","Lo","Vo","Diff")
```

Now, we can create the model.

```
m1 <- lm(D$Next.Op ~ D$Ad + D$Diff + D$Vo)
summary(m1)
##
## Call:
## lm(formula = D$Next.Op ~ D$Ad + D$Diff + D$Vo)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -23.5149  -4.4501   0.5144   3.5156  13.5259
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.541e+01  8.845e-01   17.43   <2e-16 ***
## D$Ad         9.391e-01  2.410e-02   38.98   <2e-16 ***
## D$Diff       8.538e+00  4.102e-01   20.82   <2e-16 ***
## D$Vo        -1.183e-06  1.045e-07  -11.32   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.721 on 2050 degrees of freedom
## Multiple R-squared:  0.5246,Adjusted R-squared:  0.5239
## F-statistic:   754 on 3 and 2050 DF,  p-value: < 2.2e-16
```

The volume of trading in the stock does not seem to play a significant role, so we leave it out.

```
m2 <- lm(D$Next.Op ~ D$Ad + D$Diff)
summary(m2)
##
## Call:
## lm(formula = D$Next.Op ~ D$Ad + D$Diff)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -23.6521  -4.9734   0.9299   3.8559  11.0599
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.95237    0.81603   13.42   <2e-16 ***
## D$Ad         1.03550    0.02323   44.57   <2e-16 ***
## D$Diff       6.46346    0.37820   17.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.865 on 2051 degrees of freedom
## Multiple R-squared:  0.4948,Adjusted R-squared:  0.4944
## F-statistic:  1005 on 2 and 2051 DF,  p-value: < 2.2e-16
```

From the output of the command `summary(m2)` we learn that all the variables are significant now. The R^2 is slightly down, but in return, one has a much more stable model that is not over-fitted (or at least less over-fitted).

Some more tests can be done. We should also make a Q-Q plot to make sure the residuals are normally distributed. This is done with the function `qqnorm()`.

```
qqnorm(m2$residuals)
qqline(m2$residuals, col = 'blue', lwd = 2)
```

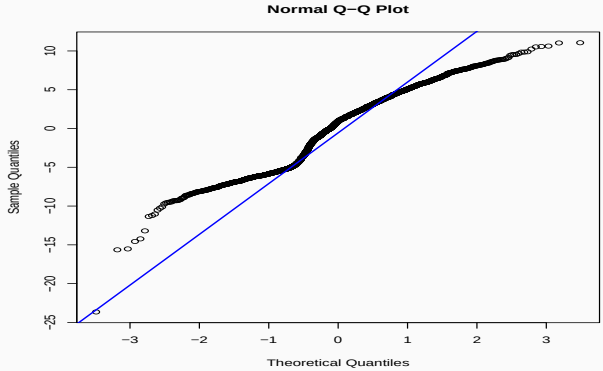


Figure 40: The Q-Q plot of our naive model to forecast the next opening price of the HSBC stock. The results seems to be reasonable.

Figure 40 on slide 203 shows that the model does capture well the tail-behaviour of the forecasted variable. However, the predicting power is not great.

Testing the model robustness i

To check the robustness of our model we should now check how well it fits the test-data. The idea is that since the model was built only on the training data, that we can assess its robustness by checking how well it does on the test-data.

First, we prepare the test data in the same way as the training data:

```
m.qm.tst <- specifyModel(Next(Op(HSBC.test)) ~  Ad(HSBC.test)
            + Hi(HSBC.test) - Lo(HSBC.test) + Vo(HSBC.test))

D.tst <- modelData(m.qm.tst)
D.tst$diff.HSBC.test <- D.tst$Hi.HSBC.test-D.tst$Lo.HSBC.test
#tail(D.tst)                          # the last value is NA
D.tst <- D[-nrow(D.tst),]  # remove the last value that is NA

colnames(D.tst) <- c("Next.Op","Ad","Hi","Lo","Vo","Diff")
```

We could of course use the function `predict()` to find the predictions of the model, but here we illustrate how coefficients can be extracted from the model object and used to obtain these predictions. For the ease of reference we will name the coefficients.

```
a   <- coef(m2)['(Intercept)']
bAd <- coef(m2)['D$Ad']
bD  <- coef(m2)['D$Diff']
est <- a + bAd * D.tst$Ad + bD * D.tst$Diff
```

Testing the model robustness ii

Now, we can calculate all possible measures of model power.

```
# -- Mean squared prediction error (MSPE):
#sqrt(mean(((predict(m2,newdata = D.tst) - D.tst$Next.Op)^2)))
sqrt(mean(((est - D.tst$Next.Op)^2)))
## [1] 4.862097

# -- Mean absolute errors (MAE):
mean((abs(est - D.tst$Next.Op)))
## [1] 4.174993

# -- Mean absolute percentage error (MAPE):
mean((abs(est - D.tst$Next.Op))/D.tst$Next.Op)
## [1] 0.09218155

# -- Squared sum of residuals:
print(sum(residuals(m2)^2))
## [1] 48544.39

# -- Confidence intervals for the model:
print(confint(m2))
##                 2.5 %    97.5 %
## (Intercept) 9.3520325 12.552698
## D$Ad        0.9899451  1.081063
## D$Diff      5.7217706  7.205149
```

Testing the model robustness iii

These values give us an estimate on what error can be expected by using this simple model.

```
# Compare the coefficients in a refit:
m3 <- lm(D.tst$Next.Op ~ D.tst$Ad + D.tst$Diff)
summary(m3)
##
## Call:
## lm(formula = D.tst$Next.Op ~ D.tst$Ad + D.tst$Diff)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -23.6336  -4.9728   0.9232   3.8609  11.0621
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.95993    0.81620   13.43   <2e-16 ***
## D.tst$Ad     1.03528    0.02324   44.55   <2e-16 ***
## D.tst$Diff   6.45949    0.37829   17.08   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.866 on 2050 degrees of freedom
## Multiple R-squared:  0.4947,Adjusted R-squared:  0.4942
## F-statistic:  1004 on 2 and 2050 DF,  p-value: < 2.2e-16
```

One will notice that the estimates for the coefficients are close to the values found in model $m2$. Since the last model, $m3$, includes the most recent data it is probably best to use that one and even update it regularly with new data.

Finally, one could compare the models fitted on the training data and on the test-data and consider if on what time horizon the model should be calibrated before use. One can consider the whole dataset, the last five years, the training dataset, etc. The choice will depend on the reality of the environment rather than on naive mathematics. Although one machine-learning approach would consist of using all possible data-horizons and finding the optimal one.

part 05: Modelling

↓

chapter 27:

Multi Criteria Decision Analysis (MCDA)

PART 05: MODELLING

↓

CHAPTER 27: MULTI CRITERIA DECISION ANALYSIS (MCDA)

↓

SECTION 1:

What and Why

Decision Levels in Companies

1. *Super-strategic:* Mission statement (typically the founders, supervisory board and/or owners have decided this) — this should not be up to discussion, so nothing to decide here (but note that the company most probably started by a biased vision and a bold move on what was actually a multi-criteria problem).
2. *Managerial Control / strategic:* Typical the executive management (executive committee) — almost all problems will be ideally fit for MCDA analysis.
3. *Operational Control / tactical:* Typical middle management — some multi criteria problems, but most probably other methods are more fit.[6]

PART 05: MODELLING

↓

CHAPTER 27: MULTI CRITERIA DECISION ANALYSIS (MCDA)

↓

SECTION 2:

General Work-flow

Make sure that the problem is well understood, that all ideas are on the table, that the environment is taken into account, and that we view the issue at hand through different angles and from different points of view. Use for example exploratory techniques such as:

- SWOT analysis,
- 7Ps of Marketing,
- Business Model Canvas,
- NPV, IRR, cost benefit analysis, etc.,
- time-to-break-even, time to profit, largest cumulated negative, etc.,
- two-parameter criteria (e.g. income/cost) referred,
- make sure that the problem is within one level of decision (strategic / managerial / operational) — see p. 210.

Step 2: Identify the Problem at Hand

Make sure that the question is well formulated and is that it is *the* right questions to ask at this moment in these circumstances.

- Brainstorming techniques or focus groups to
  - get all alternatives
  - get all criteria
  - understand interdependencies
  - etc..
- Make sure you have a clear picture on what the problem is, what the criteria and what the possible alternatives are
- Note: This step is best within one level of decision (strategic/managerial/operational).

Step 3: Get Data, Construct and Normalise the Decision Matrix

This step makes the problem quantifiable. At the end of this step, we will have numbers for all criteria for all alternative solutions.

If we miss data, we can sometimes mitigate this by adding a best estimate for that variable, and then using “risk” as an extra parameter.

The work-flow can be summarised as follows:

1. Define how to measure all solutions for all criteria, i.e. make sure we have an ordinal scale for all criteria.
2. Collect all data so that you can calculate all criteria for all solutions.
3. Put these number is a “decision matrix”.
4. Make sure that the decision matrix is as small as possible: can some criteria be combined into one? For example, it might be useful to fit criteria such as the presence of tram, bus, parking, etc. into one “commuting convenience” criterion.

Normalizing a decision matrix is making sure that

1. All criteria need to be maximized.
2. The lowest alternative for each criterion has a value 0 and the highest equals 1.

Step 4: Leave Out Unacceptable and Inefficient Alternatives

1. Leave out all alternative that do not satisfy the **minimal criteria** – eventually rethink the minimal criteria.
2. Drop the non-optimal solutions (the “dominated ones”).
3. Consider dropping the alternatives that score lowest on some key-criteria.

Step 5: Use a Multi Criteria Decision Method to Get a Ranking

If the problem cannot be reduced to a mono criterion problem then we will necessarily have to make some trade-off when selecting a solution. A – very subjective – top-list of multi criteria decision methods (MCDMs) is the following.

1. Weighted sum method.
2. ELECTRE (especially I and II).
3. PROMethEE (I and II).
4. PCA analysis (aka “Gaia” in this context).

Step 6: Recommend a Solution

In practice, we never make a model or analysis just out of interest, there is always a goal when we do something. Doing something with our work is the reason why we do it in the first place. The data scientist needs to help the management to make good decisions. Therefore it is necessary to write good reports, make presentations, discuss the results and make sure that the decision maker has understood your assumptions and has a fair idea of what the model does.

This step could also be called “do something with the results.”

Keep the following into account:

- Connect back to the company, its purpose and strategic goals (steps 1 and 2)
- Provide the rationale
- Provide confidence to decision makers
- Conclude
- Make an initial plan (assuming an Agile approach, and suggest how to omplement the proposed solution).

Definition 7 (MCDA wording)

- A possible solution for the key-question a_i is called a **alternative**.
- The **set of all alternatives** is $\mathcal{A}$ (in what follows we assume all alternatives to be discrete, and $\mathcal{A}$ is finite (and hence countable – we assume A possible alternatives that are worth to consider) — as opposed to continuous.[a]
- A **criterion** is a measure for success, it is considered to be a function on $\mathcal{A}$ that is indicative of how good an alternative is for on aspect. We consider – without loss of generality – K possible criteria.
- The **decision matrix** $M = (m_{ik})$, is an $A \times K$ matrix for which we choose
  - the alternatives to be headings of the rows (so M has A rows) and
  - the criteria to be headings of the columns (so M has K columns).
- The **normalized decision matrix** is $M = (m_{ik})$, so that $\forall k \in \{1 \dots K\} : \exists i : m_{ik} = 0$ and $\forall k \in \{1 \dots K\} : \exists i : m_{ik} = 1$
- An alternative that cannot be rejected (is not dominated nor preferred under another alternative) is a **solution**.

[a] So, we consider in this chapter problems of *choice* and not problems of *design*.

PART 05: MODELLING

↓

CHAPTER 27: MULTI CRITERIA DECISION ANALYSIS (MCDA)

↓

SECTION 3:

Identify the Issue at Hand: Steps 1 and 2

R-bank is UK based and till now it has 10 000 people working in five large service centres in Asia and South America. These centres are in Bangalore, Delhi, Manilla, and Hyderabad and São Paulo. These cities also happen to be top destinations for Shared Service Centres (SSC) and Business Process Outsourcing (BPO) – as presented by the Tholons index (see `http://www.tholons.com`).[7]

The bank wants to create a central analytics function to supports its modelling and in one go it will start building one central data warehouse with data scientists to make sense of it for commercial and internal reasons (e.g. risk management).

For possible destinations we retain the top ten of Tholons:

1. Bangalore,
2. Delhi,
3. Manilla,
4. Hyderabad,
5. São Paulo,
6. Dublin,
7. Kraków,
8. Chennai, and
9. Buenos Aires.

Definition of Criteria

1. *Talent:* Availability of talent and skills (good universities and enough students)
2. *Stability:* Political stability and fiscal stability
3. *Cost:* The of running the centre
4. *Cost inflation:* Salary inflation
5. *Travel:* Cost and convenience of travelling to the centre (important since we expect lots of interaction between the headquarters and the SSC Risk and Analytics)
6. *Time-zone* Time-zone overlap (as alternative to travel)
7. *Infrastructure:* Office space, roads, etc.
8. *Life quality:* Personal risk and quality of life (museums, restaurants, public transport, etc.)
9. An international airport in close proximity.

PART 05: MODELLING

↓

CHAPTER 27: MULTI CRITERIA DECISION ANALYSIS (MCDA)

↓

SECTION 4:

Step 3: the Decision Matrix

Quantify all Criteria for All Alternatives

1. *Talent:* Use Tholons' "talent, skill and quality" 2017 index – see `http://www.tholons.com`
2. *Stability:* the 2017 political stability index of the World Bank – see `http://info.worldbank.org/governance/WGI`
3. *Cost:* Use Tholons' "cost" 2017 index – see `http://www.tholons.com`
4. *Cost inflation* "Annualized average growth rate in per capita real survey mean consumption or income, total population (%)" from `https://data.worldbank.org`
5. *Travel:* Cost and convenience of travelling to the centre (important since we expect lots of interaction between the headquarters and the SSC Risk and Analytics) – our assessment of airline ticket price between R-bank's headquarters, the travel time, etc.
6. *Time-zone:* Whether there is a big time-zone differnce – this is roughly one point if in the same time-zone as R-bank's headquarters, zero if more than 6 hours difference.
7. *Infrastructure:* Use Tholons' "infrastructure" 2017 index – see `http://www.tholons.com`
8. *Life quality:* Use Tholons' "risk and quality of life" 2017 index – see `http://www.tholons.com`
9. International airport in close proximity: Not withheld as a criterion, because all cities in the Tholons top-10 have international airports.

Location	tlnt	stab	cost	infl	trvl	tm-zn	infr	life
Bangalore	1.6	-0.83	1.4	4.7%	H	1	0.9	1.1
Mumbai	1.8	-0.83	1.0	4.7%	H	1	0.9	0.8
Delhi	1.8	-0.83	1.2	4.7%	H	1	0.9	0.6
Manilla	1.6	-1.24	1.4	2.8%	H	1	0.9	0.8
Hyderabad	0.9	-0.83	1.4	4.7%	H	1	0.7	0.8
Sao Polo	0.9	-0.83	0.8	4.7%	H	1	0.7	0.6
Dublin	0.7	1.02	0.2	2.0%	L	3	1.1	1.3
Krakow	1.1	0.52	1.0	1.3%	L	3	0.6	0.9
Chennai	1.2	-0.83	1.3	4.7%	H	1	0.8	0.5
Buenos Aires	0.9	0.18	0.9	7.3%	H	1	0.8	0.6

Table 2: The decision matrix summarises the information that we have gathered. In this stage the matrix will mix variables in different units, and even qualitative appreciations (e.g. high and low).

Creating This Decision Matrix in R

```
M0 <- matrix(c(
 1.6 , -0.83 , 1.4 , 4.7 , 1 , 0.9 , 1.1 ,
 1.8 , -0.83 , 1.0 , 4.7 , 1 , 0.9 , 0.8 ,
 1.8 , -0.83 , 1.2 , 4.7 , 1 , 0.9 , 0.6 ,
 1.6 , -1.24 , 1.4 , 2.8 , 1 , 0.9 , 0.8 ,
 0.9 , -0.83 , 1.4 , 4.7 , 1 , 0.7 , 0.8 ,
 0.9 , -0.83 , 0.8 , 4.7 , 1 , 0.7 , 0.6 ,
 0.7 ,  1.02 , 0.2 , 2.0 , 3 , 1.1 , 1.3 ,
 1.1 ,  0.52 , 1.0 , 1.3 , 3 , 0.6 , 0.9 ,
 1.2 , -0.83 , 1.3 , 4.7 , 1 , 0.8 , 0.5 ,
 0.9,  0.18 , 0.9 , 7.3 ,  1 , 0.8 , 0.6 ),
 byrow = TRUE, ncol = 7)
colnames(M0) <- c("tlnt","stab","cost","infl","trvl","infr","life")
# We use the IATA code of a nearby airport as abbreviation,
# so, instead of:
# rownames(M0) <- c("Bangalore", "Mumbai", "Delhi", "Manilla", "Hyderabad",
#                   "Sao Polo", "Dublin", "Krakow", "Chennai", "Buenos Aires")
# ... we use this:
rownames(M0) <- c("BLR", "BOM", "DEL", "MNL", "HYD", "GRU",
                  "DUB", "KRK", "MAA", "EZE")

M0  # inspect the matrix
##     tlnt  stab cost infl trvl infr life
## BLR  1.6 -0.83  1.4  4.7    1  0.9  1.1
## BOM  1.8 -0.83  1.0  4.7    1  0.9  0.8
## DEL  1.8 -0.83  1.2  4.7    1  0.9  0.6
## MNL  1.6 -1.24  1.4  2.8    1  0.9  0.8
## HYD  0.9 -0.83  1.4  4.7    1  0.7  0.8
## GRU  0.9 -0.83  0.8  4.7    1  0.7  0.6
```

Normalising the Decision Matrix in R i

Normalising the Decision Matrix in R ii

```r
# Political stability is a number between -2.5 and 2.5
# So, we make it all positive by adding 2.5:
M0[,2] <- M0[,2] + 2.5

# Lower wage inflation is better, so invert the data:
M0[,4] <- 1 / M0[,4]

# Then we define a function:

# mcda_rescale_dm
# Rescales a decision matrix M
# Arguments:
#    M -- decision matrix
#         criteria in columns and higher numbers are better.
# Returns
#    M -- normalised decision matrix
mcda_rescale_dm <- function (M) {
  colMaxs <- function(M) apply(M, 2, max, na.rm = TRUE)
  colMins <- function(M) apply(M, 2, min, na.rm = TRUE)
  M <- sweep(M, 2, colMins(M), FUN="-")
  M <- sweep(M, 2, colMaxs(M) - colMins(M), FUN="/")
  M
}

# Use this function:
M <- mcda_rescale_dm(M0)

# Show the new decision matrix:
```

PART 05: MODELLING

↓

CHAPTER 27: MULTI CRITERIA DECISION ANALYSIS (MCDA)

↓

SECTION 5:

Step 4: Delete Inefficient and Unfit Alternatives

Function to Create a Dominance Matrix i

Function to Create a Dominance Matrix ii

```r
# mcda_get_dominated
# Finds the alternatives that are dominated by others
# Arguments:
#    M -- normalized decision matrix with alternatives in rows,
#         criteria in columns and higher numbers are better.
# Returns
#    Dom -- prefM -- a preference matrix with 1 in position ij
#                    if alternative i is dominated by alternative j.
mcda_get_dominated <- function(M) {
  Dom  <- matrix(data=0, nrow=nrow(M), ncol=nrow(M))
  dominatedOnes <- c()
  for (i in 1:nrow(M)) {
    for (j in 1:nrow(M)) {
      isDom <- TRUE
      for (k in 1:ncol(M)) {
        isDom <- isDom && (M[i,k] >= M[j,k])
      }
      if(isDom && (i != j)) {
        Dom[j,i] <- 1
        dominatedOnes <- c(dominatedOnes,j)
      }
    }
  }
  colnames(Dom) <- rownames(Dom) <- rownames(M)
  Dom
}
```

Get the Dominating Alternatives with that Function i

```r
# mcda_get_dominants
# Finds the alternatives that dominate others
# Arguments:
#    M -- normalized decision matrix with alternatives in rows,
#         criteria in columns and higher numbers are better.
# Returns
#    Dom -- prefM -- a preference matrix with 1 in position ij
#                    if alternative i dominates alternative j.
mcda_get_dominants <- function (M) {
   M <- t(mcda_get_dominated(M))
   class(M) <- "prefM"
   M
   }
```

Apply the Function to Get the Dominated Alternatives i

```
Dom <- mcda_get_dominants(M)
print(Dom)
##     BLR BOM DEL MNL HYD GRU DUB KRK MAA EZE
## BLR   0   0   0   0   1   1   0   0   1   0
## BOM   0   0   0   0   0   1   0   0   0   0
## DEL   0   0   0   0   0   1   0   0   0   0
## MNL   0   0   0   0   0   0   0   0   0   0
## HYD   0   0   0   0   0   1   0   0   0   0
## GRU   0   0   0   0   0   0   0   0   0   0
## DUB   0   0   0   0   0   0   0   0   0   0
## KRK   0   0   0   0   0   0   0   0   0   0
## MAA   0   0   0   0   0   0   0   0   0   0
## EZE   0   0   0   0   0   0   0   0   0   0
## attr(,"class")
## [1] "prefM"
```

We see that

- Hyderabad (HYD) is dominated by Bangalore: it has a worse talent pool and lower quality of life, while it scores the same for all other criteria.
- São Paulo is dominated by Bangalore, Mumbai, Delhi, and Hyderabad.
- Chennai is dominated by Bangalore.

Deleting the Dominated Alternatives i

```
# mcda_del_dominated
# Removes the dominated alternatives from a decision matrix
# Arguments:
#    M -- normalized decision matrix with alternatives in rows,
#         criteria in columns and higher numbers are better.
# Returns
#    A decision matrix without the dominated alternatives
mcda_del_dominated <- function(M) {
  Dom <- mcda_get_dominated(M)
  M[rowSums(Dom) == 0,]
}
```

This function allows us to reduce the decision matrix M to $M1$ that only contains alternatives that are not dominated.

```
M1 <- mcda_del_dominated(M)
knitr::kable(round(M1,2))
```

	tlnt	stab	cost	infl	trvl	infr	life
BLR	0.82	0.18	1.00	0.12	0	0.6	0.75
BOM	1.00	0.18	0.67	0.12	0	0.6	0.38
DEL	1.00	0.18	0.83	0.12	0	0.6	0.12
MNL	0.82	0.00	1.00	0.35	0	0.6	0.38
DUB	0.00	1.00	0.00	0.57	1	1.0	1.00
KRK	0.36	0.78	0.67	1.00	1	0.0	0.50
EZE	0.18	0.63	0.58	0.00	0	0.4	0.12

PART 05: MODELLING

↓

CHAPTER 27: MULTI CRITERIA DECISION ANALYSIS (MCDA)

↓

SECTION 6:

Plotting Preference Relationships

Creating an S3 Method to Plot `prefM` Objects

```r
# First, we load diagram:
require(diagram)

# plot.prefM
# Specific function to handle objects of class prefM for the
# generic function plot()
# Arguments:
#    PM  -- prefM -- preference matrix
#    ... -- additional arguments passed to plotmat()
#           of the package diagram.
plot.prefM <- function(PM, ...)
{
  X <- t(PM) # We want arrows to mean '... is better than ...'
             # plotmat uses the opposite convention, because it expects flows.
  plotmat(X,
          box.size    = 0.1,
          cex.txt     = 0,
          lwd         = 5 * X,  # lwd proportional to preference
          self.lwd    = 3,
          lcol        = 'blue',
          self.shiftx = c(0.06, -0.06, -0.06, 0.06),
          box.lcol    = 'blue',
          box.col     = 'khaki3',
          box.lwd     = 2,
          relsize     = 0.9,
          box.prop    = 0.5,
          endhead     = FALSE,
          main        = "",
```

Testing the Function on the Dominance Matrix i

```r
# We pass the argument 'curve = 0' to the function plotmat, since otherwise
# the arrow from BLR to MAA would be hidden after the box of EZE.
plot(Dom, curve = 0)
```

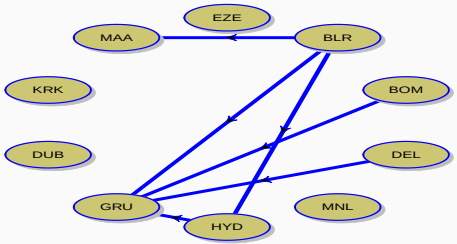


Figure 41: A visualization of the dominance relationship.

PART 05: MODELLING

↓

CHAPTER 27: MULTI CRITERIA DECISION ANALYSIS (MCDA)

↓

SECTION 7:

Step 5: MCDA Methods

Non-Compensatory Methods vs. Compensatory Methods

- Non-compensatory methods
  - for example, “dominance” is one of those methods
  - **they do not allow weaknesses on one attribute to be compensated by strong aspects of other attributes**, but …
  - typically they do not lead to a unique solution
  - typically they even are insufficient to find a small enough set of the best solutions
- Compensatory methods
  - **They allow full or partial compensation of weaknesses**
  - the rest of this course …

The MaxMin Method

1. find the weakest attribute for all solutions
2. select the solution that has the highest weak attribute (0 in a normalized decision matrix)

This method makes sense if

- the attribute values are expressed in the same units, and
- when the “a chain is as weak as the weakest link reasoning” makes sense.

1. Find the strongest attribute for all solutions.
2. Select the solution that has the strongest strong attribute.

This method makes sense if

- the attribute values are expressed in the same units, and
- when one knows that the best of the best in one attribute is most important.

The MCDA is replaced by finding the maximum for:

$$\max_{\mathbf{x}\in\mathcal{A}}\{N(a)\}$$

with $N(.)$ the function $\Re^n \mapsto \Re^n$ so that

$$N(\mathbf{a_i}) = \sum_{k=1}^{K} w_k\, m_{ik} \qquad \text{or}$$

$$\mathbf{N(a)} = \mathbf{M.w}$$

where **M** is the decision matrix where each element is transformed according to a certain function.

In R this can be obtained as follows.

```r
# mcda_wsm
# Calculated the Weigthed Sum MCDA for a decision matrix M and weights w.
# Arguments:
#    M -- normalized decision matrix with alternatives in rows,
#         criteria in columns and higher numbers are better.
#    w -- numeric vector of weights for the criteria
# Returns
#    a vector with a score for each alternative
mcda_wsm <- function(M, w) {
  X <- M %*% w
  colnames(X) <- 'pref'
  X
}
```

The WSM Applied to our Example

Taking into account that the SSC will not be very large, that we cannot expect employees just to be ready (so we will do a lot of training ourselves and work with universities to fine-tune curricula, etc.), we need a long time to set up such centre of expertise and hence need stability, etc. we came up with the following weights.

```
# The critia: "tlnt" "stab" "cost" "infl" "trvl" "infr" "life"
w <- c(        0.125, 0.2,   0.2,   0.2,  0.175,  0.05,  0.05)
w <- w / sum(w)  # the sum was 1 already, but just to be sure.

# Now we can execute our function mcda_wsm():
mcda_wsm(M1, w)
##          pref
## BLR 0.4282418
## BOM 0.3628739
## DEL 0.3819215
## MNL 0.4162013
## DUB 0.5898333
## KRK 0.7309687
## EZE 0.2850577
```

Rewrite the function `mcda_wsm()` to Return a Score Matrix Object

The complete ranking can be represented with `plotmat` but it might make more sense to use `ggplot2`. To do so neatly, we take a step back and re-write the function `mcda_wsm()` and make it return a "matrix of scores" ("scoreM" for short).

```r
# mcda_wsm_score
# Returns the scores for each of the alternative for each of
# the criteria weighted by their weights.
# Arguments:
#    M -- normalized decision matrix with alternatives in rows,
#         criteria in columns and higher numbers are better.
#    w -- numeric vector of weights for the criteria
# Returns
#    a score-matrix of class scoreM
mcda_wsm_score <- function(M, w) {
  X <- sweep(M1, MARGIN = 2, w, `*`)
  class(X) <- 'scoreM'
  X
}
```

Now we are ready to define a specialised plotting function for objects of the class “scoreM:”

```
# plot.scoreM
# Specific function for an object of class scoreM for the
# generic function plot().
# Arguments:
#    M -- scoreM -- score matrix
# Returns:
#    plot
plot.scoreM <- function (M) {
  # 1. order the rows according to rowSums
  M <- M[order(rowSums(M), decreasing = T),]

  # 2. use a bar-plot on the transposed matrix
  barplot(t(M),
     legend = colnames(M),
     xlab   = 'Score',
     col    = rainbow(ncol(M))
     )
}
```

```
# Whith the normalised decision matrix M1 and the weights w, we calculate the score matrix:
sM <- mcda_wsm_score(M1, w)

# Then we plot the result:
plot(sM)
```

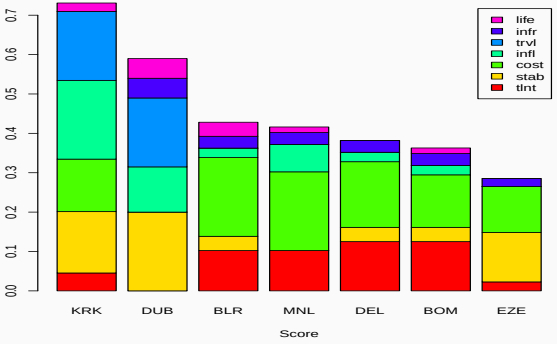


Figure 42: The scores of different cities according to the WSM.

Let w_j be the weight of the criterion j, and m_{ij} the score (performance) of alternative i on criterion j then solutions can be ranked according to their total score as follows

$$P(a_i) = \Pi_{j=1}^{n} (m_{ij})^{w_j}$$

Let w_j be the weights of the criteria, and m_{ij} the score (performance) of alternative i on criterion j then a solution a_i is preferred over a solution a_n if the preference $P(a_i, a_j) > 1$, with

$$P(a_i, a_j) := \Pi_{k=1}^{n} \left(\frac{m_{ik}}{m_{jk}} \right)^{w_k}$$

This form of the WPM is often called **dimensionless analysis** because its mathematical structure eliminates any units of measure. Note however, that it requires a ratio scale.

If the decision matrix M has elements m_{ik}, then we prefer the alternative a_i over the alternative a_j for criterion k if $m_{ik} > m_{jk}$. In other words, we prefer alternative i over alternative j for criterion k if its score is higher for that criterion. The amount of preference can be captured by a function $\Pi()$.

In ELECTRE the preference function is supposed to be a step-function.

Definition 8 (Preference of one solution over another)

The preference of a solution a_i over a solution a_j is

$$\pi^+(a_i, a_j) := \sum_{k=1}^{K} \pi_k(m_{ik} - m_{jk})\, w_k$$

We can also define an anti-prefrence as a measure that estimates the amount of negative preference that comes into one solution as compared to another.

Definition 9 (Anti-preference of one solution over another)

The anti-preference of a solution a_i over a solution a_j is

$$\pi^-(a_i, a_j) := \sum_{k=1}^{K} \pi_k (m_{jk} - m_{ik})\, w_k$$

We note that:

$$\begin{aligned}
\pi^+(a_i, a_j) &= \sum_{k=1}^{K} \pi_k (m_{ik} - m_{jk})\, w_k \\
&= -\sum_{k=1}^{K} \pi_k (m_{jk} - m_{ik})\, w_k \\
&= -\pi^+(a_j, a_i) \\
&= -\pi^-(a_i, a_j) \\
&= \pi^-(a_j, a_i)
\end{aligned}$$

Even with a preference function $\pi()$ that is a strictly increasing function of the difference in score, it might be that some solutions have the same score for some criteria and hence are incomparable for these criteria. So, it makes sense to define a degree of "indifference."

Definition 10 (The Weighted Degree of Indifference)

The Weighted Degree of Indifference of a solution *a* and *b* is

$$\begin{aligned}\pi^0(a,b) &= \sum_{j=a}^{k} w_j - \pi^+(a_i,a_j) - \pi^-(a_i,a_j) \\ &= 1 - \pi^+(a_i,a_j) - \pi^-(a_i,a_j)\end{aligned}$$

The last line assumes that the sum of weights is one.

There are two particularly useful possibilities for this index of comparability. We will call them C_1 and C_2.

Definition 11 (Index of comparability of Type 1)

$$C_1(a,b) = \frac{\Pi^+(a,b) + \Pi^0(a,b)}{\Pi^+(a,b) + \Pi^0(a,b) + \Pi^-(a,b)}$$

Note that $C_1(a,b) = 1 \Leftrightarrow aDb$. This, however, should not be the case in our example as we already left out all dominated solutions.

Definition 12 (Index of comparability of Type 2)

$$C_2(a,b) = \frac{\Pi^+(a,b)}{\Pi^-(a,b)}$$

Note that $C_2(a,b) = \infty \Leftrightarrow aDb$.

Further, to this index of comparability it makes sense to define a threshold Λ below which we consider the alternatives as "too similar to be discriminated."

For each criterion individually we define:

- for each criterion a maximal discrepancy in the "wrong" direction if a preference would be stated: $r_k, k \in \{1 \dots K\}$. This will avoid that a solution a is preferred over b while it is too much worse than b for at least one criterion.

With all those definitions we can define the preference structure as follows:

- for C_1 : $\left.\begin{array}{l} \Pi^+(a,b) > \Pi^-(a,b) \\ C_1(a,b) \geq \Lambda_1 \\ \forall j : d_j(a,b) \leq r_j \end{array}\right\} \Rightarrow a \succ b$
- for C_2 : $\left.\begin{array}{l} \Pi^+(a,b) > \Pi^-(a,b) \\ C_2(a,b) \geq \Lambda_2 \\ \forall j : d_j(a,b) \leq r_j \end{array}\right\} \Rightarrow a \succ b$

In a last step one can present the results graphically and present the kernel (the best solutions) to the decision makers. The kernel consists of all alternatives that are "efficient" (there is no other alternative that is preferred over the latter).

Definition 13 (Kernel of an MCDA problem)

The kernel of a MCDA problem is the set

$$\mathcal{K} = \{a \in \mathcal{A} \mid \nexists b \in \mathcal{A} : b \succ a\}$$

Below is one way to program the ELECTRE I algorithm in R. One of the major choices that we made was create a function with a side effect. This is not the best solution if we want others to use our code (e.g. if we would like to wrap the functions in a package). The alternative would be to create a list of matrices, that then could be returned by the function.

Since we are only calling the following function within another function this is not toxic, and suits our purpose well.

```r
# mcda_electre Type 2
# Push the preference matrixes PI.plus, PI.min and
# PI.indif in the environment that calls this function.
# Arguments:
#    M -- normalized decision matrix with alternatives in rows,
#         criteria in columns and higher numbers are better.
#    w -- numeric vector of weights for the criteria
# Returns nothing but leaves as side effect:
#    PI.plus   -- the matrix of preference
#    PI.min    -- the matrix of non-preference
#    PI.indif  -- the indifference matrix
mcda_electre <- function(M,  w) {
  # initializations
  PI.plus  <<- matrix(data=0, nrow=nrow(M), ncol=nrow(M))
  PI.min   <<- matrix(data=0, nrow=nrow(M), ncol=nrow(M))
  PI.indif <<- matrix(data=0, nrow=nrow(M), ncol=nrow(M))

  # calculate the preference matrix
  for (i in 1:nrow(M)){
    for (j in 1:nrow(M)) {
      for (k in 1:ncol(M)) {
        if (M[i,k] > M[j,k]) {
          PI.plus[i,j] <<- PI.plus[i,j] + w[k]
        }
        if (M[j,k] > M[i,k]) {
          PI.min[i,j] <<- PI.min[i,j] + w[k]
        }
        if (M[j,k] == M[i,k]) {
          PI.indif[i,j] <<- PI.indif[i,j] + w[k]
```

The function `mcda_electre1()` is now ready for use. We need to provide the decision matrix, weights and the cut-off value and a vector for maximum inverse preferences. The code below does this, prints the preference relations a matrix and finally plots them with our custom method `plot.prefM()` in Figure 43 on slide 259.

```
# the critia: "tlnt" "stab" "cost" "infl" "trvl" "infr" "life"
w <- c(        0.125, 0.2,   0.2,   0.2,  0.175,  0.05,  0.05)
w <- w / sum(w)  # the sum was 1 already, but just to be sure.
r  <- c(0.3,    0.5,  0.5,   0.5,   1,     0.9,   0.5)

eM <- mcda_electre1(M1, w, Lambda=0.6, r=r)
print(eM)
##     BLR BOM DEL MNL DUB KRK EZE
## BLR   0   1   1   1   0   0   1
## BOM   0   0   0   0   0   0   1
## DEL   0   1   0   0   0   0   1
## MNL   0   1   1   0   0   0   1
## DUB   0   0   0   0   0   0   1
## KRK   0   0   0   0   1   0   1
## EZE   0   0   0   0   0   0   0
## attr(,"class")
## [1] "prefM"

plot(eM)
```

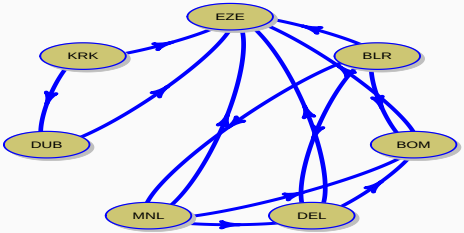


Figure 43: The preference structure as found by the ELECTRE I method given all parameters in the code.

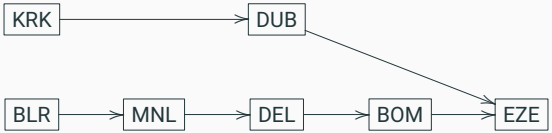


Figure 44: Another representation of Figure 43. It is clear that Krakow and Bangalore are quite different places. Therefore they are not ranked between each other and choosing between them means making compromises.

ELECTRE II: the Idea

Hence, the idea of ELECTRE II was born to force a complete ranking by

- gradually lower the cut-off level Λ_1 and
- increasing the cut-off level for opposite differences in some criteria r_j.

In our example *r* needs to be equal to the unit vector and Λ can be zero in order to obtain a full ranking. The code below uses these values and plots the preference relations in Figure 45 on slide 263.

```
# The critia: "tlnt" "stab" "cost" "infl" "trvl" "infr" "life"
w <- c(        0.125, 0.2,   0.2,   0.2,  0.175,  0.05,  0.05)
w <- w / sum(w)  # the sum was 1 already, but just to be sure.
r  <- c(1,    1,  1,   1,   1,     1,   1)

eM <- mcda_electre1(M1, w, Lambda = 0.0, r = r)
print(eM)
##     BLR BOM DEL MNL DUB KRK EZE
## BLR   0   1   1   1   0   0   1
## BOM   0   0   0   0   0   0   1
## DEL   0   1   0   0   0   0   1
## MNL   0   1   1   0   0   0   1
## DUB   1   1   1   1   0   0   1
## KRK   1   1   1   1   1   0   1
## EZE   0   0   0   0   0   0   0
## attr(,"class")
## [1] "prefM"

plot(eM)
```

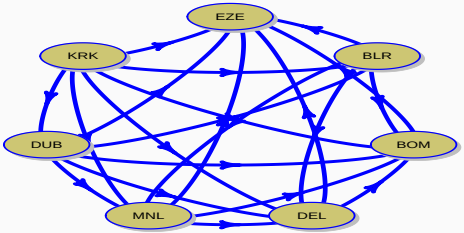


Figure 45: The preference structure as found by the ELECTRE II method given all parameters in the code.

Figure 46: The results for ELECTRE I with comparability index C_2.

ELECTRE: Advantages and Disadvantages

Advantages

- No need to add different variables in different units
- All that is needed is a conversion to “preference” and add this preference
- Richer information than the Weighted Sum Method
- The level of compensation can be controlled

Disadvantages

- There is still an “abstract” concept “preference,” which has little meaning and no pure interpretation
- To make matters worse, there are also the cut-off levels
- So to some extend it is still so that concepts that are expressed in different units are compared in a naive way.

- Enrich the preference structure of the ELECTRE method.
- In the ELECTRE Method one prefers essentially a solution a over b for criterion k if and only if $f_k(a) > f_k(b)$.
- This 0-or-1-relation (black or white) can be replaced by a more gradual solution with different shades of grey.
- This preference function will be called $\pi_k(a, b)$ and it can be different for each criterion.

The idea is that the preference for alternative a_i and a_j can be expressed in function of the weighted sum of differences of their scores m_{ik} in the decision matrix.

$$\pi(a_i, a_j) = \sum_{k=1}^{K} P_k(m_{ik} - m_{jk}) w_k \tag{8}$$

$$= \sum_{k=1}^{K} P_k\left(d_k(a_i, a_j)\right) w_k \tag{9}$$

In which we used the following "distance definition":

Definition 14 (Distance $d_k(a, b)$)

$$d_k(a, b) = f_k(a) - f_k(b)$$

Examples:

- step-function with one step (similar to ELECTRE preferences)
- step-function with more than one step
- step-wise linear function
- $\pi(d) = max(0, min(g \times d, d_0))$ (linear, gearing g)
- sigmoid equation: $\pi(d) = \frac{1}{1-\left(\frac{1}{d_0}-1\right)e^{-dt}}$
- $\pi(d) = tanh(d)$
- $\pi(d) = erf\left(\frac{\sqrt{(\pi)}}{2}d\right)$
- $\pi(d) = \frac{d}{\sqrt{1+x^2}}$
- Gaussian: $\pi(d) = \begin{cases} 0 & \text{for } d < 0 \\ 1 - \exp\left(-\frac{(d-d_0)^2}{2s^2}\right) & \text{for } d \geq 0 \end{cases}$
- ...

The preference function allows us to define a flow of how much each alternative is preferred, Φ_i^+, as well as a measure of how much other alternatives are preferred over this one: Φ_i^-. The process is as follows.

1. Define preference functions $\pi : \mathcal{A} \times \mathcal{A} \mapsto [0, 1]$
2. They should only depend on the difference between the scores of each alternative as summarized in the decision matrix m_{ik}:
$$\pi_k(a_i, a_j) = \pi_j\left(m_{ik} - m_{jk}\right) = \pi_j\left(d_k(a_i, a_j)\right)$$
3. Define a preference index: $\Pi(a_i, a_j) = \sum_{k=1}^{K} w_k \pi_k(a_i, a_j)$
4. Then sum all those flows for each solution – alternative – to obtain
   1. a positive flow: $\Phi^+(a_i) = \frac{1}{K-1} \sum_{\mathbf{a_j} \in \mathcal{A}} \Pi(a_i, a_j) = \frac{1}{K-1} \sum_{k=1}^{K} \sum_{j=1}^{A} \pi_k(a_i, a_j)$
   2. a negative flow: $\Phi^-(a_i) = \frac{1}{K-1} \sum_{\mathbf{a_j} \in \mathcal{A}} \Pi(a_j, a_i) = \frac{1}{K-1} \sum_{k=1}^{K} \sum_{j=1}^{A} \pi_k(a_j, a_i)$
   3. a net flow: $\Phi(a_i) = \Phi^+(a_i) - \Phi^-(a_i)$

   where the w_k are the weights of the preference for each criteria so that $\sum_{k=1}^{K} w_k = 1$ and $\forall k \in \{1 \dots K\} : w_k > 0$

Based on these flows, we can define the preference relations for PROMethEE I as follows:

- $$a \succ b \Leftrightarrow \begin{cases} \Phi^+(a) \geq \Phi^+(b) \wedge \Phi^-(a) < \Phi^-(b) \text{ or} \\ \Phi^+(a) > \Phi^+(b) \wedge \Phi^-(a) \leq \Phi^-(b) \end{cases}$$
- indifferent $\Leftrightarrow \Phi^+(a) = \Phi^+(b) \wedge \Phi^-(a) = \Phi^-(b)$
- in all other cases: no preference relation

We will first define a base function that calculates the flows Φ and pushes the results a in the environment where the function is called (similar to the approach for the ELECTRE method).

```
# mcda_promethee
# delivers the preference flow matrices for the Promethee method
# Arguments:
#    M      -- decision matrix
#    w      -- weights
#    piFUNs -- a list of preference functions,
#              if not provided min(1,max(0,d)) is assumed.
# Returns (as side effect)
# phi_plus <<- rowSums(PI.plus)
# phi_min  <<- rowSums(PI.min)
# phi_     <<- phi_plus - phi_min
#
mcda_promethee <- function(M, w, piFUNs='x')
{
  if (piFUNs == 'x') {
       # create a factory function:
       makeFUN <- function(x) {x; function(x) max(0,x) }
       P <- list()
       for (k in 1:ncol(M)) P[[k]] <- makeFUN(k)
       } # in all other cases we assume a vector of functions
# initializations
PI.plus  <<- matrix(data=0, nrow=nrow(M), ncol=nrow(M))
PI.min   <<- matrix(data=0, nrow=nrow(M), ncol=nrow(M))
# calculate the preference matrix
for (i in 1:nrow(M)){
  for (j in 1:nrow(M)) {
```

Now, we can define a function `mcda_promethee1()` that calls the function `mcda_promethee()` to define the preference flows.

```r
# mcda_promethee1
# Calculates the preference matrix for the Promethee1 method
# Arguments:
#    M      -- decision matrix
#    w      -- weights
#    piFUNs -- a list of preference functions,
#              if not provided min(1,max(0,d)) is assumed.
# Returns:
#    prefM object -- the preference matrix
#
mcda_promethee1 <- function(M, w, piFUNs='x') {
  # mcda_promethee adds phi_min, phi_plus & phi_ to this environment:
  mcda_promethee(M, w, piFUNs='x')

  # Now, calculate the preference relations:
  pref     <- matrix(data=0, nrow=nrow(M), ncol=nrow(M))
    for (i in 1:nrow(M)){
      for (j in 1:nrow(M)) {
        if (phi_plus[i] == phi_plus[j] && phi_min[i]==phi_min[j]) {
            pref[i,j] <- 0
          }
          else if ((phi_plus[i] > phi_plus[j] &&
                    phi_min[i] < phi_min[j] ) ||
                   (phi_plus[i] >= phi_plus[j] &&
                    phi_min[i] < phi_min[j] )) {
            pref[i,j] <- 1
          }
          else {
            pref[i,j] <- NA
```

The function that we have created can also take a list of preference functions via its `piFUNs` argument. Below, we illustrate how this can work and we plot the results in Figure 49 on slide 274.

```
# Make shortcuts for some of the functions that we will use:
gauss_val <- function(d) 1 - exp(-(d - 0.1)^2 / (2 * 0.5^2))
x         <- function(d) max(0,d)
minmax    <- function(d) min(1, max(0,2*(d-0.5)))
step      <- function(d) ifelse(d > 0.5, 1,0)

# Create a list of 7 functions (one per criterion):
f <- list()
f[[1]] <- gauss_val
f[[2]] <- x
f[[3]] <- x
f[[4]] <- gauss_val
f[[5]] <- step
f[[6]] <- x
f[[7]] <- minmax

# Use the functions in mcda_promethee1:
m <- mcda_promethee1(M1, w, f)

# Plot the results:
plot(m)
```

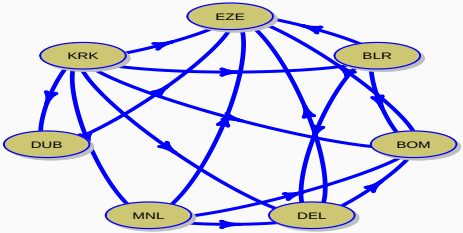


Figure 49: The result for PROMethEE I with different preference functions provided.

Interestingly, the functions that we have provided, do change the preference structure as found by PROMethEE I, even the main conclusions differ. The main changes are that KRK became comparable to BLR and MNL to DEL.

Note that besides the plot that we obtain automatically via our functon `plot.prefM()`, it is also possible to create a plot that uses the transitivity to make the image lighter and easier to read. This is presented in Figure 50

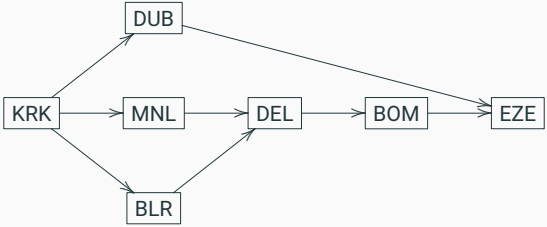


Figure 50: The results for PROMethEE I method with the custom preference functions. In this case there is one clear winner that is preferred over all other options: Krakoów.

Advantages and Disadvantages of PROMethEE I

Advantages:

- It is easier and makes more sense to define a preference function than the parameters Λ_j and $\mathbf{r}$ in ELECTRE.
- It seems to be stable for addition and deletion of alternatives (the ELECTRE and WPM have been proven inconsistent here).
- No comparison of variables in different units.
- The preference is based on rich information.

Disadvantages:

- Does not readily give too much insight in why a solution is preferred.
- Needs more explanation about how it works than the WSM.
- Some decision makers might not have heard about it.
- There are a lot of arbitrary choices to be made, and those choices can influence the result.

We can condense this information further for each alternative:

$$\begin{aligned}\Phi(a) &= \sum_{x \in \mathcal{A}} \sum_{j=1}^{k} \pi_j(f_j(a), f_j(x)) \\ &= \sum_{x \in \mathcal{A}} \pi(a, x)\end{aligned}$$

This results in a preference relation that will almost in all cases show a difference (in a small number of cases there is indifference, but all are comparable – there is no “no preference”)

- $a \succ b \Leftrightarrow \Phi(a) > \Phi(b)$
- indifferent if $\Phi(a) = \Phi(b)$
- in all other cases: no preference relation

Advantages

- Almost sure to get a full ranking.
- The preference structure is rich and preference quantifiable.
- The preferences are transitive: $a \succ b \wedge b \succ c \Rightarrow a \succ c$.
- No conflicting rankings possible, logically consistent for the decision makers.

Disadvantages

- More condensed information (loss of information, more compensation).
- Might be more challenging to understand for some people.
- A lot of arbitrary functions and parameters relating to preference.

In the context of MCDA, this projection in the (PC_1, PC_2) plane is also referred to as method for “geometrical analysis for interactive aid” (Gaia). It is, however, nothing more than one part of a principal component analysis (PCA).

Principal component analysis is part of the functionalities of the package `stats` and hence is available by default. We have already demonstrated how to use PCA in R in Section **??** *"??"* on page **??**, here we only repeat the basics. In the following code, we calculate the principle components (PCs), plot the variance explained per principle component in Figure **??** on slide **??** and the biplot (projection in the in (PC_1, PC_2) plane) in Figure **??** on slide **??**.

```
pca1 <- prcomp(M1)
summary(pca1)
## Importance of components:
##                           PC1    PC2    PC3     PC4    PC5     PC6
## Standard deviation     0.8196 0.4116 0.3492 0.18995 0.1103 0.04992
## Proportion of Variance 0.6626 0.1671 0.1203 0.03559 0.0120 0.00246
## Cumulative Proportion  0.6626 0.8297 0.9499 0.98555 0.9975 1.00000
##                            PC7
## Standard deviation     4.682e-18
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00

# plot for the prcomp object shows the variance explained by each PC
plot(pca1, type = 'l')
```

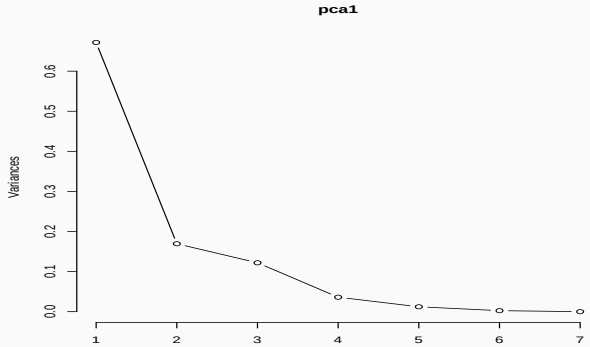


Figure 51: The variance explained by each principal component.

```
# biplot shows a projection in the 2D plane (PC1, PC2)
biplot(pca1)
```

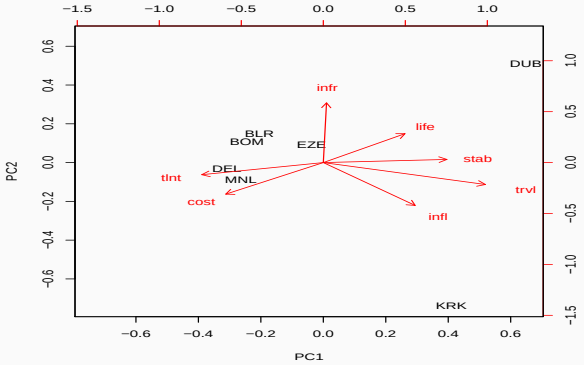
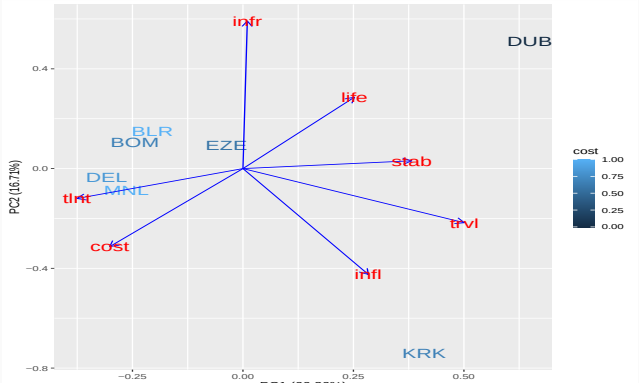


Figure 52: A projection of the space of alternatives in the 2D-plane formed by the two most dominating principal components.

As mentioned in earlier, also with `ggplot2` and `ggfortify` it is easy to obtain professional results with little effort. The code below does this and shows two versions: first, with the labels coloured according to cost (in Figure **??** on slide **??**), second with the visualisation of two clusters in Figure **??** on slide **??**

```
library(ggplot2)
library(ggfortify)
library(cluster)

# Autoplot with labels colored
autoplot(pca1, data = M1, label = TRUE, shape = FALSE, colour='cost', label.size = 6,
         loadings = TRUE, loadings.colour = 'blue',
         loadings.label = TRUE, loadings.label.size = 6
         )
```



```
# Autoplot with visualization of 2 clusters
autoplot(fanny(M1,2), label=TRUE, frame=TRUE, shape = FALSE, label.size = 6,
         loadings = TRUE,  loadings.colour = 'blue',
         loadings.label = TRUE, loadings.label.size = 6)
```

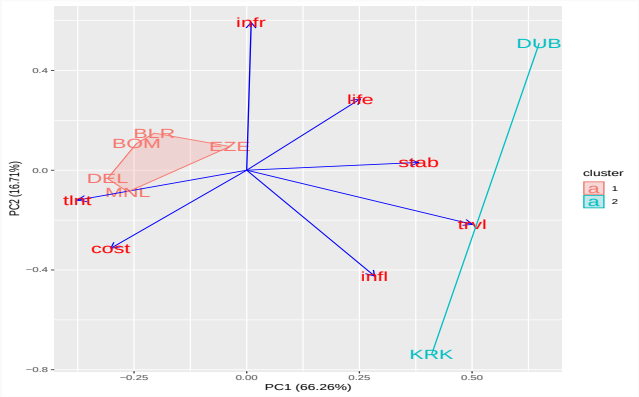


Figure 54: Autoplot with visualization of two clusters

These visualization show already a lot of information, but we can still add the “decision vector” (the vector of weights projected in the (PC_1, PC_2) plane). This shows us where the main decision weight its located, and it shows us the direction of an ideal soluton in the projection. This can be done by adding an arrow to the plot with the function `annotate()`.

```
# Use the weights as defined above:
w
## [1] 0.125 0.200 0.200 0.200 0.175 0.050 0.050

# Calculate coordinates
dv1 <- sum( w * pca1$rotation[,1])  # decision vector PC1 component
dv2 <- sum( w * pca1$rotation[,2])  # decision vector PC2 component

p <- autoplot(pam(M1,2), frame=TRUE, frame.type='norm', label=TRUE,
         shape=FALSE,
         label.colour='blue',label.face='bold', label.size=6,
         loadings=TRUE,  loadings.colour = 'dodgerblue4',
         loadings.label = TRUE, loadings.label.size = 6,
         loadings.label.colour='dodgerblue4',
         loadings.label.vjust = 1.2, loadings.label.hjust = 1.3
         )
p <- p + scale_y_continuous(breaks =
                        round(seq(from = -1, to = +1, by = 0.2), 2))
p <- p + scale_x_continuous(breaks =
                        round(seq(from = -1, to = +1, by = 0.2), 2))
p <- p + geom_segment(aes(x=0, y=0, xend=dv1, yend=dv2), size = 2,
                      arrow = arrow(length = unit(0.5, "cm")))
p <- p + ggplot2::annotate("text", x = dv1+0.2, y = dv2-0.01,
```

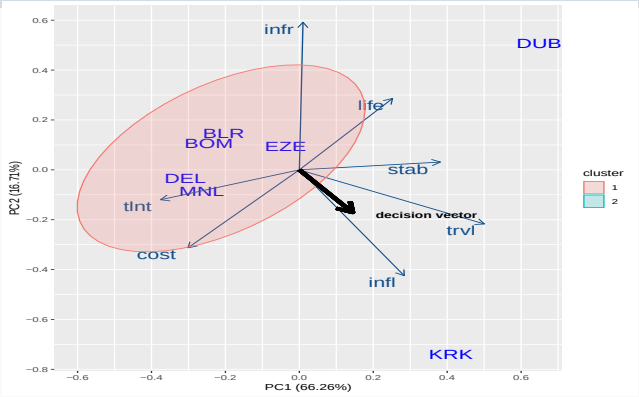


Figure 55: Clustering with elliptoid borders, labels of alternative, projections of the criteria and a "decision vector" (black arrow) – the projection of the weights – constitute a "Gaia-plot."

On plot of Figure 55 on slide 286 is an orthogonal projection in the (PC_1, PC_2) plane – the plane of the two most important principal components – we find the following information:

1. The name of the alternatives appears centred around the place where they are mapped. The projection coincides with the alternatives being spread out as much as possible

2. Two clusters are obtained by the function `pam()`: the first cluster has a red ellipsoid around it and the second one generates the error message “Too few points to calculate an ellipse” since there are only two observations in the cluster (KRK and DUB).
3. Each criterion is projected in the same plane. This shows that for example DUB offers great life quality, KRK optimal location and low wage inflation, whereas the group around DEL and MNL have low costs and a big talent pool, etc.
4. A “decision vector,” which is the projection of the vector formed by using the weights as coefficients in the base of criteria. This shows the direction of an ideal solution.

When we experiment with the number of clusters and try three clusters, then we see that KRK breaks apart from DUB. Thus we learn that – while both in Europe – Krakow and Dublin are very different places.

This plot shows us how the alternatives are different and what the selection of weights implies. In our example we notice the following.

- The cities in Asia are clustered together. These cities offer a deep talent pool with hundreds of thousands of already specialized people and are – still – cheap locations: these locations are ideal for large operations where cost is multiplied.
- Dublin offers best life quality and a stable environment. The fact that it has great infrastructure is not so clear in this plot and also note that we left out factors such as “digital enabled” for which again Dublin scores great. Ireland has also as stable low-tax regime. However, we notice that it is opposite to the dimensions “tInt” and “cost”: it is a location with high costs and a really small talent pool. This means that it would be the ideal location for a head-quarter rather than a shared service centre.

- Krakow is – just as Dublin – a class apart. Poland has a stable political environment thanks to the European Union, is close to R-bank's headquarters and further offers reasonable costs and best-in-class wage inflation. However, we note that it sits (almost) opposite to the dimension infrastructure. Krakow is indeed the ideal location for a medium sized operation, where specialization is more important than a talent pool of millions of people. It is also the ideal place for long-term plans (it has low wage inflation and a stable political situation), but still has to invest in its infrastructure. A reality check learns us that this is happening, and hence it would be a safe solution to recommend.

The idea of outranking methods is to prefer a solution that does better on more criteria. We can think of the following mechanisms:

- *Direct Ranking:* A solution *a* is preferred over *b* if *a* does better on more criteria than *b*
- *Inverse Ranking:* A solution *a* is preferred over *b* if there are more alternatives that do better than *b* than there are alternatives that do better than *a*
- *Median/Average Ranking:* Use the median/average of both previous
- *Weighted Ranking:* Use one of the previous in combination with weights w_j

Outranking in R

```
### Outrank
# M is the decision matrix (formulated for a maximum problem)
# w the weights to be used for each rank
outrank <- function (M, w)
{
  order      <- matrix(data=0, nrow=nrow(M), ncol=nrow(M))
  order.inv  <- matrix(data=0, nrow=nrow(M), ncol=nrow(M))
  order.pref <- matrix(data=0, nrow=nrow(M), ncol=nrow(M))

  for (i in 1:nrow(M)){
    for (j in 1:nrow(M)) {
      for (k in 1:ncol(M)) {
        if (M[i,k] > M[j,k]) { order[i,j] = order[i,j] + w[k] }
        if (M[j,k] > M[i,k]) { order.inv[i,j] = order.inv[i,j] + w[k] }
      }
    }
  }
  for (i in 1:nrow(M)){
    for (j in 1:nrow(M)) {
      if (order[i,j] > order[j,i]){
        order.pref[i,j] = 1
        order.pref[j,i] = 0
      }
      else if (order[i,j] < order[j,i]) {
        order.pref[i,j] = 0
        order.pref[j,i] = 1
      }
      else {
        order.pref[i,j] = 0
```

Replace $\max\{f_1(x), f_2(x), \ldots, f_n(x)\}$ by

$$\min \{y_1 + y_2 + \ldots + y_j + \ldots + y_k \,|\mathbf{x} \in \mathcal{A}\}$$

with
$$\begin{cases} f_1(x) & +y_1 & & & & = M_1 \\ f_2(x) & & +y_2 & & & = M_2 \\ \ldots & & & & & = \ldots \\ f_j(x) & & & +y_j & & = M_j \\ \ldots & & & & & = \ldots \\ f_k(x) & & & & +y_k & = M_k \end{cases}$$

- Of course, the y_i have to be additive, so have to be expressed in the same units.
- This forces us to convert them first to the same unit: e.g. introduce factors r_j that eliminate the dimensions, and then minimize $\sum_{j=1}^{k} r_j\, y_j$
- This can be solved by a numerical method.

It should be clear that the r_j play the same role as the $f_j(x)$ in the Weighted Sum Method. This means that the main argument against the Weighted Sum Method (adding things that are expressed in different units) remains valid here.[8]

The target unit that is used will typically be “a unit-less number between zero and one” or “points” (marks) . . . as it indeed looses all possible interpretation. To challenge the management, it is worth to try in the first place to present “Euro” or “Dollar” as common unit. This forces a strict reference frame.

- define a target point, **M** (e.g. the best score on all criteria)
- define a "distance" to the target point: $||\mathbf{F} - \mathbf{x}||$, with $\mathbf{F} = (f_1(x), f_2(x), \ldots, f_k(x))'$ (defined as in the Weighted Sum Method, so reducing all variables to the same units).
  For the distance measure be inspired by:
  - the Manhattan Norm: $L_1(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{k} |x_j - y_j|$
  - the Euler Norm: $L_2(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^{k} (x_j - y_j)^2\right)^{\frac{1}{2}}$
  - the general p-Norm: $L_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^{k} (x_j - y_j)^p\right)^{\frac{1}{p}}$
  - the Rawls Norm: $L_\infty(\mathbf{x}, \mathbf{y}) = \max_{j=1\ldots k} |x_j - y_j|$

The problem was introduced in Page 291 as the Manhattan norm, but we can of course use other norms too.

Advantages and Disadvantages of Goal Programming

Advantages

- Reasonably intuitive.
- Better adapted to problems of “design” (where $\mathcal{A}$ is infinite).

Disadvantages

- One has to add variables in different units, or at least reduce all different variables to unit-less variables via an arbitrary preference function.
- The choice of the weights is arbitrary.
- Even more difficult to gain insight.

PART 05: MODELLING

↓

CHAPTER 27: MULTI CRITERIA DECISION ANALYSIS (MCDA)

↓

SECTION 8:

Summary MCDA

Golden Rule

MCDA is not a science, it is an art!

The Decision-making paradox

- MCDA-methods used for solving multi-dimensional problems (for which different units of measurement are used to describe the alternatives), are not always accurate in single-dimensional problems
- When one alternative is replaced by a worse one, the ranking of the others can change
- This is proven for both ELECTRE and WPM. However, WSM and PROMethEE (most probably) are not subjected to this paradox.