Verwaltung

Hilfsseite in R-Studio: **?***befehlname* → Bsp.: **?read.csv**
CSV R-Studio: Schaltfläche *Import Dataset* oder Befehl:
name <- read.csv(*dateiname*, stringsAsFactors = TRUE/FALSE, header = TRUE/FALSE)
write.csv(*tabelle oder dataframe***, file=***"name.csv"***)** → speichert Tabelle / Dataframe
RDA R-Studio: Symbole für Laden und Speichern
load(„*filename.rda*“) → Wird automatisch importiert (Variabeln heissen automatisch so, wie sie abgespeichert wurden)
save(x,y,z**, file =** „*filename.rda*“**)** → speichert Datei 1,2,3 als *filename.rda*
save.image(file = „*filename.rda*“**)** → speichert Workspace als *filename.rda*
RDS
name **<- readRDS(***dateiname/Pfad***)** → RDS wird unter *name* ins Environment importiert
saveRDS(*objekt***, file =** „*name*“**)** → Das Objekt wird unter *name* abgespeichert
Packages R-Studio: Packages → Install (Hilfseite: **?install.packages**)
library(*Packagename*)
Datenpfad R-Studio: Default-Datenpfad ändern: Tools → Global options → Default working directory ändern
getwd() → Zeigt Datenpfad an
setwd(*Datenpfad***)** → ändert Datenpfad für Dateiablage (slash / statt backslash \ )
dir() → Zeigt Verfügbare Daten im Ordner an

Vektoren, Matrizen, Dataframes bearbeiten

Datentyp: numeric, character, factor (Namen als Stufen, alphabetisch nummeriert), logical (true/false)

- Vektor, Matrix: Überall gleicher Datentyp
- Dataframe: Pro Spalte gleicher Datentyp

Datentyp eingeben & bearbeiten

Vektor erstellen und bearbeiten

a <- c(1,2,3**)** → Datentyp numeric
b <- c(“Hans“, “Otto“, “Fritz“) → Datentyp character
c <- as.factor(“Hans“,“Otto“,“Fritz“**)** → Datentyp factor, Fritz=1, Hans=2, Otto=3
d <- c(TRUE, FALSE, TRUE**)**
e <- vector(“numeric”,*5***)** → Leerer Nullvektor (0 0 0 0 0)
f <- rnorm(*10***)** → 10 Zufallszahlen [Aus Normalverteilung um 0 mit Standartabw. 1]
class(*a***)** → Zeigt Datentyp des Vektors *a* an
as.character(*a***)** → Vektor a hat nun Datentyp character
as.numeric(*c***)** → Vektor c hat nun Datentyp numeric. Fritz = 1, Hans = 2, Otto = 3
as.numeric(*d***)** → Vektor d hat nun Datentyp numeric. TRUE = 1, FALSE = 0

a[3] → zeigt drittes Element an
a[c(1,3)] → zeigt erstes und drittes Element an
a[3] <- 2 → Ersetzt drittes Element durch den Wert 2
a[c(TRUE,FALSE,FALSE,TRUE**)]** → Zeigt 1. & 4. Element von a an (überall wo TRUE steht)

sum(a**)** → Summiert alle Werte (auch bei Matrix möglich)
mean(a**)** → Zeigt Mittelwert aller Werte an (auch bei Matrix möglich)
min(a**) , max(**a**)** → Zeigt kleinsten bzw. grössten Wert an
which.min , which.max(a**)** → Zeigt an, der wievielte Wert der kleinste/grösste ist

Matrix erstellen und bearbeiten

m <- rbind(vektorA, c(5,8,3)) → Matrix mit Zeile 1 = vektorA und Zeile 2 = Vektor (5,8,3) !alle Vektoren müssen gleich lang sein!

m <- matrix(c(1,2,3,4,5,6) ,2,3) → Matrix mit Werten 1-6 in 2 Zeilen & 3 Spalten
Werte aus *Vektor* werden Spaltenweise eingesetzt. Verdeutlichung:
a <- c(1,2,3,4,5,6) m <- matrix(data=a, 3, 2)
m → output:

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Dataframe

df <- data.frame(*Name***=** *vektorA*, *Preis* = *vektorB*, stringsasfactors = TRUE) → Dataframe mit Spalte 1 = *vektorA* und Spalte 2 = *vektorB*. *Name* und *Preis* sind die Spaltennamen

m[2,3] → zeigt Element in zweiter Zeile und dritter Spalte an
m[c(1,3),c(2,3)] → Zeigt von der 1.+3. Zeile jeweils die 2.+3. Spalte an
m[2,] → Zeigt gesamte zweite Zeile an
m[,2] → Zeigt gesamte zweite Spalte an
m[,“name“] → Zeigt gesamte Spalte mit dem Spaltentitel *name* an
m$*name* → Zeigt gesamte Spalte mit dem Spaltentitel *name* an

m <- m[-2,] → Dritte Zeile von *m* wird gelöscht
cor(m) → Korrelationsmatrix von *m* wird angezeigt

sum(df**[,**1**])** → Summe aller Werte von Spalte 1 vom Dataframe *df*
sum(df**$***name***)** → Summe aller Werte von Spalte *name* vom Dataframe *df*
mean(df[,1]**)** → Mittelwert aller Werte von Spalte 1 vom Dataframe *df*
mean(df$y[df$x == ‘rot‘**])** → Mittelwert aller Werte von Spalte y bei denen in Spalte x “rot” steht
mean(df$y[df**[,**2**] <** 3**])** → Mittelwert aller Werte von Spalte y bei denen in Spalte 2 der Wert kleiner oder gleich 3 ist

datei$*name* → Zeigt Spalte mit Spaltennamen *name* an
head(*df***)** → Zeigt erste paar Zeilen inkl. Titel an
colnames(*df***)** → Zeigt Spaltennamen an
dim(*df***)** → Zeigt Anzahl Zeilen und Spalten an
nrow(*df***) , ncol(***df***)** → Zeigt Anzahl Zeilen bzw. Anzahl Spaltenan
attach(*df***)** → Zeigt Anzahl Spalten an (als Variable)
summary(*df***)** → Zeigt Zusammenfassung von jeder Spalte an
str(*df***)** → Zeigt Struktur von *df* an (Spaltenname und Inhalt)
plot(y **~** *a* + *b*, **data =** *df***)** → Zeigt Streudiagramm an

df <- as.data.frame(*tab***)** → Wandelt die Tabelle *tab* in ein Dataframe um
tab <- xtabs(*y* **~ . , data =** *df***)** → Wandelt Dataframe *df* in eine Tabelle um
tab <- xtabs(y~*a*+*b***, data=***df***)** →Wandelt Teil Teil des Dataframe *df* in Tabelle um

Logischer Vektor

Logische Verknüfpungen:
< kleiner als > grösser als & und-Verknüpfung von zwei Bedingungen
%in% *[vgl. Beispiel]*
Beispiele zum besseren Verständnis
jg <- (*personen***[,**“*Jahrgang*“**] <** *1990***)** → Spalte *Jahrgang* des Dataframes *personen* wird mit *1990* verglichen → *jg* ist ein logischer Vektor mit TRUE überall dort wo der *Jahrgang* < 1990 ist
personen[jg,] → Es werden nur die Zeilen vom Dataframe *personen* angezeigt, bei welchen der Vektor *jg* TRUE ist (= alle, die älter sind als Jahrgang 1990)

stud <- (personen**[,**“*Fach*“**]** **%in%** **c(**“*Bio*“,“*Chemie*“**))** → Spalte *Fach* des Dataframes *personen* wird mit *Bio* und *Chemie* verglichen → *stud* ist ein logischer Vektor mit TRUE überall dort wo das Fach *Bio* oder *Chemie* ist
personen[stud,] → Es werden diejenigen Zeilen vom Dataframe *personen* angezeigt, bei welchen der Vektor *stud* TRUE ist (= alle diejenigen, die Bio oder Chemie studieren
jgstud <- ((personen**[,**“Fach“**]** **%in%** **c(**“Bio“,“Chemie“) **&** (personen**[,“**Jahrgang**“] <** 1990**))** → *jgstud* ist ein logischer Vektor der TRUE hat wo das Fach *Bio* oder *Chemie* ist und der *Jahrgang* <1990
personen[jgstud,] → Es werden diejenigen Zeilen vom Dataframe *personen* angezeigt, bei welchen der Vektor *jgstud* TRUE ist (= Alle die Bio oder Chemie studieren und älter als 1990 sind)

Graphiken

plot(x,y) → Zeigt Graphik von x und y an.
plot(x, y, **main** =“*Titel*“, **xlab =** “*X-Achse*“, **ylab =** “*Y-Achse*“, **type =** “l“**)** → Beschriftungen eingestellt. type = “l“ = linie, type = “b“ = Punkt+Linie
lines (x = c(*2,8*), y = c(*15,5*)**)** → Linie einfügen von x=2 bis x=8 und von y=15 bis y=5
boxplot(*x***~***y*, **data=***dat***)** → Zeigt Boxplot an
par(mfrow=c(1,3**))** → Die *3* nächsten Plots werden nebeneinander dargestellt

Listen

mylist <- list(a=4, b=vec1, c=m, d=“hallo”**)** → In Listen können beliebige Elemente eingetragen werden (Zahlen, Matrizen, Text,…). a,b,c,d = Titel der Elemente

mylist$*b* → Zeigt Element mit dem Titel b an, hier ist b = vec1
mylist[[*3***]]** → Zeigt das dritte Element an, hier ist drittes Element c (es braucht zwei Klammern!)

```
> mylist$b
[1] 1 2 3
>
> mylist[[3]]
     [,1] [,2] [,3] [,4]
[1,]    5    7    2    4
[2,]   12    7   -3   48
[3,]    2    0   98  -12
```

for-Schleife

for(i in 1:*10***) {** → In diesem Beispiel wird die Schlaufe 10 mal durchlaufen
befehl → *befehl* wird 10x wiederholt. Man kann Variable i im Befehl benutze
} → Start und Ende der Schleife mit geschweifter Klammer kennzeichne

- Statt einer Zahl darf für die #Schlaufen auch eine Variable eingesetzt werden
- Es dürfen auch mehrere for-Schlaufen verschachtelt werden
- Sämtliche Variabeln, Vektoren,… müssen vor der Schlaufe erstellt werden

Beispiel:
reps <- vector(“numeric”,*8***)** → *reps wird definiert als Nullvektor mit 8 Stellen*
n <- *8* → *n gibt die Anzahl Wiederholungen der Schlaufe an*

```
for(i in 1:n) {
    reps[i] <- i
}
```

reps → *reps ist ein Vektor der die Schlaufen mitzählt: (1 2 3 4 5 6 7 8)*

falls: B = 1.2 und y=loga+b*logx, b?, simply: b=B
falls logb steht: B=logb => exp(B)=b (umformen)

Funktion

fkt1 <- function(*var1 = 10, var2 = 1***) {**
befehl 1
befehl 2
}

Vorhersagen für neue Daten:
> newP <- data.frame(x1 = 0.45, x2 = 1.54) #oder mehr Variablen
> predict.lm(fit, newdata = newP)
Vetrauens- & Vorhersageintervalle für vorhergesagte Werte:
> predict.lm(fit, newdata = newP, interval = "confidence", level = 0.9149)
> predict.lm(fit, newdata = newP, interval = "prediction", level = 0.9476)

- fkt1 = Name der Funktion
- (var1=10, var2=1) → Default von allen Variablen der Funktion definieren
- Man kann beliebig viele Befehle sowie for-Schlaufen einbauen
- Bei Funktionen wird immer die letzte Zeile ausgegeben, [hier: *befehl 2*]

➢ Funktionen nach erstellen immer einmal ausführen, danach kennt R die Funktion und kann sie normal ausführen

fkt1 → Gibt Funktions-Beschreibung aus
fkt1() → Gibt Funktion mit den Default-Werten aus
fkt(var1 = *5***, var2 =** *0.5***)** → Gibt Funktion mit Werten var1 = *5* und var2 = *0.5* aus
fkt(*5, 0.5***)** → Gibt Funktion mit den Werten var1 = 5 und var2 = 0.5 aus

Multiple lineare Regression

Modell: $Y_i = \beta_0 + \beta_1 * x_{i,1} + \beta_2 * x_{i,2} + \ldots + \beta_{p-1} * x_{i,p-1} + \varepsilon_i$; $\varepsilon_i \sim N(0,\sigma^2)$ iid

Falls sich x_1 um eine Einheit erhöht und alle anderen x_i gleich bleiben, erhöht sich y um β_1.

Multiple R-Squared

$R^2 = 1 - \frac{SS_{err}}{SS_{tot}}$

R^2 = %Anteil der beobachteten Varianz, der durch das Modell erklärt wird. Je näher R^2 an 1 liegt, desto besser passen die Werte. (Oft sind Werte für β_i gross, wenn R^2 gross ist). Je grösser R^2 desto relevanter ist das Ergebnis. Es kann sein, dass das Ergebnis signifikant (kleiner P-Wert) aber nicht relevant (kleiner R-Wert) ist.

lineare Regression plotten im R-Studio

fit <- lm(*y ~ x₁ + x₂ + x₃***, data =** *dat***)** → lineare Regression
fit <- lm(log(*y***) ~** *x₁ + x₂ + x₃***, data =** *dat***)** → mit exponentiellem Zusammenhang
fit <- lm(log(*y***) ~ log(***x₁***) + log(***x₂***) + log(***x₃***), data =** *tab***)** → polynomiell
summary(fit) → gibt den output der linearen Regression aus Output:

coefficients

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	β_0	$\sigma(\beta_0)$	t-Werte	p-Werte
x_1	β_1	$\sigma(\beta_1)$	für	für
x_2	β_2	$\sigma(\beta_2)$	$\beta_0, \beta_1, \beta_2, \beta_3$	$\beta_0, \beta_1, \beta_2, \beta_3$
x_3	β_3	$\sigma(\beta_3)$		

Residual standart error: *5,914* on *270* degrees of freedom
Multiple R-squared: *0.8115* **Adjusted R-squared:** *0.8108*
F-statistic: *1162* on *1* and *270* DF, p-value < *2,2 e-16*
[p-value = p-Wert vom F-Test]

Interpretation Output:

- # Freiheitsgrade = # Messungen - # β's
- t-value = estimate/std.error
- Zweiseitiger Verwerfungsbereich von $\beta_i = (-\infty, -t_{n-1,\,1-\alpha/2}] \cup [t_{n-1,\,1-\alpha/2}, \infty)$
- 95%-V.I. von $\beta_i = \beta_i \pm 2 * \text{std.error}$
- $\varepsilon_i \sim N(0,(\text{Residual standart error})^2)$ iid

Modelle für lineare, exponentielle und polynomielle Zusammenhänge

linear: y ~ x	exponentiell: log(y) ~ x	polynomiell: log(y) ~ log(x)
Modell: y = a + b*x + ε	Modell: log(y) = c + d*x +ε *c = log(a)*	Modell: log(y) = e + f*log(x) +ε *e = log(a)*
→ **plot(x,y)** ergibt eine Gerade	→ **plot(log(y),x)** ergibt eine Gerade	→ **plot(log(y),log(x))** ergibt eine Gerade

Vertrauensintervalle und Vorhersageintervalle

confint(*fit***)** → Gibt 95% Vertrauensintervalle von *fit* aus
confint(*fit***,level =** *0.99***)** → Gibt 99% Vertrauensintervalle von *fit* aus

Vorhersage für konkrete Werte

fit **<- lm(y ~ age + g, data = dat)** → Lineare Regression fitten
newdat **<- data.frame(age =** *25***, g = "***Female***")** → Man muss die Werte definieren, für die man eine Vorhersage machen will. Hier: Alter = 25, Geschlecht = weiblich.
predict.lm(*fit***, newdata =** *newdat***)** → Erwarteter Wert für Werte aus *newdat*
predict.lm(*fit***, newdata =** *newdat***, interval = „confidence", level = 0.90)** → Gibt ein 90% Vertrauensintervall für den erwarteten Wert für die Werte aus *newdat* an
predict.lm(*fit***, newdata =** *newdat***, interval = „prediction", level = 0.95)** → → Gibt ein 95% Vorhersageintervall für den erwarteten Wert für die Werte aus *newdat* an

F-Test

(F-value > t-value (mächtiger))

$H_0: \beta_1 = \beta_2 = \ldots = \beta_p = 0$; H_A: mind. ein $\beta_i \neq 0$

Wenn F-Test signifikant, kann man bei t-Tests die signifikanten Variabeln suchen

Kollinearität

Wenn zwei β's stark korrelieren kann es sein, dass der F-Test signifikant ist, aber keiner der Parameter signifikant ist. In diesem Fall muss man einen der beiden korrelierten Parameter weglassen.
Herausfinden, ob zwei Parameter x und y stark korrelieren:
cor(x,y**)** → Je näher an 1 desto stärker korreliert

Residuenanalyse

plot(*fit***, which = c(1:2))** → ohne *which* werden 4 Plots angezeigt

Faktoren als erklärende Variabeln

Faktoren: Haarfarbe, Alter, Geschlecht, Hobbies,…
Level: Werte, die der Faktor annehmen kann

Zwei Levels

$x_i = 0$ für männlich, $x_i = 1$ für weiblich
Bsp: Körpergrösse = $\beta_0 + \beta_1 * x_i + \beta_2 * \text{Alter}_i + \varepsilon_i$

mehr als zwei Levels

1 Referenzlevel, 1 binäre Variable für jedes andere Level → R-Studio regelt das
Informationen holen:
str(*datei***)** → zeigt Zusammenfassung von jeder Spalte an
dim(*datei***)** → Zeigt Anzahl Zeilen und Spalten an
levels(*datei$Gender***)** → "Male", "Female"; gibt alle Level dieser Spalte an.
datei$*Gender* **<- relevel(***datei$Gender***, ref="***Female***")** → Referenz ändern auf *Female*

Wechselwirkungen WW

WW zwischen zwei oder mehr Variablen (z.B. Alter und Vermögen) hat je nach Geschlecht unterschiedlichen Einfluss = **Steigungen können unterschiedlich sein** je nach Geschlecht

Wählen eines Modelles: mit oder ohne WW?

Man schaut bei der linearen Regression mit WW, ob die WW signifikant ist. In unserem Beispiel unten ist die WW *Age:GenderFemale* → Wenn *Age:GenderFemale* signifikant ist, wählen wir die Variante mit WW!

Modell

fit <- lm(income ~ Age * Gender, data = *datei***)** → Lineare Regression mit WW
fit <- lm(income ~ Age + Gender + Age:Gender, data = *datei***)** → = wie oben

output: summary(fit)
Intercept a = Achsenabschnitt für *Male* (Referenzlevel)
Age b = Steigung für *Male* (Referenzlevel)
GenderFemale c = **Änderung** Achsenabs. für *Female* (Achsenabs.= a + c)
Age:GenderFemale d = **Änderung** Steigung für *Female* (Steigung= b + d)
Modell: Income = $(a + c*\text{Gender}_i) + (b + d*\text{Gender}_i) * \text{Age} + \varepsilon_i$
$\varepsilon_i \sim N(0,(\text{residual std. error})^2)$, $\text{Gender}_i = 0$ Referenzlevel (*Male*), $\text{Gender}_i = 1$ *Female*

plot(*fit1***, which = c(1,2))** → Residuenanalyse

MSE – Mean Squared Error

$MSE = \frac{1}{n} * \sum_{i=1}^{n} R_i^2 = \frac{1}{n} * RSS$ → Root mean squared error = $\sqrt{MSE}$

Trainings-MSE vs. Test-MSE

Trainings-MSE: f(x) möglichst gut an vorhandene Messwerte anpassen. Trainings-MSE kann beliebig klein gemacht werden bei genügend vielen Parametern.
Test-MSE: Neue Messwerte dazugeben und schauen, wie gut f(x) diese Werte voraussagt. MSE davon = Test-MSE. Anm.: Selbst bei perfektem f(x) ist TestMSE > 0 da es in der Realität immer auch einen zufälligen Fehler (ε) in den Daten hat.

Der minimale Trainings-MSE ist nicht immer der beste ist für die zukünftigen Variablen (→ Test-MSE). Grund: bei einem perfekten f(x) modelliert man den zufälligen Fehler ε mit und dieser zufällige Fehler ε ist im Test-MSE anders.

Test-MSE Schätzen: Variante 1

Prinzip: Daten aufteilen in einen Test- und einen Trainingssatz. Das Modell f(x) auf den Trainingssatz schätzen und auf den Testsatz evaluieren

+ einfach, schnell
- Je nach Wahl des Testsatzes erhält man unterschiedliche Test-MSE
- Trainingssatz ist kleiner als urspr. Datensatz: f(x) basiert auf weniger Daten

R-Studio: Trainings-MSE:

n **<- nrow(***dat***)**

train **<- sample(***n*,*90***)** → aus Zahlenwerten *n* werden *90* Zufallswerte ausgewählt

fit **<- lm(***y* **~ . , data =** *dat***, subset =** *train***)** → **Lineare Regression** mit y abhängig von allen anderen Variablen ("." = alle anderen Variablen) mit denjenigen Werten aus der Datei *dat*, die durch die 90 Zufallswerte von *train* bestimmt werden.

fit0 **<- lm(***y* **~** *1***, data =** *dat,* **subset =** *train***)** → Bsp. für Modell mit Grad 0

fit3 **<- lm(***y* **~** *x1* **+** *x2* **+** *x3***, data =** *dat,* **subset =** *train***)** → Bsp. Grad 3

summary(*fit***)** → Infos zur Regression *fit*

sum(*fit***$residuals^2)** → RSS

mean(*fit***$residuals^2)** → Trainings-MSE

R-Studio: Test-MSE:

yHut **<- predict(***fit***,** *dat***)** → Macht Vorhersage für alle Datenpunkte aus *dat*

quadratResid **<- (***dat***$***y***, - *yHut*)^2** → Residuenquadrat für alle Datenpunkte aus *dat*

quadratResidTest **<-** *quadratResid***[-***train***]** → Alle Datenpunkte aus Test-Set

TestMSE **<- mean(***quadratResidTest***)** → Test-MSE

TestRMSE **<- sqrt(**TestMSE**)** → Root Squared Error vom Test-MSE

Test-MSE Schätzen: Variante 2 = Cross-Validation

Single-line

Prinzip: Jede Zeile ist einmal Testset, der Rest ist Trainingsset

\+ Jeder Durchgang liefert dasselbe Resultat

\- Sehr langsam da es ein Fit pro Zeile gibt

K-Fold

Prinzip: Teile die Daten in *k* Blöcke, jeder Block ist einmal Testset (10-Fold = 10 Blöcke)

\+ viel schneller da man nicht für jede Zeile einen Fit machen muss

\+ Je nach Unterteilung in *k* Blöcke erhält man bei mehreren Durchgängen immer anderen Test-MSE (da Unterteilung in Blöcke willkürlich ist)

R-Studio: Single-Line Methode

fitVoll **<- glm(***y* **~ . , data =** *dat***)** → glm() ≈ lm()

cv.err **<- cv.glm(data =** *dat***, glmfit =** *fitVoll***)**

sqrt(*cv.err***$delta)** → Erste Zahl ist der Test-MSE

R-Studio: 10-fold

fitVoll **<- glm(***y* **~ . , data =** *dat***)** → glm() ≈ lm()

cv.err10 **<- cv.glm(data =** *dat***, glmfit =** *fitVoll***, K =** *10***)**

sqrt(cv.err10**$delta)** → Erste Zahl ist der Test-MSE

Test-MSE Schätzen: Indirekte Variante

Prinzip: Kriterium K verbindet RSS und Anzahl Parameter d → K = RSS + f(d)

Modell mit bestem K ist optimal (unter bestimmten Annahmen). K kann verschiedene Formen annehmen, im Paket „leaps" ist K = BIC = $\frac{1}{n}$*(*RSS* + log(n)*d*$\hat{\sigma}^2$). **Ziel**: BIC soll möglichst klein sein

\+ Schnell zu rechnen

\- Approximativ, trifft Annahmen, Test-MSE wird nicht berechnet

Praxis: Anwenden, falls viele oder komplizierte Modelle geschätzt werden.

Modellwahl mit BIC

➔ Finde ein Modell mit möglichst kleinem Vorhersagefehler

➔ Finde diejenigen Variablen, die für eine gute Vorhersage nötig sind

Faustregel: Mache Test-MSE mit Cross Validation und die Modellwahl mit dem BIC

Best Subset Prinzip: Berechne eine lineare Regression für alle möglichen Kombinationen von erklärenden Variablen, speichere BIC

\+ findet das beste Subset

\- Rechenaufwand ist riesig: *p* Variablen → 2^p Subsets

Stepwise forward Prinzip: Starte mit einem leeren Modell und füge eine Variable nach der anderen hinzu. Stopp wenn BIC nicht mehr kleiner wird

Beispiel: y mit x1, x2, x3, x4, …. beschreiben

M1: y ~ 1; BIC = 20 → M2: y ~ x1; BIC = 17 → M3: y ~ x1 + x2; BIC = 18

BIC bei M3 ist schlechter als bei M2 → Stopp und wähle Variante M2

Stepwise backward

Prinzip: Starte mit einem vollen Modell und lasse eine Variable nach der anderen weg. Stoppe, sobald BIC nicht mehr kleiner wird.

R-Studio: Modellwahl

library(leaps) → Package *leaps* wird benötigt

m **<- regsubsets(***y* **~. , data =** *dat***, method = "***exhaustive***", nvmax =** *20***)**

→ oder method = "*forward*" oder "*backward*". nvmax gibt maximale Setgrösse = maximale #Variablen an (für weniger Rechenaufwand)

summary(*m***)**

Ausgabe: Spalten = Variablen, Zeilen = #Variablen, Stern * zeigt immer die beste Kombination von Variablen für diese #Variablen an

Beispiel (Ausschnitt aus der Tabelle [nvmax = 5]):

```
1 subsets of each size up to 5
Selection Algorithm: exhaustive
          x1  x2  x3  x4  x5  x6  x7  x8  x9  x10 x11 x12 x13
1  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " " "
2  ( 1 ) " " " " " " " " " " " " " " " " " " "*" " " " " " "
3  ( 1 ) " " " " " " " " " " " " " " " " " " "*" " " " " " "
4  ( 1 ) " " " " " " " " " " " " " " " " " " "*" " " " " " "
5  ( 1 ) "*" " " " " " " " " " " " " " " " " "*" " " " " " "
          x14 x15 x16 x17 x18 x19 x20 x21 x22 x23 x24 x25 x26
1  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " "
2  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " "
3  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " "
4  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " "
5  ( 1 ) " " " " " " " " " " " " " " " " " " " " " " " " " "
```

Interpretation: Die erste Zeile 1 zeigt an, dass wenn nur eine Variable gewählt wird die Variable x20 die beste ist. In der zweiten Zeile 2 sieht man, dass bei 2 Variablen die Variablen x20 & x10 am besten sind. usw.

ms **<- summary(**m**)** → *ms* speichert beste Modelle für vorgegebene #Variablen

*ms***$bic** → Gibt die BIC-Werte für die Modelle mit 1,2,3,… Variablen aus

ncoef **<- which.min(***ms***$bic)** → Zeigt welches Modell mit welcher #Variablen die beste Vorhersage hat

coef(m,ncoef**)** → Zeigt die Variablen inkl. Werte für das beste Modell an

Bsp: *bei coef kommen die Variablen x3 und x5 heraus. Dann macht man so weiter:*

fitBest **<- glm(**y **~** x3 **+** x5**, data =** dat**)** → Das ist das Beste Modell!

cv.errBest **<- cv.glm(data =** *dat***, glmfit =** *fitBest***)**

sqrt(*cv.errBest***$delta)** → erste Zahl = Test-MSE

GLM – Generalisiertes Lineares Modell

Parameter einer Verteilung hängt von erklärenden Variablen ab.

Bsp.: lineare Regression, logistische Regression, ANOVA,…

Logistische Regression

Modell

$\log(\frac{P(x)}{1-P(x)}) = y = \beta_0 + \beta_1 * x$ y ~ Bin(n, P(x))

➔ y ist nicht die W'keit, sondern die log-odds der W'keit! Vorhersagen über die Änderung der W'keit sind nicht einfach so möglich, aber für konkrete Werte können odds und W'keit problemlos berechnet werden.

- Wenn x um eine Einheit erhöht wird, erhöhen sich die log-odds um +β_1 (additiv!
- Wenn x um eine Einheit erhöht wird, erhöhen sich odds um *e^{β_1} (multiplikativ!)
- 95%-V.I. der log-odds = [β_1 ± 2*std.err(β_1)] →*std.err mit summary(fit) bestimme*
- 95%-VI der odds = $e^{[\beta_1 \pm 2*std.err(\beta_1)]}$ → *std. err mit summary(fit) bestimmen*

odds und log-odds

odds(A) = $\frac{P(A)}{1-P(A)}$ → P(A) = $\frac{odds(A)}{1+odds(A)}$ *odds ∈ [0 , ∞)*

log-odds(A) = log($\frac{P(A)}{1-P(A)}$) *log-odds ∈ (-∞ , ∞)*

Odds sowie log-odds wachsen monoton mit der W'keit P. Das heisst, wenn $y = \beta_0 + \beta_1 * x$ grösser wird, wird auch die W'keit P grösser. Y selber bezeichnet aber nicht die W'keit, sondern die log-odds der W'keit.

Wahrscheinlichkeit berechnen (von Hand)

log-odds = log($\frac{P(A)}{1-P(A)}$) = $\beta_0 + \beta_1 * x$ | odds = $e^{\text{log-odds}}$

$\text{odds}(A) = \frac{P(A)}{1-P(A)} = e^{\beta_0 + \beta_1 * x}$ | wir wissen, dass P(A) = $\frac{odds(A)}{1+odds(A)}$

P(A) = $\frac{e^{\beta_0 + \beta_1 * x}}{1+e^{\beta_0 + \beta_1 * x}}$

odds(x) = exp(log-odds(x))
probs(x) = odds(x) / (1 + odds(x))
class(x) = probs(x) ≥ 0.5 = odds(x) ≥ 1 = log-odds(x) ≥ 0
Wenn man class wechseln will (zB krank auf gesund), einfach Wert mal (-1) machen.

Rechenbeispiel:

gegeben: β_0 = -10; β_1 = 0.01, x = 1'000

log-odds = -10 + 0.01 * 1'000 = -10 + 10 = 0

odds = $e^{\text{log-odds}} = e^0 = 1$

P(A) = $\frac{odds}{1+odds} = \frac{1}{1+1} = \underline{\underline{0.5}}$

Vorhersagen (log-odds) für neue Daten:
> predict.glm(fit, newdata = dat1, type = "link")
Vorhersagen (Wahrscheinlichkeiten) für neue Daten:
> predict.glm(fit, newdata = dat1, type = "response")
Vorhersagen (Klasse) für neue Daten:
> predict.glm(fit, newdata = dat1, type = "response") >= 0.5

R-Studio

Logistische Regression machen

fit **<- glm(***y* **~** *x* **+** *g***, data =** *dat***, family = binomial)**

log-odds, Wahrscheinlichkeiten

newdata: Vor der Bestimmung der W'keit müssen neue Daten gegeben werden.

- new **<- data.frame(***x* **=** *5***,** *g* **=** *„w"***)** →Erstelle neues Df mit allen Variablen
- nimm bestehenden Datensatz, z.B. **dat[42,]** = Werte der Zeile 42 aus *dat*

logodds **<- predict.glm(***fit***, newdata=***new***, type ="link", se.fit=TRUE)**

p **<- predict.glm(***fit***, newdata=dat[***42,***] , type="response", se.fit=TRUE)**

*p***$fit/(1-***p***$fit)** → odds(y) \→ p = W'keit P(y)

Bsp: den 101.ten betrachten: predict.glm(fit.x, newdata = dat0[101,], type = "response") können Vorhersagen der Wahrscheinlichkeiten gemacht werden

Klassifikation

fit **<- glm(y ~ x, data=***dat0***, family=binomial)** → *Regression auf Übungsdaten*
pp **<- predict(***fit***, newdata =** *dat1***, type = "response")**
→ *W'keitsvorhersage für den kompletten Testdatensatz statt nur für einen einzigen Wert*
class **<- (pp >= 0.5)** → *Für alle Zeilen, bei denen pp ≥ 0.5 ist steht beim Vektor class TRUE, für alle Zeilen bei denen pp < 0.5 ist steht FALSE*
classn **<- as.numeric(***class***)** → *classn ist ein Vektor mit TRUE = 1, FALSE = 0*
sum(*classn* **==** *dat1***$***y***)** → Gibt **Anzahl** Messwerte = Zeilen an, bei denen die Voraussage durch die Regression das *richtige* Resultat geliefert hat

Vorhersagefehler mit Cross-Validation

k <- 10
n <- nrow(*dat0***)** → *dat0* = Trainingsdatensatz
folds <- sample(1:k, n, replace = TRUE)
cv.pp <- rep(NA,n) → [*pp* haben wir weiter oben definiert]
for (i in 1:k) {
mod <- glm(*y* **~** *x***, data =** *dat0***[folds != i,], family = binomial)** → Modell wird mit allen Blöcken ausser Block i geschätzt
idx <- which(folds == i) → Zeilen im Datensatz, die zu Block i gehören
cv.pp[idx] <- predict(mod, newdata = *dat0***[folds == i,], type = "response")**
→ Vohersage für Block i direkt in cv.pp schreiben
}
cv.class <- (cv.pp > 0.5)
table(cv.class, dat0$y)→ output ist eine Tabelle:

```
> table(cv.class, dat1$y)

cv.class  0  1
   FALSE 50 10
   TRUE   8 81
```

Fehlerrate = $\frac{falsch\ Positive + falsch\ Negative}{Alle} = \frac{8+10}{50+10+8+81} = \frac{18}{149} = 0.12$

Anm: Klassifikation bei mehr als 2 Klassen: Erweiterung der Logistischen Regression („one vs. all") oder z.B. Random forest, lineare Diskriminanzanalyse, nearest neigbour Methode,….

Simpson-Paradox

Wenn man eine zweite Variable einfügt, ändert sich die Steigung β_1 für die erste Variable. Falls sich die Steigung extrem stark ändert (vielleicht sogar von negativer Steigung zu positiver), kann das sehr paradox wirken. Das Simpson – Paradox kann natürlich auch dann auftreten, wenn man eine dritte/vierte/fünfte/… Variable einführt.
Beispiel: Studenten und Schulden: Studenten haben oft hohe Schulden und deshalb ein hohes Risiko für Pleite. *Aber* wenn zwei Leute hohe Schulden haben ist der nicht-Student schneller Pleite.

Mixed Effects Models

Wiederholte Messungen (z.B. Wachstumskurven)

Block Effects

Modell: $y_{ij} = (\beta_0 + \beta_{0,i}) + \beta_1 * x_j + \varepsilon_{ij}$ $\varepsilon_{ij} \sim N(0, \sigma^2)$ iid

Block Effekte sind eigentlich lineare Regressionen mit oder ohne WW.

y = Zielgrösse = Kraft, Gewinn, Grösse,…
i = z.B. Person (Alex, Philip, Silvia,…), Berge (Jungfrau, Eiger, Mönch,…)
j = z.B. Zeitdauer

- β's sind fixe Effekte = konkrete Werte
- β_0 = Achsenabs., $\beta_{0,i}$ = Änderung Achsenabs. je nach Individuum, β_1 = Steigung
- Dieses Modell erlaubt **Aussagen über Individuen.** (nicht über Populationen)

Mixed Effects

- Fixed effects + random effects = mixed effects
- Dieses Modell erlaubt **Aussagen über die ganze Population** (nicht zu Individuen)

Random Intercept RI

Modell: $y_{ij} = (\beta_0 + u_i) + \beta_1 * x_j + \varepsilon_{ij}$ $\varepsilon_{ij} \sim N(0, \sigma^2)$ iid, $u_i \sim N(0, \sigma_u^2)$ iid

- β's sind fixe Effekte = konkrete Werte
- u_i =zufälliger Effekt =kein konkreter Wert, folgt einer Verteilung (oft Normalverteilung)

Random Slope Random Intercept RIRS

Modell: $y_{ij} = (\beta_0 + u_{1,i}) + (\beta_1 + u_{2,i}) * x_j + \varepsilon_{ij}$
$\varepsilon_{ij} \sim N(0, \sigma^2)$ iid $u_{1,i} \sim N(0, \sigma_1^2)$ iid $u_{2,i} \sim N(0, \sigma_2^2)$ iid cor(u_1, u_2) = ρ

Schätzen von Mixed Effects Models

Maximum Likelihood ML

- Varianzschätzungen („u_i" im Modell oben) haben Bias
+ Tests zwischen Modellen mit verschiedenen fixen Effekten sind möglich

Restricted Maximum Likelihood REML

- Kann nur Modelle mit gleichen fixen Effekten vergleichen
+ Varianzschätzugen haben keine Bias

→ Ist die default-Einstellung in R

R-Studio

library(lme4), library(lmerTest), library(lattice)

RIRS und RI

fit **<- lmer(***y* **~** *x* **+ (***x***|***subject***) , data =** *dat***)**
→ **RIRS Modell**: *(x|subject)* = Zufällige Schwankung in Achsenabschnitt und Steigung pro Person
→ **RI Modell**: *(1|subject)* = Zufällige Schwankung in Achsenabschnitt aber keine Schwankung in der Steigung

summary(*fit***)** → Output (Auszug aus dem wichtigen Teil des Outputs):

```
Random effects:
 Groups   Name        Variance Std.Dev. Corr
 Subject  (Intercept) 65.31    8.082
          x           1.94    1.393    -0.41
 Residual             12.07    3.474
Number of obs: 30, groups:  R, 6

Fixed effects:
            Estimate Std. Error      df t value Pr(>|t|)
(Intercept)  44.6333     3.4773  5.0000   12.84 5.11e-05 ***
x            20.0667     0.7242  5.0000   27.71 1.15e-06 ***
```

→ **Modell aufgrund dieses Outputes wäre**:

$$y_{ij} = (\beta_0 + u_{1,i}) + (\beta_1 + u_{2,i}) * x_j + \varepsilon_{ij}$$
$$y_{ij} = (44.63 + u_{1,i}) + (20.07 + u_{2,i}) * x_j + \varepsilon_{ij}$$

$u_{1,i} \sim N(0, 8.082^2)$, $u_{2,i} \sim N(0, 1.393^2)$, $\varepsilon_{ij} \sim N(0, 3.474^2)$ iid, cor($u_{1,i}$, $u_{1,i}$) = ρ = -0.41

Residuenanalyse

plot(*fit***)** → Tukey-Anscombe Plot
qqnorm(residuals(*fit***))** → QQ-Plot der Residuen
qq **<- ranef(***fit***)** →Zufällige Achsenabschnitt und Steigung pro Person (Tabelle)
qqnorm(*qq***$***Subject***[,1])** → QQ-Plot von zufälligen Achsenabschnitten pro Person
qqnorm(*qq***$***Subject***[,2])** → QQ-Plot von zufälligen Steigungen pro Person

```
> ranef(fitRIRS)
$Subject
  (Intercept)           x
1    8.695172 -0.02236134
2   -6.951618 -1.28619303
3    8.953713 -0.81581613
4    2.614770 -0.23626972
```

Modell zu Person 3 (β_0, β_1 von summary(fit) oben):
$y_{ij} = (44.63 + 8.95) + (20.07 + (-0.82)) * x_j + \varepsilon_{ij}$

RI oder RIRS?

fitRIRS **<- lmer(***y* **~** *x* **+ (***x***|***subject***) , data =** *dat***)** → Erstelle RIRS Modell
fitRI **<- lmer(***y* **~** *x* **+ (***1***|***subject***) , data =** *dat***)** → Erstelle RI Modell
anova(*fitRIRS***,** *fitRI***)** → Vergleiche beim Output, welches Modell tieferes = besseres AIC und BIC hat. → Wähle dieses Modell

Vertrauensintervalle

confint(*fit***)** → 95% Vertrauensintervall für Parameter
confint (*fit***, level =** *0.99***)** → 99% Vertrauensintervall für Parameter

Was sagen diese Intervalle aus? Wenn ein Wert ausserhalb des V.I. liegt, ist der Parameter signifikant. (P-Wert kleiner als α)

confint von RIRS – Modell $y_{ij} = (\beta_0 + u_{1,i}) + (\beta_1 + u_{2,i}) * x_j + \varepsilon_{ij}$

```
> confint(fitRIRS)
Computing profile confidence intervals ...
                2.5 %    97.5 %
.sig01       3.918095 15.358961   → σ1 (von u1,i) ≙ Schwankung von Achsenabschnitt
.sig02      -1.000000  1.000000   → ρ = cor(u1,i , u2,i)
.sig03       0.000000  4.204986   → σ2 (von u2,i) ≙ Schwankung von Steigung
.sigma       2.587218  5.010960   → σ (von ε)
(Intercept) 37.269447 51.997214   → β0
x           18.533068 21.600264   → β1
```

ANOVA – Analysis of Variance - Varianzanalyse

- Macht Aussagen über Mittelwerte indem es Varianzen analysiert
- Ist ein Spezialfall einer linearen Regression

Einweg – ANOVA

Modell

Y ~ X + ε_i → Y kontinuierlich, X Faktor (z.B. Farbe Peperoni Rot, Grün, Gelb)

Detailliertes Modell

$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$ $\varepsilon_{ij} \sim N(0, \sigma^2)$ iid

- Y_{ij} = Zielgrösse. Y hat beliebige (unbekannte) Verteilung.
- μ = Mittelwert über alle Gruppen, ε_{ij} = zufälliger Fehler
- α = Verschiebungen des Mittelwertes μ_i pro Gruppe

Beispiel: Grösse von Peperoni beschreiben (→ kontinuierliche Zielvariable Y) durch beschreibenden Faktor „Farbe". Faktor Farbe hat 3 Faktorstufen: Rot, Grün, Gelb.

SS_B = *Between-sum-of-Squares* ≙ *Streuung zw. den Gruppen*

Jede Faktorstufe ist eine eigene Gruppe. D.h. wir haben die drei Gruppen Rot, Grün, Gelb. Beim SS_B wird geschaut, wie weit die Mittelwerte dieser Gruppen gestreut sind = wie unterschiedlich sie sind. SS_B ist grösser, je unterschiedlicher die Mittelwerte.

$$SS_B = p * \sum_{i=1}^{g} (\overline{Y_i} - \overline{Y})^2$$

p = # Beobachtung pro Gruppe (in jeder Gruppe gleich)
g = # Gruppen (hier: 3 Gruppen Rot, Grün, Gelb)
$\overline{Y_i}$= Mittelwert der einzelnen Gruppen (Mittelwert von Gruppe Rot im ersten Durchgang, von Gruppe Grün im zweiten Durchgang, von Gruppe Gelb im dritten)
$\overline{Y}$= Mittelwert aller Werte (jeder Wert von jeder Gruppe)

SS_W = *Within-sum-of-Squares* ≙ *Streuung innerhalb der Gruppen*

Schaut die Streuung innerhalb der drei Gruppen an, also Streuung aller Werte der grünen Peperoni und der roten Peperoni und der gelben Peperoni. SS_W ist grösser, je grösser die Streuung innerhalb der Gruppen.

$$SS_W = \sum_{i=1}^{g} \sum_{i=1}^{P} (Y_{ij} - \overline{Y_i})^2$$

p = # Beobachtung pro Gruppe (in jeder Gruppe gleich)
g = # Gruppen (hier: 3 Gruppen Rot, Grün, Gelb)
Y_{ij}= Jeder einzelne Wert innerhalb der Gruppe i (Alle Werte von Gruppe Rot im ersten Durchgang, von Gruppe Grün im zweiten, von Gruppe Gelb im dritten)
$\overline{Y_i}$= Mittelwert der einzelnen Gruppen (Mittelwert von Gruppe Rot im ersten Durchgang, von Gruppe Grün im zweiten Durchgang, von Gruppe Gelb im dritten)

Teststatistik

Die Teststatistik ist grösser, je grösser der Unterschied der Mittelwerte zwischen den Gruppen ist und je kleiner die Streuung innerhalb der Gruppen ist. (Teststatistik T ist proportional zu $\frac{SS_B}{SS_W}$)

Exakte Teststatistik T = ***F-Wert***

T = F = $\frac{MS_B}{MS_W}$

$MS_B = \frac{SS_B}{df_B}$ $df_B = g - 1$

g = # Levels im beschreibenden Faktor (hier 3 Levels: Rot, Grün, Gelb)

$MS_W = \frac{SS_W}{df_W}$ $df_W = g * (p-1)$

p = # Beobachtungen/Messwerte pro Gruppe (in allen Gruppen gleich)

Analyse der Teststatistik:

H_0: $\alpha_1 = \alpha_2 = \alpha_3 = ... = 0$ *[H_0 sagt, dass Faktoren keinen Einfluss auf Y haben]*

Verteilung der Teststatistik T unter H_0: $T \sim F_{g-1,\ g*(p-1)}$

[g-1 und g(p-1) geben die Freiheitsgrade an]*

- Nun könnte ein Hypothesentest durchgeführt werden. Der F-Wert = Teststatistik kann mit dem Wert aus der Tabelle mit den F-Verteilungen verglichen werden; wenn der F-Wert höher ist als der kritische Wert der Tabelle ist der Test signifikant.

R-Studio

fit **<- aov(*y ~ x*, data =** *dat*) → Fitte ANOVA mit beschreibendem Faktor *x*
summary(*fit*) → Output:

```
> fit <- aov(y ~ M, data = dat)
> summary(fit)
             Df Sum Sq Mean Sq F value  Pr(>F)
M             6  700.1  116.69   9.228 3.3e-08 ***
Residuals   112 1416.3   12.65
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- Df = Degrees of Freedom: df_B = 6, df_W = 112
- Sum Sq = Sum of Squares = SS: SS_B = 700.1, SS_W = 1'416.3
- Mean Sq = Mean of Squares = MS:
  MS_B = 700.1 / 6 = 116.69, MS_W = 1'416.3 / 112 = 12.65
- F value = F-Wert = Teststatistik: T = F = 9.228
- Pr(>F) = p-Wert: Wenn p-Wert ≤ α, dann ist der Einfluss von *M* auf *Y* signifikant

Falls ANOVA signifikant: Zw. welchen Gruppen sind Unterschiede signifikant?

Methode 1: **t-Test** für alle Gruppenpaare durchführen. → Problem: Gibt $\binom{n}{2}$ Tests bei *n* Gruppen. Da es so viele Tests gibt, gibt es auch viele falsch positive Tests.
→ Lösung: Bonferroni-Korrektur (statt Signifikanzniveau α neu Signifikanzniveau $\frac{\alpha}{n}$)

Methode 2: **TukeyHSD = Tukey Honestly Significant Difference**
\+ Gibt V.I. für die Differenzen der Gruppenmittelwerte an
\+ Die W'keit, dass alle wahren Differenzen im V.I. liegen ist = α

TukeyHSD(*fit*) → Output (Ausschnitt):

df.... SS.... MS.... F....
M m-1 SS.B MS.B F-Wert
Res m*(p-1) SS.W MS.W
MS = SS/df und F = MS.B/MS.W

```
> TukeyHSD(fit)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = y ~ M, data = dat)

$M
            diff        lwr       upr     p adj
M2-M1  0.2042983 -3.4580073  3.866604 0.9999981
N1-M1  3.4631251 -0.1991805  7.125431 0.0766331
N2-M1  6.1961208  2.5338153  9.858426 0.0000309
```

- *diff* = Geschätzte Unterschiede zw. Gruppen M2 zu M1, N1 zu M1 ,....
- *lwr* / *upr* = Grenzen des 95% V.I. für die geschätzten Unterschiede
- *p adj* = für multiples Testen korrigierter p-Wert

plot(TukeyHSD(*fit*)) → Interpretation:

- horizontale Striche sind 95%-V.I. für die Unterschiede zwischen den Levels. Der geschätzte Unterschied ist in der Mitte des Intervalls mit einem Strich angegeben.
- Überall, wo das Intervall die Nulllinie kreuzt, ist der Unterschied nicht signifikant
- Überall dort, wo das Intervall komplett auf der rechten (positiven) Seite der Nulllinie ist, ist das erste angegebene Level signifikant wirksamer als das zweite (z.B. M2 – M1 → Medikament M2 ist wirksamer)
- Überall dort, wo das Intervall komplett auf der linken (negativen) Seite der Nulllinie ist, ist das zweite angegebene Level signifikant wirksamer als das erste (z.B. A – B → Medikament B ist wirksamer)

plot(*fit*, which = c(1:2)) → Residuenanalyse und QQ-Plot

Kontraste

Statt Differenz von 2 Gruppen schauen wir Linearkombinationen von beliebigen Gruppen an. Was wir dafür definieren müssen:

- Vektor mit Gruppenmittelwerten μ: **Vektor μ** = (μ_{Rot} , $\mu_{Grün}$, μ_{Gelb})
- **Kontrast-Matrix K**: Definieren, welche Gruppen wir vergleichen
- **Parameter-Vektor m**: Kann man selber definieren, meist Nullvektor

Idee: Nullhypothese H_0 : K * μ = m → Computer berechnet die p-Werte für alle in K definierten Varianten und korrigiert für multiples Testen

Kontrastmatrix K

Die Zeilen der Kontrastmatrix geben immer eine Linearkombination von Gruppen an, die getestet werden soll. Das heisst eine Zeile = ein Hypothese = ein Vergleich = ein Kontrast.

- **Je weniger Kontraste (= Zeilen) umso mehr Macht hat der Test**.
- R-Studio korrigiert die p-Werte für multiples Testen
- Korrektur für multiples Testen wird nur pro Funktionsaufruf gemacht. Deshalb definiert man am Anfang einen einzigen Satz von Kontrasten (= eine einzige Kontrastmatrix). Man untersucht danach keinen neuen Satz von Kontrasten mehr.

Beispiel: Kontrastmatrix, um gruppenweise Vergleiche zu machen
Es muss in der **Summe immer +1 bzw. -1** geben, d.h. wenn man nur eine Gruppe hat ist es ±1, wenn man zwei Gruppen hat sind beide je ±0.5, etc.

Kontrastmatrix * μ = m

$$\begin{pmatrix} 0.5 & 0.5 & -1 \\ -0.5 & 1 & -0.5 \end{pmatrix} * \begin{pmatrix} \mu_{Rot} \\ \mu_{Grün} \\ \mu_{Gelb} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Erste Zeile → $\mu_{Rot} - 0.5 * \mu_{Grün} - 0.5 * \mu_{Gelb} = 0$
Zweite Zeile: → $-0.5 * \mu_{Rot} + \mu_{Grün} - 0.5 * \mu_{Gelb} = 0$
In der ersten Zeile vergleichen wir die roten Peperoni gegen alle anderen Peperoni.
In der zweiten Zeile vergleichen wir grüne Peperoni gegen alle anderen Peperoni.

R-Studio

library(multcomp)
K <- **rbind("*Rot – Grün*" = c(1 , -1 , 0),** → *"Rot–Grün" = Zeilenname*
"*Rot – Gelb*" = c(1 , 0 , -1),
"*Grün – Gelb*" = c(0 , 1 , -1))
colnames(*K*) <- levels(df$*x*) → Spaltennamen. x aus *fit*

```
> K
             Rot Grün Gelb
Rot - Grün     1   -1    0
Rot - Gelb     1    0   -1
Grün - Gelb    0    1   -1
```

summary(glht(*fit*, linfct = mcp(*x* = K))) → *x* = beschreibender Faktor aus *fit*
Output (nächste Seite)

```
> summary(glht(fit,linfct=mcp(M=K)))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: User-defined Contrasts


Fit: aov(formula = y ~ M, data = dat)

Linear Hypotheses:
             Estimate Std. Error t value Pr(>|t|)
M1-P == 0  -6.3886     1.2197  -5.238   <0.001 ***
M2-P == 0  -6.1843     1.2197  -5.070   <0.001 ***
N1-P == 0  -2.9255     1.2197  -2.399   0.0834 .
N2-P == 0  -0.1925     1.2197  -0.158   1.0000
N3-P == 0  -2.0478     1.2197  -1.679   0.3553
N4-P == 0  -1.5157     1.2197  -1.243   0.6542
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

- Der Unterschied von *M1* zu *P* ist geschätzt *-6.3886*
- 95% - Vertrauensintervall vom Unterschied von *M1* zu *P* ist *[estimate ± 2 * std. error] = [-6.3886 ± 1.2197]*
- Pr(>|t|) ist der p-Wert des Unterschiedes zwischen den verglichenen Faktoren (z.B. zwischen M1 und P). Ist der *p-Wert ≤ α* dann ist der Unterschied *signifikant*

ohne Korrektur: t.test(dat$y[dat$M=='M2'], dat$y[dat$M=='M1'], conf.level = 0.99) #evtl. Werte anpassen
verwenden und nicht tukeyHSD(aov(y~., data=dat), conf.level=alpha)
Bonferroni: t.test(dat$y[dat$M=='M4'], dat$y[dat$M=='M3'], conf.level = 1-(1-alpha)/K)
K ist Anzahl Tests

2-weg – ANOVA

Modell

$Y \sim X_1 + X_2 + \varepsilon_i$ → Y ist kontinuierlich, X_i sind Faktoren (z.B. Farbe, Geschlecht, Länder,…)

Detailliertes Modell

Ohne Wechselwirkung:

$Y_{ijk} = \mu + \alpha_i + \beta_j + \varepsilon_{ijk}$ $\varepsilon_{ijk} \sim N(0, \sigma^2)$ iid

Mit Wechselwirkung:

$Y_{ijk} = \mu + \alpha_i + \beta_j + \delta_{ij} + \varepsilon_{ijk}$ $\varepsilon_{ijk} \sim N(0, \sigma^2)$ iid

- Y_{ijk} ist die kontinuierliche Zielgrösse. Y hat eine beliebige (unbekannte) Verteilung.
- μ entspricht dem Mittelwert μ_{ij} pro Gruppe
- α, β sind die beschreibenden Faktoren. Es kann mehrere beschreibende Faktoren haben, von denen die Zielgrösse Y abhängt.
- α_i und β_j sind die Effekte der Faktoren α und β auf Y. (Genauer: α_i , β_j sind die Verschiebungen zum Mittelwert μ_{ij} pro Gruppe)
- δ_{ij} beschreibt die Interaktion zwischen den Faktoren α und β. („Farbe der Peperoni hat je nach Herkunftsland anderen Einfluss auf Grösse der Peperoni.“)

Nullhypothesen

Zum Modell von oben gibt es drei Nullhypothesen, die geprüft werden sollen:

$H_{0,1}$: $\alpha_i = 0$ → Kein Farb-Effekt

$H_{0,2}$: $\beta_j = 0$ → Kein Herkunftsland-Effekt

$H_{0,3}$: $\delta_{ij} = 0$ → Keine Interaktion, d.h. kein Herkunftslandspezifischer Effekt der Farbe

SS = Sum-of-Squares ≙ Streuung zwischen den Gruppen

Jede der Faktorstufen ist eine eigene Gruppe. Bsp: Faktor Farbe hat drei Gruppen Rot, Grün, Gelb und Faktor Herkunftsland zwei Gruppen Spanien, Mexiko.

Alle SS (SS_F, SS_H, SS_{FH}, SS_{Res}) sind grösser, je unterschiedlicher die Mittelwerte sind.

SS_F = wie weit sind Mittelwerte der drei Gruppen vom Faktor Farbe gestreut = wie unterschiedlich sind sie.

SS_H = wie weit sind Mittelwerte der zwei Gruppen vom Faktor Herkunftsland gestreut = wie unterschiedlich sind sie.

SS_{FH} = wie unterschiedlich sind Mittelwerte der Farben *je nach Herkunftsland*.

SS_{Res} = wie weit sind einzelne Werte in einzelnen Gruppenkombinationen gestreut. Dabei schaut man immer alle Gruppen-Kombinationen von Ländern und Farben an, nicht einfach nur separat Länder und separat Farben. D.h. Streuung der roten Peperoni aus Spanien, der grünen Peperoni aus Mexiko, etc..

$$SS_F = p * h * \sum_{i=1}^{g} (\overline{Y_i} - \overline{Y})^2$$

$$SS_H = p * g * \sum_{j=1}^{h} (\overline{Y_j} - \overline{Y})^2$$

$$SS_{FH} = p * \sum_{i=1}^{g} \sum_{j=1}^{h} (\overline{Y_{ij}} - \overline{Y_i} - \overline{Y_j} + \overline{Y})^2$$

$$SS_{Res} = \sum_{i=1}^{g} \sum_{j=1}^{h} \sum_{k=1}^{p} (Y_{ijk} - \overline{Y_{ij}})^2$$

p = # Beobachtung pro Gruppe (in jeder Gruppe gleich)
g = # Farben (hier: 3 Gruppen Rot, Grün, Gelb)
h = # Herkunftsländer (hier: 2 Gruppen Spanien, Mexiko)
Y_{ijk} = Jeder einzelne Wert separat

$\overline{Y_i}$= Mittelwert der einzelnen Gruppen vom Faktor Farbe (Mittelwert von Gruppe Rot im ersten Durchgang, von Gruppe Grün im zweiten, von Gruppe Gelb im dritten)

$\overline{Y_j}$= Mittelwert der einzelnen Gruppen vom Faktor Herkunftsland (Mittelwert von Gruppe Spanien im ersten Durchgang, von Gruppe Mexiko im zweiten Durchgang)

$\overline{Y_{ij}}$= Mittelwert jeder Farb-Land-Kombination (rote Peperoni aus Spanien, rote Peperoni aus Mexiko, grüne Peperoni aus Spanien, etc.)

$\overline{Y}$= Mittelwert aller Werte

Degrees of Freedom df

df_F : g -1 = #Farben – 1

df_H : h – 1 = #Länder – 1

df_{FH} : (g – 1) * (h – 1)

df_{Res}: g * h * (p – 1) = #Farben * #Länder * (#Messungen pro Gruppe – 1)

Mean Squares

$$MS_F = \frac{SS_F}{df_F}; \quad MS_H = \frac{SS_H}{df_H}; \quad MS_{FH} = \frac{SS_{FH}}{df_{FH}}; \quad MS_{Res} = \frac{SS_{Res}}{df_{Res}}$$

Teststatistik und Verteilung unter $H_{0,1}$, $H_{0,2}$, $H_{0,3}$

$H_{0,1}$: $\alpha_1 = 0$ → Kein Farb-Effekt

Falls $H_{0,1}$ stimmt: $T_1 = \frac{MS_F}{MS_{Res}} \sim F_{df_F\,;\,df_{Res}}$

$H_{0,2}$: $\beta_j = 0$ → Kein Herkunftsland-Effekt

Falls $H_{0,2}$ stimmt: $T_2 = \frac{MS_H}{MS_{Res}} \sim F_{df_H\,;\,df_{Res}}$

$H_{0,3}$: $\delta_{ij} = 0$ → Keine Interaktion = kein Herkunftslandspezifischer Effekt der Farbe

Falls $H_{0,3}$ stimmt: $T_3 = \frac{MS_{FH}}{MS_{Res}} \sim F_{df_{FH}\,;\,df_{Res}}$

*Anmerkung zum **p-Wert*** am Beispiel vom p-Wert von Faktor *X*

p-Wert = P(F > F-Wert von *X*) = Wahrscheinlichkeit, dass F grösser ist als der F-Wert von *X*, wobei F eine F(df_X , df_{Res}) – Verteilung hat

→ R-Studio Befehl: **pf(** *F-Wert von X* , df_X , df_{Res} **, lower.tail = FALSE)**

R-Studio

fit **<- aov(***Y ~ F * H***, data =** *dat***)** → ANOVA mit WW mit Faktoren *F* und *H*

fit2 **<- aov(***Y ~ F + H***, data =** *dat*) → ANOVA ohne WW mit Faktoren *F* und *H*

summary(*fit***)** → Output: [F=Farben, H=Herkunftsland, F:H =WW zwischen F und H]

```
> summary(fit)
             Df Sum Sq Mean Sq F value  Pr(>F)
F             2  329.8  164.90  56.945   5e-14 ***
H             1   32.1   32.10  11.084 0.00157 **
F:H           2    0.4    0.21   0.072 0.93101
Residuals    54  156.4    2.90
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

- Df = Degrees of Freedom: $df_F = 2$, $df_H = 1$, $df_{FH} = 2$, $df_{Res} = 54$
- Sum Sq = Sum of Squares = SS: $SS_F = 329.8$, $SS_H = 32.1$, $SS_{FH} = 0.4$, $SS_{Res} = 156.4$
- Mean Sq = Mean of Squares = MS: $MS_F = 329.8 / 2 = 164.90$, $MS_H = 32.1 / 1 = 32.10$, $MS_{FH} = 0.4 / 2 = 0.21$, $MS_{Res} = 156.4 / 54 = 2.90$
- F value = F-Wert = Teststatistik: $T_1 = 56.945$, $T_2 = 11.084$, $T_3 = 0.072$
- Pr(>F) = p-Wert: Wenn p-Wert ≤ α, dann ist der Einfluss signifikant

TukeyHSD(*fit***)** → Output:

TukeyHSD(*fit***, conf.level =** *0.99***)** → 99%-V.I. (95% ist default)

```
> TukeyHSD(fit)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Y ~ F * H, data = Peperoni)

$F
               diff       lwr      upr     p adj
Gruen-Gelb 1.633479 0.3366259 2.930332 0.0101554
Rot-Gelb   5.584670 4.2878167 6.881523 0.0000000
Rot-Gruen  3.951191 2.6543379 5.248044 0.0000000

$H
                diff       lwr      upr     p adj
Spanien-Mexiko 1.462805 0.5819206 2.34369 0.0015736

$`F:H`
                              diff         lwr        upr     p adj
Gruen:Mexiko-Gelb:Mexiko     1.5960868 -0.6523081  3.8444817 0.3041327
Rot:Mexiko-Gelb:Mexiko       5.7392972  3.4909023  7.9876921 0.0000000
Gelb:Spanien-Gelb:Mexiko     1.5409625 -0.7074324  3.7893574 0.3421850
Gruen:Spanien-Gelb:Mexiko    3.2118334  0.9634385  5.4602283 0.0012607
Rot:Spanien-Gelb:Mexiko      6.9710046  4.7226097  9.2193995 0.0000000
Rot:Mexiko-Gruen:Mexiko      4.1432104  1.8948155  6.3916053 0.0000187
Gelb:Spanien-Gruen:Mexiko   -0.0551243 -2.3035192  2.1932706 0.9999997
Gruen:Spanien-Gruen:Mexiko   1.6157466 -0.6326483  3.8641415 0.2911892
Rot:Spanien-Gruen:Mexiko     5.3749178  3.1265229  7.6233127 0.0000000
Gelb:Spanien-Rot:Mexiko     -4.1983347 -6.4467296 -1.9499398 0.0000144
Gruen:Spanien-Rot:Mexiko    -2.5274638 -4.7758587 -0.2790689 0.0190459
```

- *diff* = Geschätzte Unterschiede zwischen den Gruppen
- *lwr / upr* = Grenzen des 95% V.I. für die geschätzten Unterschiede
- *p adj* = für multiples Testen korrigiertes V.I. (→ zeigt, ob Unterschiede signifikant sind)
- **$F** : Zeigt Unterschied vom Mittelwert an, wenn man Farbe wechselt
  - Wenn man von Grünen zu Gelben Peperoni wechselt, ist der Unterschied in der Grösse im Schnitt 1.633479. *[Grüne sind im Schnitt um so viel grösser]*
- **$H** : Unterschied vom Mittelwert, wenn man Herkunftsland wechselt
  - Wenn man von Spanien zu Mexiko wechselt, ist der Unterschied in der Grösse der Peperoni im Schnitt 1.462805
- **$‘F:H‘** : Zeigt Unterschiede, je nachdem wie man Gruppen kombiniert
  - Wenn man von grünen Peperoni aus Mexiko zu Gelben Peperoni aus Mexiko wechselt, sind die grünen Peperoni im Schnitt 1.5960868 [cm] grösser → Dieser Unterschied ist aber nicht signifikant, da der p-Wert ≈ 0.30
  - Wenn man von gelben Peperoni aus Spanien zu roten Peperoni aus Mexiko wechselt, sind die gelben Peperoni aus Spanien im Schnitt -4.1983347 [cm] grösser → Unterschied ist signifikant, da p-Wert ≈ 0.00

plot(*fit***, which = 1:2) → Residuenanalyse und QQ-Plot**

Annahmen: Daten sind in jeder Gruppe normalverteilt, gleiche Varianz in den Gruppen, Fehler ε_{ij} ist unabhängig.

ANOVA & TukeyHSD vs. Lineare Regression

ANOVA & TukeyHSD geben totale Effekte an, es gibt kein Referenzlevel

Lineare Regression gibt Effekte bezüglich Referenzlevel an

Randomized Block Design

= Verallgemeinerung des gepaarten t-Test. Statt ein Medikament & ein Placebo pro Person wird neu pro Person mehrere Medikamente und ein Placebo verabreicht (Reihenfolge zufällig). Auswertung via 2-weg Anova: Y ~ Medi + Person

Unbalanciertes Design

Balanciertes Design: Alle Gruppen haben die gleiche #Stichproben = #Messungen

Unbalanciertes Design: Verschieden grosse Stichproben pro Gruppen

Nachteile:

- Man kann nicht mehr den Effekt eines Faktors bestimmen und dabei die übrigen Faktoren ignorieren. Deshalb: Alle Parameter müssen gleichzeitig geschätzt werden. (Nicht nacheinander, wie es beim balancierten Design möglich ist)

Es gilt F.G = MS.G / MS.r

	df.....	SS.....	MS.....	F.....	P.....
M	df.M	270.2	MS.M	F.M	p.M
G	df.G	8.6	MS.G	F.G	p.G
M:G	df.MG	8.0	MS.MG	F.MG	p.MG
Res	df.r	289.2	MS.r		

Es gilt p-Wert = P(F > F.M), wobei F eine F(df.M, df.r) Verteilung hat. In R kann dies mit pf(F.M, df.M, df.r, lower.tail=FALSE) berechnet werden

- Quadratsumme SS kann nicht mehr den einzelnen erklärenden Variablen zugeteilt werden → SS ist unterschiedlich, je nach dem welche anderen Variablen schon im Modell sind

Lösung:

Wir verwenden bei unbalancierten Designs statt der Befehle *aov, summary* den Befehl *drop1*. Grund: *aov* und *summary* sind von der Reihenfolge abhängig, *drop1* aber nicht. *drop1* lässt alle Werte Weg, die in der Schnittmenge von A und B liegen.**Beispiel**: P-Werte und Residuenquadratsummen sind bei summary(*fit1*) und summary(*fit2*) beim unbalancierten Design unterschiedlich. Bei drop1(*fit1*) und drop1(*fit2*) aber identisch.

fit1 <- aov(*y ~ Farbe + Herkunftsland*) → **drop1(*fit1*, test = "F")**

fit2 <- aov(*y ~ Herkunftsland + Farbe*) → **drop1(*fit2*, test = "F")**

Anm.: Beim Befehl lm() für eine lineare Regression passiert das nicht, dort kann man auch bei unbalancierten Gruppen den Befehl summary() ohne Probleme anwenden.

Kategorielle Daten

Fisher's exact Test

2 x 2 – Tabellen, Verteilung der Teststatistik **exakt**

Nullhypothese H_0 = Spalten und Zeilen sind unabhängig

Hypergeometrische Verteilung (Repetition)

Urne mit *N* Kugeln, *m* sind markiert, *n* Kugeln ohne zurücklegen ziehen → wie viele davon sind markiert? = Zufallsvariable *X*

X ~ Hyper(N,m,n)

$$P(X = x) = \frac{\binom{m}{x}\binom{N-m}{n-x}}{\binom{N}{n}} \qquad E(X) = \frac{n*m}{N} \qquad Var(X)\ kompliziert$$

Prinzip

Ist das Ergebnis plausibel unter der Nullhypothese H_0? Beispiel:

	Medikament	Placebo	Total
Geheilt	15 = x	9	24 = m
Nicht geheilt	10	11	21
Total	25 = n	20	45 = N

- Zufallsvariable X = # geheilter Patienten in der Medikamenten-Gruppe
- H_0 = Medikament hat keinen Einfluss auf die Heilung
- Verteilung von X unter H_0: X ~ Hyper(N , m , n) = X ~ Hyper(45, 25, 24)
- P-Wert = P(X ≥ #m in n) = P(X ≥ x) = P(X ≥ 15) = 1 – P(X ≤ 14) = 0.76 = 0.24

R-Studio:

1 – phyper(x-1 , m , N-m , n) = **1 – phyper(*14*, *24*, *21*, *25*)** = 0.24 = P-Wert

Wenn P-Wert ≤ α, dann können wir die Nullhypothese verwerfen.

Odds und Odds Ratio (Repetition)

$$Odds\ (A) = \frac{P(A)}{1-P(A)} \qquad P(A) = \frac{Odds(A)}{1+Odds(A)} \qquad Log-Odds(A) = \log\left(Odds(A)\right)$$

$$Odds-Ratio\ = \frac{Odds(A)}{Odds(B)} = \frac{Odds(Geheilt\ mit\ Medi)}{Odds(Geheilt\ ohne\ Medi)}$$

Odds geben an, wie viel wahrscheinlicher A ist als A^c („nicht A"). Odds-Ratio ist das Verhältnis der Odds von A zu den Odds von B. Es gibt an, wie viel grösser die Odds von A als die Odds von B sind.

Bsp: Wenn das Odds-Ratio grösser als 1 ist, sind Odds von „Geheilt mit Medi" grösser als Odds von „Geheilt ohne Medi" → W'keit, mit Medi gesund zu werden ist grösser als die W'keit ohne Medi gesund zu werden. Wäre das Odds-Ratio genau 1, wäre die Heilungswahrscheinlichkeit mit und ohne Medi gleich gross. Wäre das Odds-Ratio kleiner als 1 würde man ohne Medikament eher gesund werden als mit Medikament. Das *Odds-Ratio sagt aber nichts darüber aus, wie hoch die Heilungsw'keit an sich ist! Es sagt nur aus, ob sie mit oder ohne Medikament grösser ist!*

Wie gross ist die Differenz zwischen dem geschätzten odds-ratio O(G | M) / O(G | P) und dem theoretischen odds-ratio, bei dem überhaupt kein Zusammenhang zwischen der Behandlung und der Heilung existiert?

Es gilt O(G | M) / O(G | P) = (medi.g/medi.ng) / (plac.g/plac.ng). Falls keinen Einfluss existiert, ist der odds-ratio gleich 1.

R-Studio

m **<- matrix(c(*15*,*10*,*9*,*11*) , 2 , 2)** → 2,2 = Anzahl Zeilen und Spalten

→ 15 = geheilt mit Medi, 10 = nicht geheilt mit Medi,

9 = geheilt ohne Medi, 11 = nicht geheilt ohne Medi

fisher.test(*m* , alternative = "*greater*") → oder "two.sided" oder "less".

fit <- fisher.test(*m* , alternative = "*greater*") → Resultat unter *fit* speichern

*ft***$conf.int** → 95% - Vertrauensintervall für Odds

Chi Quadrat Test

n x m – Tabellen, Verteilung der Teststatistik **asymptotisch**

Ziel: Visualisierung oder Abhängigkeiten finden

Prinzip

Nullhypothese H_0 = Spalten und Zeilen sind unabhängig. Die W'keit, dass eine Messung genau in der Spalte i und der Zeile j landet, ist unter H_0 =

[A = Spalten, B = Zeilen, i = Nummerierung der Spalten, j = Nummerierung der Zeilen]

$$P(A = i \cap B = j) = P(A = i) * P(B = j) \approx \hat{P}(A = i) * \hat{P}(B = j) = \frac{N_i}{N} * \frac{N_j}{N}$$

Wobei N_i die #Beobachtungen in der i-ten Spalte und N_j die #Beobachtungen in der j-ten Zeile sind.

Falls H_0 stimmt ist der Erwartungswert jeder Zelle $\mathbf{E_{ij}} = N * \frac{N_i}{N} * \frac{N_j}{N} = \frac{N_i * N_j}{N}$

Pearson Chi-Quadrat Statistik:

Fragestellung: Wie verschieden sind die beobachteten Werte von den erwarteten?

[O_{ij} = beobachteter Wert in der Spalte i und Zeile j]

$$X^2 = \sum_{i=1}^{n}\sum_{j=1}^{m}\frac{(O_{ij} - E_{ij})^2}{E_{ij}} = \sum_{i=1}^{n}\sum_{j=1}^{m}R_{ij}^2$$

Falls H_0 stimmt, folgt Teststatistik X^2 einer Chi-Quadrat Verteilung mit (n-1)*(m-1) Freiheitsgraden..

Pearson Residuen

$$R_{ij} = \frac{O_{ij} - E_{ij}}{\sqrt{E_{ij}}}$$ = Beitrag jeder Zelle zur Abweichung vom Modell

Faustregel

Gute Approximation: Bei jeder Zelle ist der Erwartungswert > 5 ist = **jedes E_{ij} > 5**

Approximation ist gerade noch **ok**: **Jedes E_{ij} > 1**

Ungenügende Approximation: Bei mindestens **einer einzigen Zelle ist E_{ij} < 1**

In diesem Fall muss man Kateogrien zusammenfassen oder andere Tests verwenden.

R-Studio

Umformen einer Tabelle zu einem Dataframe: df <- **as.data.frame(*tab*)**

Umformen eines Dataframes zu einer Tabelle:

tab <- **xtabs(*Y* ~ . , data= *df*)** → Alle Spalten werden miteinbezogen

→ Y muss eine kontinuierliche Variable sein (d.h. keine Faktorstufen)

tab <- **xtabs(*Y* ~ A + B , data= *df*)** → Wenn man nicht alle Spalten des Dataframes miteinbeziehen will

chisq.test(*tab*) → Output: X-Squared = Pearson Chi-Quadrat Statistik, df = (m-1)*(n-1) = #Freiheitsgrade, p-value = 1 – pchisq(q = X^2 , df = *df*)

ct <- chisq-test(tab)

ct$residuals → Gib die Pearson Residuen aus.

Konstruiere Tabelle:

```
> tab1 <- xtabs(Freq ~ A2 + B2 , data=dat)  # 2x2 Tabelle von Fisher's Test
> tab2 <- xtabs(Freq ~ A + B , data=dat)  # 4x3 Tabelle von Chi-Quadrat Test
> tab3 <- xtabs(Freq ~ A2 + B , data=dat)  # 2x3 Tabelle von Logistische Regression
```

Chi-square test:

```
> test <- chisq.test(tab2)
> test
> test$expected
> test$residuals
```

Logistische Regression:

```
> fit1 <- glm(A2 ~ B, weights = Freq, family = binomial, data = dat)
> fit2 <- glm(A2 ~ B * K, weights = Freq, family = binomial, data = dat) # weights nicht vergessen
> summary(fit)
```

Visualisierung

Mosaic Plot = Flächen proportional zu Tabelleneinträgen R-Studio:

library(vcd)

mosaic(*Y* ~ *X* + *Z*, data = *df*, shade = TRUE)

→ Einfärben vom Mosaik, Farbe falls Pearson Residuen ausserhalb von [-2 , 2] liegen. Rot = sehr kleiner Tabelleneintrag, Blau = sehr grosser.

→ Unten rechts: p-Wert des Chi-Quadrat Tests

Logistische Regression

2 x m – Tabellen, Verteilung der Teststatistik **asymptotisch**

Mix aus mehreren kontinuierlichen & kategoriellen erklärenden Variablen möglich

Repetition:

Die logistische Regression kann man benutzen, wenn die Zielvariable binär ist (0/1, richtig/falsch, krank/gesund, …)

Y ~ Bin(n , p(x))

$Y = \text{log-odds} = \log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 * x$

Folgende Zusammenhänge gelten:

odds(x) = exp(log-odds(x))

probs(x) = odds(x) / (1 + odds(x))

class(x) = probs(x) ≥ 0.5 = odds(x) ≥ 1 = log-odds(x) ≥ 0

R-Studio :

glm(*y ~ x* , family = binomial, data = *dat*) → logistische Regression

glm(*y ~ x* ,weights = z, family = binomial, data = *dat*)

→ weights = eine Art Gewichtung der beschreibenden Variablen

- Wenn man *x* um eine Einheit erhöht, erhöhen sich die log-odds um $+\beta_1$
  95%-Vertrauensintervall = $\beta_1 \pm 2 * \text{Std.error}$
- Wenn man *x* um eine Einheit erhöht, erhöhen sich die odds um $*e^{\beta 1}$
  95%-Vertrauensintervall = $e^{\beta 1 \pm 2 * \text{Std.error}}$

Einfache oder multiple Regression

Einfache Regression: Y ~ X; Wenn sich X um eine Einheit erhöht, erhöht sich Y um β_1

Multiple Regression : Y ~ X+Z; Wenn sich X um eine Einheit erhöht und Z gleich bleibt, erhöht sich Y um β_1

Poweranalyse – die richtige Stichprobengrösse

Wie viele Stichproben braucht es, um eine bestimmte Alternative mit einer bestimmten Macht erkennen zu können? Zu viele Stichproben sind unnötiger Aufwand, zu wenige machen die Studie nutzlos.

```
macht <- function(n, beta0, betaA, alpha, reps){
  res <- vector("numeric", reps)
  for(i in 1:reps){
    ## Simuliere Daten
    X <- sim(n, betaA)
    ## Mache Test
    p.val <- test(X, beta0)
    ## Speichere Ergebnis
    res[i] <- (p.val < alpha)
  }
  list(m = mean(res), s = sd(res)/sqrt(reps) )
}
```

Idee: Vor dem Experiment eine **Power-Analyse** durchführen, um die Stichprobengrösse zu bestimmen. Nachdem Experiment ein **Vertrauensintervall** für die gesuchten Parameter bestimmen.

Nach dem Experiment erstellt man immer ein V.I., um das Resultat besser interpretieren zu können.

Fehler 1. Art

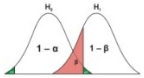
H_0 stimmt aber wird verworfen. W'keit für Fehler 1. Art ist ≤α

Fehler 2. Art

H_A stimmt, H_0 stimmt nicht. H_0 wird aber trotzdem nicht verworfen.
W'keit für Fehler 2. Art ist = 1 – Macht

Macht

Macht ist die Wahrscheinlichkeit, eine Alternativhypothese H_A zu erkennen (falls sie stimmt). W'keit für die Macht ist **= 1 – P(Fehler 2. Art)**
Fehler 2. Art wie auch Macht können nur mit einer *konkreten* Alternativhypothese berechnet werden.



- grün = α = Fehler 1. Art (falls H_0 stimmt), da Nullhypothese H_0 fälschlicherweise verworfen wird obwohl sie stimmen würde.
- rot = β = Fehler 2. Art (falls H_A stimmt), da dort H_0 nicht verworfen wird obwohl sie falsch ist und somit die Alternativhypothese nicht erkannt wird.
- nicht markiert = 1-β = Macht (falls H_A stimmt), da dort H_0 verworfen und somit die Alternativhypothese erkannt wird.

Berechnung der Macht

- Mit Theorie ist es sehr genau und schnell berechnet, aber extrem kompliziert
- Mit Simulation ist es fast immer möglich, hat aber einen grossen Programmieraufwand, ist ungenau(er) und evtl. langsam
- Problem bei beiden Varianten: Was soll die konkrete Alternative H_A sein?

Simulation

Prinzip: Simuliere 1'000 Binomialtests für eine Alternativhypothese und schaue, wie oft der Test verworfen wird. Die Macht ist dann = $\frac{\#\,Verworfene\ Tests}{1000}$

R-Studio

Macht beim Binomialtest:

```
machtBinom <- function(n = 50 , reps = 1000 , alpha = 0.05 , pA = 0.7 , p0 = 0.5 , alt = "two.sided") {
    res <- vector("numeric" , reps)
    for (i in 1:reps) {
        x <- rbinom(n = 1 , size = n , prob = pA)
        tmp <- binom.test(x , n = n , p = p0 , alternative = alt)
        res[i] <- (tmp$p.value < alpha)
    }
    list (m = mean(res), s = sd(res)/sqrt(reps))
}
```

Setze set.seed(4120). Berechnen Sie die Macht von einem ANOVA-Test mit einem Faktor mit 4 Levels mit den default-Werten zuerst mit n = 11 und danach mit n = 22 Beobachtungen pro Level. Wie gross ist die Differenz von der Macht? Geben Sie den Absolutbetrag an
Die machtAnova1(n = c(11, 11, 11, 11)) ist 0.618 und die machtAnova1(n = c(22, 22, 22, 22)) ist 0.92. Somit ist die Lösung 0.302. # c(n,n,n,n) heisst wir haben 4 levels für n jeweils

machtBinom() → Berechnen mit Default-Werten
machtBinom(n = 120) → Berechnen mit n = 120, restliche Variablen mit Default
→ *list(…) :* m = **Macht,** s = **Standardfehler**

- *function(…)* → alles innerhalb der Klammern sind Variablen, die in der Funktion vorkommen. Der Wert, der in der Klammer gesetzt ist, ist der Default-Wert.
- *res* ist ein Null-Vektor mit *reps* Stellen. Da *reps* = 1'000 hat also *res* 1'000 Stellen.
- *for(i in 1:reps)* bedeutet, dass die folgende Schleife (= alles in der geschweiften Klammer) reps = 1'000 Mal wiederholt wird. Dabei wird jedes Mal die Variable i eines höher gesetzt (erster Durchgang i = 1, zweiter Durchgang i = 2, …)
- *rbinom(…)* simuliert Daten einer Binomialverteilung (=gibt #Erfolge an) bei *size = n* Versuchen und einer Erfolgswahrscheinlichkeit von *prob = pA* → *rnorm wäre analog für Normalverteilung*
- *binom.test* =exakter Test eines Bernoulli-Experiments. *x* =#Erfolge, *n* =#Stichproben.
- *res[i] <- (tmp$p.value < alpha)* im Vektor *res* wird von allen tausend Durchgängen gespeichert, ob der Test signifikant ist oder nicht (signifikant, wenn p.value < α)

Suche der richtigen Stichprobengrösse – von Hand:

(Funktioniert auch bei t-Test und Anova, dort muss man den Befehl machtBinom *ersetzen)*

Der Test gibt die Macht für eine bestimmte Stichprobengrösse an. Um die Stichprobengrösse für eine bestimmte Macht zu berechnen, kann man von Hand annähern. Beispiel: Wir wollen eine Macht von ≥90% haben. Wie gross soll die Stichprobe sein?

machtBinom(n = 10) → Macht zu klein, versuche nochmals mit grösserer Stichprobe
machtBinom(n = 100) → Macht zu gross, versuche mit kleinerer Stichprobe
…
machtBinom(n = 68) → Macht zu klein
machtBinom(n = 69) → Macht zu gross

→ Wir wollen eine Macht ≥90%, also wählen wir das erste n, bei dem Macht ≥ 0.9 ist. In unserem Bsp. wäre also die Stichprobengrösse n = 69.

Suche der richtigen Stichprobengrösse – mit R-Studio:

```
nall <- seq(10 , 100 , by = 10)          [by = 1 wenn es exakt sein muss]
macht <- vector("numeric" , length(nall))
for (j in 1:length(nall)) {
    n <- nall[j]
    macht[j] <- machtBinom(n = n, pA = 0.75 , alt = "greater")
}                          [--> befehl machtBinom muss zuerst definert werden!]
which(macht > 0.9)[1]
```

- *nall* = Vektor von 10 bis 100 in 10er-Schritten (nall = (10, 20,… , 100). by = 10 kann geändert werden. Kleinere Schritte = genauere Resultate aber längere Berechnungszeit. Auf eine Stelle genau bei **by = 1**
- *which(macht > 0.9)[1]* = erste Stichprobengrösse, bei der die Macht > 0.9 ist.

Macht beim zweiseitigen t-Test:

```
machtTtest <- function(n1 = 20 , n2 = 20 , m1 = 0 , m2 = 1 , s1 = 1 , s2 = 1 , reps = 1000 , alpha = 0.05) {
    res <- vector("numeric" , reps)
    for (i in 1:reps) {
        x <- rnorm(n = n1 , mean = m1 , sd = s1)
        y <- rnorm(n = n2 , mean = m2 , sd = s2)
        tmp <- t.test(x , y , paired = FALSE)
        res[i] <- (tmp$p.value < alpha)
    }
    list(m = mean(res) , s = sd(res)/sqrt(reps))
}
```

→ *list(…) :* m = **Macht,** s = **Standardfehler**

Suche der richtigen Stichprobengrösse – mit R-Studio:

```
nall <- seq(10 , 100 , by = 10)      [by = 1 wenn es exakt sein muss]
nn <- length(nAll)
macht <- matrix(0 , nn , nn)
for (j1 in 1:nn) {
    cat("Schleife" , j1 , " von " , nn , "\n")
    for (j2 in 1:nn) {
        n1 <- nall[j1]
        n2 <- nAll[j2]
        macht[j1 , j2] <- machtTtest(n1 =n1 , n2 =n2, s1 =1 , s2 =5)$m
    }
}
```

- seq(…) → Start- und Endpunkt der Stichprobengrösse sowie Schrittgrösse
- n1, n2 müssen definiert sein
- **s1=1, s2=5** sind Beispiele, es müssen dann die richtigen Werte eingesetzt werden
- *macht* ist eine Matrix in denen die Macht je nach Gruppengrösse eingetragen ist. Beim gesuchten Wert geben die Spalten und die Zeilen die Gruppengrössen an. *Beispiel: Spalte 1, Zeile 3*
  - Spalte 1 = erster Wert für die Gruppe n1 = Stichprobengrösse 10
  - Zeile 3 = dritter Wert für Gruppe n2 = Stichprobengrösse 30

Macht bei 1-weg ANOVA:

```
machtAnova <- function(n , mu , s = 1 , reps = 1000, alpha = 0.05) {
    res <- vector("numeric" , reps)
    for (i in 1:reps) {
        x <- rep(LETTERS[1:length(n)] , times = n)
        y <- vector("numeric", 0)
        for (j in 1:length(n)) {
            y <- c(y , rnorm(n[j] , mean = mu[j] , sd = s))
        }
        df <- data.frame (x = x , y = y)
        sm <- summary(aov(y ~ x , data = df))
        pval <- sm[ [ 1 ] ] [ [ 5 ] ] [ 1 ]
        res[i] <- ( pval < alpha )
    }
    list(m = mean(res) , s = sd(res)/sqrt(reps))
}
```

→ *list(…) :* m = **Macht,** s = **Standardfehler**

```
# Macht von z-test
alpha <- 0.05
s <- sqrt(5.29)
mu0 <- -0.9
muA <- -0.4
n <- 21
c <- qnorm(alpha, mu0, s/sqrt(n), FALSE)   # c: P_H0(T > c) = alpha
macht1 <- pnorm(c, muA, s/sqrt(n), FALSE)   # P_HA(T > c)
```

Studiendesign

Vorgehen:
1. Testbare Hypothese formulieren & testbare Vorhersage machen
2. Entscheiden, welche statistische Auswertung gemacht wird und wie gross die Stichproben sein müssen.
3. Pilotstudie machen
4. Daten erheben und auswerten

1. Hypothese Formulieren

Die Hypothese muss präzise formuliert, testbar und objektiv sein
Aus der Hypothese muss man eine testbare Vorhersage machen.

Sei X ~ N(-4.3, 3.24). Wie gross ist P(X <= x) für x = -2.8?
Mit pnorm(-2.8, -4.3, 1.8) kann die gesuchte Wahrscheinlichkeit ausgerechnet werden. Somit ist die Lösung 0.798.
Sei X ~ Binom(10, 0.7). Wie gross ist P(X = x) für x = 2?
Mit dbinom(2, 10, 0.7) kann die gesuchte Wahrscheinlichkeit ausgerechnet werden. Somit ist die Lösung 0.001.
Sei X ~ Binom(20, 0.5). Wie gross ist P(X > x) für x = 4?
Mit pbinom(4, 20, 0.5, FALSE) oder 1 - pbinom(4, 20, 0.5) kann die gesuchte Wahrscheinlichkeit ausgerechnet werden. Somit ist die Lösung 0.994.
Sei X ~ N(7.3, 0.16). Setze set.seed(2232). Ziehe dann 13 Realisationen von dieser Verteilung. Berechne anschliessend den Mittelwert.
Was ist die Differenz zwischen dem Mittelwert und mu = 7.3? Geben Sie eine positive Zahl ein.

Sei X ~ t(3). Für welches x gilt P(X <= x) = 0.6?
Mit qt(0.6, 3) kann das gesuchte Quantil berechnet werden. Somit ist die Lösung 0.277.
Sei X ~ Binom(11, 0.4). Setze set.seed(7408). Ziehe dann 9 Realisationen von dieser Verteilung.
Berechne anschliessend den Standard-Schätzer für p. Wie gross ist dieser?
Mit mean(rbinom(9, 11, 0.4)/11) kann der Standard-Schätzer für p berechnet werden. Somit ist die Lösung 0.374.

2. Statistische Auswertung

Muss vor der Datenerhebung definiert werden!

- Welche statistischen Tests werden verwendet (z.B. t-Test)?
- Wie viele und welche Daten werden benötigt?
- Wie gross sollen erhobene Stichproben sein (Poweranalyse mit Daten aus Pilotstudie, kann erst nach Pilotstudie definitiv bestimmt werden)?

3. Pilotstudie

Wenige Daten erheben. Ziel: Erfahrungen sammeln in der Datenerhebungen sowie Schwierigkeiten erkennen und beheben. Testen, ob man die Daten wie geplant auswerten kann.

➔ Wie viele Daten braucht es? (Stichwort $\sqrt{n}$ – Gesetz: Wie viele Daten brauche ich für einen genügend kleinen Standartfehler?)

➔ Fehlen noch Daten, die für die Auswertung benötigt werden? Wie kann man diese fehlenden Daten erheben?

➔ letzte Korrekturen machen

$\sqrt{n}$ – Gesetz: Einzelschwankungen wären ~120 Sekunden. Bei 16 Messungen wäre die Schwankung = Standartabweichung $\sigma = 120s * \frac{1}{\sqrt{16}} = 120s * \frac{1}{4} = 30s$

4. Daten erheben und auswerten

KERZ

Kontrolle

Kontrollgruppe, die zufällig zugeteilt ist.

Experiment

Experiment gut aufbauen. Optimal: Randomisiertes, kontrolliertes Experiment mit Replikaten

- **Korrelation ≠ Kausalität**. Mithilfe von Experimenten kann man Kausalität aufzeigen.

Beobachtungsstudie (Bsp. aus Vorlesung): Vergleiche zwischen zwei Bauern, die in möglichst vielen Punkten übereinstimmen.

- Man muss sehr skeptisch sein, da viele Confounder vorhanden sein können!
  - Confounder = Faktor, der das Resultat mitbeeinflusst.
- Man kann nur Assoziationen = Korrelationen zeigen, keine Kausalität

Replikation

Experiment muss wiederholbar sein. z.B. an verschiedenen Orten durchführen.

Zufall

Zufall ist wichtig, z.B. für Kontrollgruppenzuteilung. ABER: Trotzdem durchdenken, damit keine Confounder entstehen.

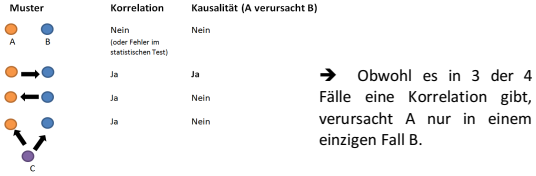
Die 7 Todsünden

Prüfe Studien immer auf diese 7 Punkte!

1. Korrelation und Kausalität

Korrelation beweist keine Kausalität. Mit Beobachtungsstudien kann man Korrelation zeigen, aber keine Kausalität. Um Kausalität zu beweisen, muss man Experimente machen. [Man kann dann Kausalität mit *sehr viel* zusätzlichem Aufwand auch durch Beobachtungsstudien beweisen.]

Mögliche Zusammenhänge Korrelation <-> Kausalität:



➔ Obwohl es in 3 der 4 Fälle eine Korrelation gibt, verursacht A nur in einem einzigen Fall B.

2. Pseudoreplikate

Replikate müssen **unabhängige** Versuchseinheiten sein. Es gibt hierbei leider nicht eine exakte Grenze; Kritiker werden beinahe immer irgendeine mögliche Abhängigkeit finden. Das Ziel ist also, die Versuche so unabhängig wie möglich zu machen.

Beispiel für Pseudoreplikat: Zwei Gruppen von Fischen werden getestet. Fische F_A werden mit Medikament behandelt und sind in Aquarium A, Fische F_B sind die Kontrollgruppe in Aquarium B. Wenn die Fische in Aquarium A gesund werden und diejenigen in Aquarium B nicht, könnte es ja theoretisch auch sein, dass die Ursache eine schlechte Wasserqualität im Aquarium B ist. Wir wissen also nicht, ob das Medikament wirklich einen Einfluss auf die Genesung hat oder ob es nur an den Unterschieden der beiden Aquarien liegt.

3. Behandlungen haben Confounder

Confounder sind Faktoren, die das Ergebnis beeinflussen können. Die Variable, die wir wirklich untersuchen wollen darf nicht mit einem Confounder korrelieren.

Beispiel für Confounder: Wir wollen testen, ob Leute im Durchschnitt lieber klassische Musik oder Hip-Hop hören. Dazu organisieren wir einen Pianisten, der klassische Musik spielt und haben eine Musikanlage, auf der wir Hip-Hop abspielen können. Wir erhalten das Resultat, dass die Mehrheit der Zuhörer lieber klassische Musik hört. Das Problem ist, dass der Confounder „Live-Musik" vs. „Musikanlage" genau mit der erklärenden Variablen „klassische Musik" vs. „Hip-Hop" korreliert. Wir wissen also schlussendlich nicht, ob die Leute wirklich die klassische Musik lieber haben als Hip-Hop oder ob sie einfach lieber jemandem zuhören, der live Musik spielt als Musik aus der Stereoanlage zu hören.

4. Beobachter hat Bias

Erwartungen beeinflussen die Wahrnehmung. Das trifft sowohl auf die Wahrnehmung des Studienleiters als auch auf diejenige der Studienteilnehmer zu (→ Deshalb wirken Placebos). Aus diesem Grund muss man möglichst objektive Messmethoden haben. Optimal wären doppelblinde Studien. Dann können weder der Studienleiter noch die Studienteilnehmer von Erwartungen beeinflusst werden.

5. Verhaltensänderung wegen Experiment-Setting

Sowohl Tiere als auch Menschen verhalten sich je nach Umgebung anders. Der Experiment-Aufbau kann also das Verhalten der Probanden beeinflussen.

Das kann man abschwächen durch verschiedene Methoden: Tarnung, Gewöhnung, etc. komplett verhindern kann man es aber nicht.

6. Schlechte/Keine Kontrollen

Kontrollen müssen sehr sorgfältig gewählt werden. Sie sollten möglichst ähnlich sein zu der Behandlungsgruppe, damit keine Confounder das Ergebnis verzerren können.

Je nach Hypothese machen vielleicht nicht immer dieselben Kontrollgruppen Sinn. Man sollte sich auf jeden Fall immer vor der Studie überlegen, wie die Kontrollgruppe aussehen soll. Vielleicht macht es sogar Sinn, mehrere verschiedene Kontrollgruppen zu machen (z.B. einmal mit Placebo, einmal ganz ohne Medikament).

7. Nullhypothese "beweisen"

Wenn die Nullhypothese H_0 nicht verworfen wird, heisst das **nicht**, dass sie automatisch stimmt. Analog wenn die Nullhypothese H_0 verworfen wird, heisst das **nicht**, dass die Alternativhypothese bewiesen ist.

Wenn wir H_0 nicht verwerfen: Entweder ist H_0 tatsächlich falsch oder wir haben zu wenig Macht, um eine Abweichung festzustellen.

Alternative (Bsp: Münzwurf): **Vergleiche Vertrauensintervall und irrelevanten Bereich**

Definiere ein Intervall, in dem die Münze „fair genug" ist. Wenn das 95% Vertrauensintervall der Gewinnwahrscheinlichkeit in diesem Bereich liegt, ist die Münze „fair genug" für uns und wir definieren sie deshalb als Fair.

PCA – Principal Component Analysis

= Hauptkomponentenanalyse. Stellt „Besten" Subraum dar = Subraum mit grösster Varianz bezüglich Residuenquadratsumme. Ziel ist, eine möglichst grosse Streuung der Daten darstellen zu können. **Mögliche Ziele:**

- Visualisieren von hochdimensionalen Datensätzen (> 3D)
- Komprimieren von vielen zu wenigen Variablen, die die Daten möglichst gut beschreiben
- 1-Dimensionalen Index erstellen, der die Subjekte möglichst gut unterscheidet

Prinzip

Man legt eine Koordinatenachse so hin, dass man eine möglichst grosse Streuung der Daten auf dieser Achse hat. Danach legt man im 90° Winkel eine zweite Koordinatenachse hin, bei der man wieder eine möglichst eine grosse Streuung der Daten auf dieser Achse hat. Danach legt man eine dritte Koordinatenachse wieder im 90° Winkel auf die beiden anderen Achsen, so dass wieder eine möglichst grosse Streuung der Daten auf diese Achse ist. So macht man immer weiter.

Wichtig: Bildlich darstellen kann man das nicht, da unser Raum nur 3 Dimensionen hat. Aber in der Theorie kann man auch 100 oder 1'000 Koordinatenachsen so aufeinander legen, dass alle im rechten Winkel zueinander stehen.

Einheiten

Wenn die Einheiten der Messungen verändert werden, verändert sich die Varianz. Bsp: eine Messung in Metern hat eine 1'000x grössere Varianz als in Kilometern.

Faustregeln

- Daten immer zentrieren.
- Falls alle Variablen in der gleichen Einheit sind: Nicht skalieren
- Falls Variablen in unterschiedlichen Einheiten sind: Skalieren

Vorgehen

1. Konvention: Zentrieren

Lege den Ursprung der Koordinatenachse ins Zentrum der Punktwolke. (Je nach dem auch noch skalieren).

Sei x = [x1, x2] ein Datenpunkt. Dann ist der zentrierte Datenpunkt xm = [x1 - m1, x2 - m2]
Sei xm = [xm1, xm2] ein zentrierter Datenpunkt. Dann gilt z1 = PC1 * xm und z2 = PC2 * xm
Hinweis: Wenn Xm die Matrix mit den zentrierten Daten ist, dann erhalten Sie die rotierte Datenmatrix mit Z = Xm * M,
wobei M die Rotationsmatrix mit den PC's in den Spalten ist

2. Setze die 1. Hauptkomponente

Lege eine Gerade in die Richtung, in die die grösste Streuung der Daten ist. Diese Gerade heisst „1. Hauptkomponente" = PC1. Es ist egal, ob die Hauptkomponente in positive oder negative Richtung schaut (nur das Vorzeichen ändert sich). Normiere dann die 1. Hauptkomponente auf die Länge 1. Dieser Vektor mit der Länge 1 hat den Ursprung auf dem Ursprung des Koordinatensystems.

Anm.: Der Vektor, der die PC1 beschreibt, wird bezüglich des schon vorhandenen Koordinatensystems angegeben. Es ist also ein normaler Vektor PC1 = (φ_{11} , φ_{21})

3. Setze die 2. Hauptkomponente

Lege eine Gerade senkrecht zur 1. Hauptkomponente wieder durch den Ursprung des Koordinatensystems. Lege die Gerade so hin, dass sie in die Richtung der grössten Streuung schaut (bei 2D gibt es nur eine einzige Möglichkeit). Normiere dann diese 2. Hauptkomponente PC2 auf die Länge 1. Dieser Vektor hat den Ursprung auf dem Ursprung des Koordinatensystems und steht senkrecht zur PC1. PC2 = (ϕ_{12} , ϕ_{22})

4. Setze die 3. Hauptkomponente

… und immer so weiter. Es gibt immer genau so viele Hauptkomponenten wie der Datensatz Dimensionen hat (die Dimensionalität ändert sich also nicht)

Loadings

= Richtung der Hauptkomponenten bezüglich der Standartbasis.
PC1 = (ϕ_{11} , ϕ_{21}) → ϕ_{11} , ϕ_{21} sind die loadings der Hauptkomponente 1
PC2 = (ϕ_{12} , ϕ_{22}) → ϕ_{12} , ϕ_{22} sind die loadings der Hauptkomponente 2
…
Vorzeichen der Werte ϕ_{11} , ϕ_{21}, … ändern je nach Orientierung der PC's.

PCA – Basiswechsel mit Rotationsmatrix

Standardbasis = ursprüngliches Koordinatensystem. Die neue Basis ist ein Koordinatensystem, das durch die Hauptkomponenten PC1, PC2, … beschrieben wird.
Wir haben am Anfang ganz viele Punkte, die durch die Koordinaten der Standardbasis beschrieben werden.

Zweidimensional	Dreidimensional	Vierdimensional	…
Punkt 1 = (X_1, X_2)	Punkt 1 = (X_1, X_2, X_3)	Punkt 1 = (X_1, X_2, X_3, X_4)	
Punkt 2 = (X_1, X_2)	Punkt 2 = (X_1, X_2, X_3)	Punkt 2 = (X_1, X_2, X_3, X_4)	

…

Wir wollen diese Punkte P_1, P_2, … neu durch die Hauptkomponenten beschreiben. Sie sehen also nachher so aus: P_1 = (Z_1, Z_2, Z_3)
2D: P_1 = (X_1, X_2) → P_1 = (Z_1, Z_2) 3D: P_1 = (X_1, X_2, X_3) → P_1 = (Z_1, Z_2, Z_3) 4D: …

Die Umformung wird mittels Skalarprodukt berechnet. Dabei werden die Koordinaten vom Punkt P_1 der Reihe nach mit den Vektoren PC1, PC2, PC3, … verrechnet. Schlussendlich erhält man die neuen Koordinaten.

Anm.: Das Skalarprodukt von einem Punkt P und einer Hauptkomponente PC beschreibt die Projektion des Punktes P auf die Hauptkomponente PC.

Beispiel: Wir machen einen Basiswechsel vom Basissystem zu einem System mit den Hauptkomponenten PC1 = (ϕ_{11} , ϕ_{21}) = ($\frac{1}{\sqrt{2}}$, $\frac{1}{\sqrt{2}}$) und PC2 = (ϕ_{12} , ϕ_{22}) = ($-\frac{1}{\sqrt{2}}$, $\frac{1}{\sqrt{2}}$) .
Wir wollen nun die neuen Koordinaten berechnen für den Punkt P_1 = (X_1, X_2) = (2, 5).

$Z_1 = P_1 * PC1 = (X_1, X_2) * (\phi_{11}, \phi_{21}) = X_1 * \phi_{11} + X_2 * \phi_{21} = 2*\frac{1}{\sqrt{2}} + 5*\frac{1}{\sqrt{2}} \approx 4.9$

$Z_2 = P_1 * PC2 = (X_1, X_2) * (\phi_{12}, \phi_{22}) = X_2 * \phi_{12} + X_2 * \phi_{22} = 2*(-\frac{1}{\sqrt{2}}) + 5*\frac{1}{\sqrt{2}} \approx 2.1$

→ $P_1 = (Z_1, Z_2) = (4.9, 2.1)$

Scores

Scores sind die Koordinaten bezüglich der Hauptkomponenten.
(4.9, 2.1) sind also die Scores des Punktes P_1. Die Scores ändern die Vorzeichen je nach dem, in welche Richtung (positiv/negativ) die Vektoren PC1, PC2, PC3, … schauen.

Rotationsmatrix

Statt bei jedem Punkt P_i für jede Hauptkomponente einzeln zu rechnen, kann man auch eine Rotationsmatrix machen. Die Spalten der Rotationsmatrix sind die loadings (erste Spalte = loadings von PC1, zweite Spalte = loadings von PC2, etc.)

$$\phi = \begin{matrix} PC1 & PC2 \\ \end{matrix} \begin{pmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{pmatrix}$$

Basiswechsel von der Standardbasis zur PC-Basis für den Punkt $P_1 = (Z_1 , Z_2)$:

$$\phi^{-1} = \begin{pmatrix} \phi_{11} & \boldsymbol{\phi_{21}} \\ \boldsymbol{\phi_{12}} & \phi_{22} \end{pmatrix} * \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix}$$ *[ϕ wird an der Hauptdiagonalen gespiegelt = ϕ^{-1}]*

Basiswechsel von der PC-Basis zurück zur Standardbasis:

$$\phi = \begin{pmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{pmatrix} * \begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix} = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

PC's finden mithilfe von Numerik

Singulärwertzerlegung der Kovarianzmatrix (alternativ Eigenwertzerlegung)
R-Studio: Funktion prcomp() = Singulärwertzerlegung der Kovarianzmatrix. [princomp() macht Eigenwertzerlegung, ist eine schlechtere Alternative zu prcomp()]

pr.out **<- prcomp(*dat*)** → default: Daten werden zentriert, aber nicht skaliert
pr.out **<- prcomp(*dat*, scale = TRUE)** → Daten skalieren
pr.out **<- prcomp(*dat*, center = FALSE)** → Daten *nicht* zentrieren

str(*pr.out*) → output zeigt Struktur von pr.out (≈ Spaltennamen).
pr.out**$center** → Koordinaten vom ehemaligen Zentrum
*pr.out***$rotation** → gibt die *loadings* aus = Richtung der PC's bzgl. Standardbasis

```
                PC1        PC2
[1,] -0.9563643  0.2921768
[2,] -0.2921768 -0.9563643
```

Interpretation: Bsp: [1,] beschreibt die Kraft und [2,] die Geschwindigkeit
→ PC1 beschreibt vor allem die Kraft, PC2 vor allem die Geschwindigkeit. Grund: loadings von PC1 sind bezüglich [1,] und von PC2 bezüglich [2,] viel grösser sind als die anderen loadings.
→ Wenn man bei einer Person P die Kraft=2 und die Geschwindigkeit=4 misst, dann sind die Koordinaten dieser Person bezüglich PC1 = 2*-0.956 + 4*-0.292 = -3.08. Die Koordinaten bezüglich PC2 sind analog = 2*0.292 + 4*-0.956 = 0.20.

*pr.out***$x** → gibt die *scores* aus = Koordinaten der Punkte bezgl. PC-Basis
→ Output: Zeilen = Punkte P_1, P_2, P_3, …; Spalten: PC1, PC2, PC3, …
*pr.out**$x[2,]*** → *gibt die Koordinate von der 2. Zeile aus*
*pr.out**$x[,2]*** → *gibt die scores der Datenpunkte 1,2,3,… bzgl. PC2 aus*

cor(pr.out$x[,1], pr.out$x[,2]) → *Korrelation zw. PC1 und PC2 bzgl. PC-Basis*

summary(*pr.out*) → output:

```
> summary(pr.out)
Importance of components:
                          PC1     PC2
Standard deviation     3.2269 0.82619
Proportion of Variance 0.9385 0.06152
Cumulative Proportion  0.9385 1.00000
```

- Standard Derivation= absolute Varianz (PC1 > PC2 >…)
- Proportion of Variance = %Varianz, die mit PC1/PC2 /… erklärt wird
- Cumulative Proportion = relative Varianz (%), die mit PC1/ PC1+2/ PC1+2+3/… erklärt wird [bei letzter PC ist sie immer 100%]

biplot(*pr.out*, scale = 0) → Projektion vom x-Dimensionalen Raum in 2D

Scree-Plot: Dimensionen verhindern

Ziel: möglichst viel Varianz in den Daten zu erfassen. Die Varianz nimmt entlang der PC's immer weiter ab. Wie viele Hauptkomponenten PC's behalten ist sinnvoll?
Faustregel: Meistens behält man so viele PC's, dass 80% der Varianz erklärt wird.
Vorgehen: Kumulative Varianzen anschauen: Wo ist zum ersten Mal ≥0.8? → Bis und mit dieser PC behalten.

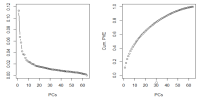
1. Variante:
summary(*pr.out*) → Schaue bei *Cumulative Proportion*, wo es ≥0.8 wird

2. Variante:
sum **<- summary(*pr.out*)**
*sum***$importance** → Gleicher Output wie bei *summary*(pr.out)
sum$importance[2,] → gibt 2. Zeile aus = unkumulative Varianzen der PC's
sum$importance[2,1] → gibt die unkumulative Varianz von PC1 aus
sum$importance[3,] → gibt die 3. Zeile aus = kumulative Varianzen der PC's
sum$importance[3,1] → gibt die kumulative Varianz von PC1 aus

which(sum$importance[3,] > *0.95*)[1] →#PC's die für 95% Varianz g braucht werden

Erstelle einen Scree-Plot:

```
pve <- sum$importance[2,]
cpve <- sum$importance[3,]
par(mfrow = c(1,2))
plot(pve, xlab = "PCs" , ylab = "PVE" , ylim = c(0,1), type = "b")
plot(cpve, xlab = "PCs" , ylab = "cum.PVE" , ylim = c(0,1), type = "b")
```



links: Varianz erklärt mit PC1/PC2/PC3/…
rechts: kumulative Varianz erklärt mit PC1 / PC1+2 / PC1+2+3 / PC1+2+3+4 / …

Weniger Dimensionen
datNeu **<- pr.out$x[1:*10*]** → nimmt PC1 bis PC10 in die Datei *datNeu*
Wichtig: obwohl Dimensionalität kleiner ist, braucht es immer noch alle Messungen!