

Computational Statistics

Prof. Dr. Dominik Liebl

Inhaltsverzeichnis

Informationen	5
1 Der Expectation Maximization (EM) Algorithmus	7
1.1 Motivation: Clusteranalyse mit Hilfe Gaußscher Mischverteilungen	8
1.2 Der EM Algorithmus zur ML-Schätzung Gaußscher Mischverteilungen	12
1.3 Der alternative (wahre) Blick auf den EM Algorithmus	19
1.4 Das Große Ganze	21
2 Regressionsmodelle im Kontext des Maschinellen Lernens	23
2.1 Das allgemeine Regressionsmodell	27
2.2 Das multivariate lineare Regressionsmodell	30
2.3 Modellauswahl	33
2.4 Anwendung: Vorhersage des Benzinverbrauchs	37
2.5 Ende	40

Informationen

Dies ist das Skript zur Vorlesung *Computational Statistik*

Vorlesungszeiten

RCodes

Die RCodes zu den einzelnen Kapiteln können hier heruntergeladen werden:
RCodes

Lesecke

Folgende *frei zugängliche* Lehrbücher enthalten Teile dieses Kurses. In den jeweiligen Kapiteln, werde ich auf die einzelnen Bücher verweisen.

- Pattern Recognition and Machine Learning (by Christopher Bishop)
- An Introduction to Statistical Learning, with Applications in R (by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani).
- Statistical Learning with Sparsity: the Lasso and Generalizations (by Trevor Hastie, Robert Tibshirani and Martin Wainwright).
- Elements of Statistical Learning: Data mining, Inference and Prediction (by Trevor Hastie, Robert Tibshirani and Jerome Friedman).
- Computer Age Statistical Inference: Algorithms, Evidence and Data Science (by Bradley Efron and Trevor Hastie)

Wochentag	Uhrzeit	Hörsaal
Dienstag	9:15-10:45	Online-Vorlesung
Freitag	8:30-10:00	Online-Vorlesung

Florence Nightingale

Das Logo zu diesem Skript stammt von einer Briefmarke zur Erinnerung an die Krankenschwester und inspirierende Statistikerin, Florence Nightingale. Nightingale war die Begründerin der modernen westlichen Krankenpflege und Pionierin der visuellen Datenanalyse. Sie nutzte statistische Analysen, um Missstände in Kliniken zu erkennen und diese dann auch nachweislich abzustellen. Sie ist die erste Frau, die in die britische Royal Statistical Society aufgenommen wurde; später erhielt sie auch die Ehrenmitgliedschaft der American Statistical Association.

Kapitel 1

Der Expectation Maximization (EM) Algorithmus

Der EM Algorithmus wird häufig verwendet, um komplizierte Maximum Likelihood Schätzprobleme zu vereinfachen bzw. überhaupt erst möglich zu machen. In diesem Kapitel stellen wir den EM Algorithmus zur Schätzung von Gaußschen Mischverteilungen vor, da der EM Algorithmus hier wohl seine bekannteste Anwendung hat. Bereits die originale Arbeit zum EM Algorithmus (Dempster et al., 1977) beschäftigt sich mit der Schätzung von Gaußschen Mischverteilungen.

Mögliche Anwendungen von Gaußschen Mischverteilungen:

- Generell: Auffinden von Gruppierungen (zwei oder mehr) in den Daten (**Clusteranalyse**). Zum Beispiel:
 - Automatisierte Videobearbeitungen (z.B. Bildeinteilungen in Vorder- und Hintergrund)
 - Automatisierte Erkennung von Laufstilen
 - etc.

Lernziele für dieses Kapitel

Sie können ...

- ein **Anwendungsfeld** des EM Algorithmuses **benennen**.

8KAPITEL 1. DER EXPECTATION MAXIMIZATION (EM) ALGORITHMUS

- die **Probleme** der klassischen Maximum Likelihood Methode zur Schätzung von Gaußschen Mischverteilungen **benennen und erläutern**.
- die **Grundidee** des EM Algorithmuses **erläutern**.
- den **EM Algorithmus** zur Schätzung von Gaußschen Mischverteilungen **anwenden**.
- die **Grundidee** der Vervollständigung der Daten durch latente Variablen **erläutern**.

Begleitlektüre(n)

Zur Vorbereitung der Klausur ist es grundsätzlich ausreichend dieses Kapitel durchzuarbeiten - aber Lesen hat ja noch nie geschadet. Dieses Kapitel basiert hauptsächlich auf:

- Kapitel 9 in **Pattern Recognition and Machine Learning** (Bishop, 2006). Die pdf-Version des Buches ist frei erhältlich: **pdf-Version**

Weiteren guten Lesestoff zum EM Algorithmus gibt es z.B. hier:

- Kapitel 8.5 in **Elements of Statistical Learning: Data Mining, Inference and Prediction** (Hastie et al., 2009). Die pdf-Version des Buches ist frei erhältlich: **pdf-Version**

R-Pakete für diese Kapitel

Folgende R-Pakete werden für dieses Kapitel benötigt:

```
pkgs <- c("tidyverse",      # Die tidyverse-Pakete
          "palmerpenguins", # Pinguin-Daten
          "scales",         # Transparente Farben: alpha()
          "RColorBrewer",   # Hübsche Farben
          "mclust",         # Schätzung/Verwendung
                           # Gaußschen Mischverteilungen
          "MASS")           # Erzeugung von Zufallszahlen aus
                           # einer multiv. Normalverteilung

install.packages(pkgs)
```

1.1 Motivation: Clusteranalyse mit Hilfe Gaußscher Mischverteilungen

Als Datenbeispiel verwenden wir die `palmerpenguins` Daten (Horst et al., 2020).

1.1. MOTIVATION: CLUSTERANALYSE MIT HILFE GAUßSCHER MISCHVERTEILUNGEN⁹

Diese Daten stammen aus Vermessungen von Pinguinpopulationen auf dem Palmer-Archipel (Antarktische Halbinsel). Pinguine sind oft schwer von einander zu unterscheiden. Wir werden versuchen, mit Hilfe einer Gaußschen Mischverteilung Gruppierungen in den Pinguindaten (Flossenlänge) zu finden. Um solche Mischverteilungen schätzen zu können, führen wir den EM Algorithmus ein.

Der folgende Code-Chunck bereitet die Daten auf.

Achtung: Wir haben zwar die Information zu den verschiedenen Pinguin-Arten (`Penguin_Art`) tun aber im Folgenden so, also ob wir diese Information nicht kennen würden. Wir wollen alleine auf Basis der Flossenlängen (`Penguin_Flosse`) die Gruppenzugehörigkeiten per Clusteranalyse bestimmen. (Im Nachhinein können wir dann mit Hilfe der Daten in `Penguin_Art` prüfen, wie gut unsere Clusteranalyse ist.)

```
library("palmerpenguins") # Pinguin-Daten
library("RColorBrewer")   # Hübsche Farben
library("scales")         # Für transparente Farben: alpha()

col_v <- RColorBrewer::brewer.pal(n = 3, name = "Set2")

## Vorbereitung der Daten:
Pinguine <- palmerpenguins::penguins %>% # Pinguin-Daten
  tidyr::as_tibble() %>%                # Datenformat: 'tibble'-dataframe
  dplyr::filter(species!="Adelie") %>%  # Pinguin-Art 'Adelie' löschen
  droplevels() %>%                     # Lösche das nicht mehr benötigte Adelie-Level
  tidyr::drop_na() %>%                 # NAs löschen
  dplyr::mutate(Art = species,          # Variablen umbenennen
                Flosse = flipper_length_mm) %>%
  dplyr::select(Art, Flosse)          # Variablen auswählen

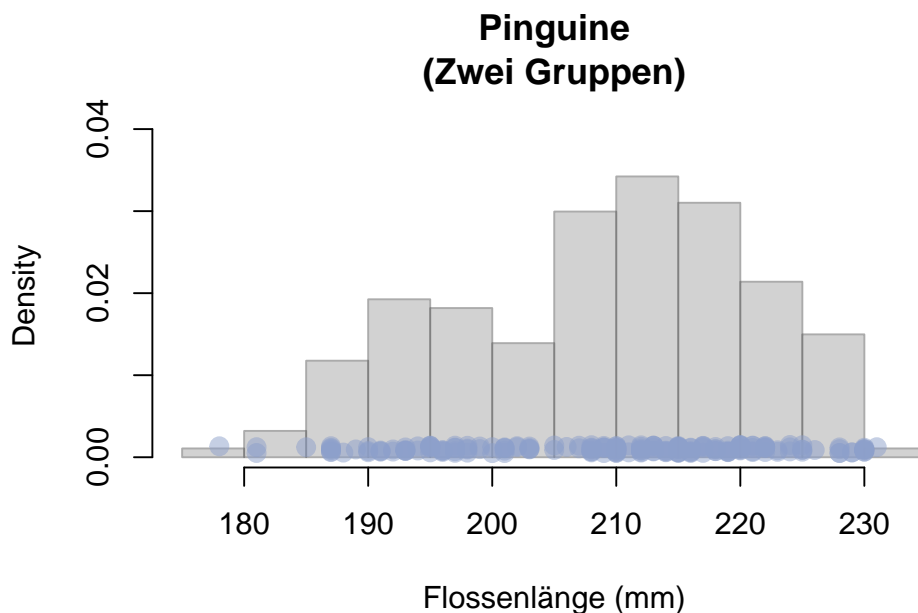
##
n <- nrow(Pinguine)                  # Stichprobenumfang

## Variable 'Penguin_Art' aus Pinguine-Daten "herausziehen"
Penguin_Art <- dplyr::pull(Pinguine, Art)

## Variable 'Penguin_Flosse' aus Pinguine-Daten "herausziehen"
Penguin_Flosse <- dplyr::pull(Pinguine, Flosse)

## Plot
## Histogramm:
hist(x = Penguin_Flosse, freq = FALSE,
     xlab="Flossenlänge (mm)", main="Pinguine\n(Zwei Gruppen)",
```

```
col=gray(.65,.5), border=gray(.35,.5), ylim=c(0.0003, 0.039))
## Stripchart hinzufügen:
stripchart(x = Penguin_Flosse, method = "jitter",
          jitter = .0005, at = .001,
          pch = 21, col=alpha(col_v[3],.5),
          bg=alpha(col_v[3],.5), cex=1.3, add = TRUE)
```



Das Clusterverfahren basierend auf Gaußschen Mischverteilungen:

1. Gaußsche Mischverteilung (**per EM Algorithmus**) schätzen
2. Die Datenpunkte x_i derjenigen Gruppe zuordnen, welche die „**a-posteriori-Wahrscheinlichkeit**“ maximiert (siehe Abbildung 1.1)

Abbildung 1.1 zeigt das Resultat einer Clusteranalyse basierend auf einer Mischverteilung zweier gewichteter Normalverteilungen. Cluster-Ergebnis: 95% der Pinguine konnten richtig zugeordnet werden - lediglich auf Basis ihrer Flossenlängen.

Mit Hilfe der folgenden R-Codes kann die obige Clusteranalyse und die Ergebnisgrafik repliziert werden:

```
## mclust R-Paket:
## Clusteranalyse mit Hilfe von Gaußschen Mischmodellen
```

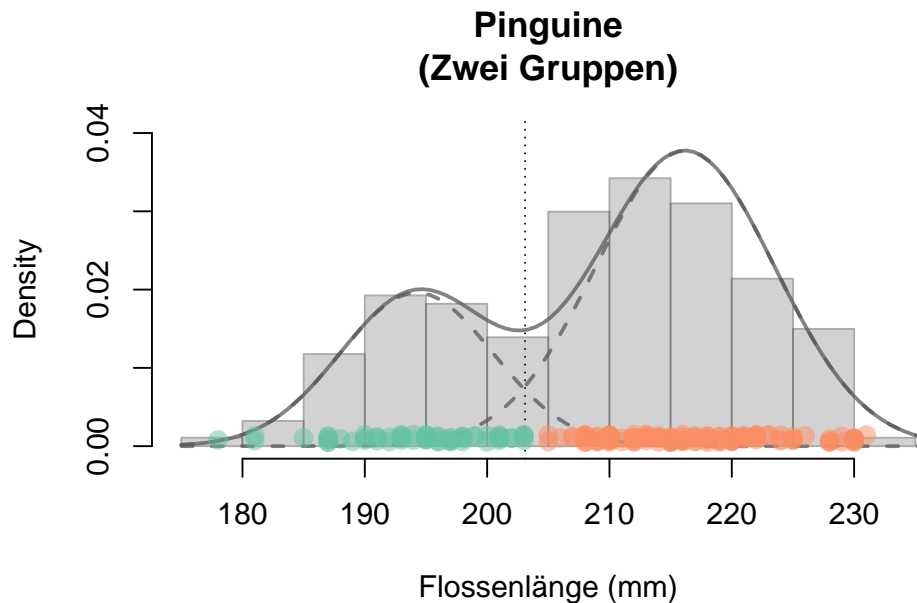


Abbildung 1.1: Clusteranalyse basierend auf einer Mischverteilung mit zwei gewichteten Normalverteilungen.

```
suppressMessages(library("mclust"))

## Anzahl der Gruppen
G <- 2

## Schätzung des Gaußschen Mischmodells (per EM Algorithmus)
## und Clusteranalyse
mclust_obj <- mclust::Mclust(data = Penguin_Flosse, G=G,
                             modelNames = "V",
                             verbose = FALSE)

# summary(mclust_obj)
# str(mclust_obj)

## Geschätzte Gruppen-Zuordnungen (Cluster-Resultat)
class <- mclust_obj$classification

## Anteil der korrekten Zuordnungen:
# cbind(class, Penguin_Art)
round(sum(class == as.numeric(Penguin_Art))/n, 2)

## Geschätzte Mittelwerte
```

```

mean_m <- t(mclust_obj$parameters$mean)

## Geschätzte Varianzen (und evtl. Kovarianzen)
cov_l <- list("Cov1" = mclust_obj$parameters$variance$sigma_sq[1],
             "Cov2" = mclust_obj$parameters$variance$sigma_sq[2])

## Geschätzte Gewichte (a-priori-Wahrscheinlichkeiten)
prop_v <- mclust_obj$parameters$pro

## Auswerten der Gaußsche Mischung-Dichtefunktion
np <- 100 # Anzahl der Auswertungspunkte
xxd <- seq(min(Penguin_Flosse)-3, max(Penguin_Flosse)+5, length.out = np)
## Mischungs-Dichte
yyd <- dnorm(xxd, mean_m[1], sqrt(cov_l[[1]]))*prop_v[1] +
      dnorm(xxd, mean_m[2], sqrt(cov_l[[2]]))*prop_v[2]
## Einzel-Dichten
yyd1 <- dnorm(xxd, mean_m[1], sqrt(cov_l[[1]]))*prop_v[1]
yyd2 <- dnorm(xxd, mean_m[2], sqrt(cov_l[[2]]))*prop_v[2]

## Plot
hist(x = Penguin_Flosse, xlab="Flossenlänge (mm)", main="Pinguine\n(Zwei Gruppen)",
     col=gray(.65,.5), border=gray(.35,.5), freq = FALSE, ylim=c(0, 0.04))
lines(x = xxd, y=yyd, lwd=2, col=gray(.35,.75))
lines(x = xxd, y=yyd1, lwd=2, col=gray(.35,.75), lty=2)
lines(x = xxd, y=yyd2, lwd=2, col=gray(.35,.75), lty=2)
stripchart(Penguin_Flosse[class==1], method = "jitter", jitter = .0005, at = .001,
           pch = 21, col=alpha(col_v[1],.5), bg=alpha(col_v[1],.5), cex=1.3, add = TRUE)
stripchart(Penguin_Flosse[class==2], method = "jitter", jitter = .0005, at = .001,
           pch = 21, col=alpha(col_v[2],.5), bg=alpha(col_v[2],.5), cex=1.3, add = TRUE)

```

1.2 Der EM Algorithmus zur ML-Schätzung Gaußscher Mischverteilungen

1.2.1 Gaußsche Mischmodelle (GMM)

Eine Zufallsvariable X , die einer Gauschen Mischverteilung folgt, bezeichnen wir als

$$X \sim \mathcal{N}_{mix}(G, \pi, \mu, \sigma)$$

Die dazugehörige Dichtefunktion einer Gaußschen Mischverteilung ist

folgendermaßen definiert:

$$f_G(x|\pi, \mu, \sigma) = \sum_{g=1}^G \pi_g f(x|\mu_g, \sigma_g) \quad (1.1)$$

- **Gewichte:** $\pi = (\pi_1, \dots, \pi_G)$ mit $\pi_g > 0$ und $\sum_{g=1}^G \pi_g = 1$
- **Mittelwerte:** $\mu = (\mu_1, \dots, \mu_G)$ mit $\mu_g \in \mathbb{R}$
- **Standardabweichungen:** $\sigma = (\sigma_1, \dots, \sigma_G)$ mit $\sigma_g > 0$
- **Normalverteilung der Gruppe $g = 1, \dots, G$:**

$$f(x|\mu_g, \sigma_g) = \frac{1}{\sqrt{2\pi}\sigma_g} \exp\left(-\frac{1}{2}\left(\frac{x - \mu_g}{\sigma_g}\right)^2\right)$$

- **Unbekannte Parameter:** π , μ und σ

1.2.2 Maximum Likelihood (ML) Schätzung

Man kann versuchen die unbekannten Parameter $\pi = (\pi_1, \dots, \pi_G)$, $\mu = (\mu_1, \dots, \mu_G)$ und $\sigma = (\sigma_1, \dots, \sigma_G)$ eines Gaußschen Mischmodells klassisch mit Hilfe der Maximum Likelihood Methode zu schätzen.

Ich sag's gleich: Der Versuch wird scheitern.

Wiederholung der Grundidee der ML-Schätzung:

- **Annahme:** Die Daten $\mathbf{x} = (x_1, \dots, x_n)$ sind eine Realisation einer einfachen (also i.i.d.) Zufallsstichprobe (X_1, \dots, X_n) mit

$$X_i \sim \mathcal{N}_{mix}(G, \pi, \mu, \sigma)$$

für alle $i = 1, \dots, n$.

Die Daten $\mathbf{x} = (x_1, \dots, x_n)$ „kennen“ also die unbekannten Parameter π , μ und σ und wir müssen ihnen diese Informationen „nur noch“ entlocken.

- **Schätz-Idee:** Wähle π , μ und σ so, dass $f_G(\cdot|\pi, \mu, \sigma)$ „optimal“ zu den beobachteten Daten \mathbf{x} passt.
- **Umsetzung der Schätz-Idee:** Maximiere (bzgl. π , μ und σ) die Likelihood Funktion

$$\mathcal{L}(\pi, \mu, \sigma|\mathbf{x}) = \prod_{i=1}^n f_G(x_i|\pi, \mu, \sigma)$$

Bzw. maximiere die Log-Likelihood Funktion (einfachere Maximierung)

$$\begin{aligned}\ln(\mathcal{L}(\pi, \mu, \sigma|\mathbf{x})) &= \ell(\pi, \mu, \sigma|\mathbf{x}) = \sum_{i=1}^n \ln(f_G(x_i|\pi, \mu, \sigma)) \\ &= \sum_{i=1}^n \ln\left(\sum_{g=1}^G \pi_g f(x_i|\mu_g, \sigma_g)\right)\end{aligned}$$

Beachte: Die Maximierung muss die Parameterrestriktionen in (1.1) berücksichtigen ($\sigma_g > 0$ und $\pi_g > 0$ für alle $g = 1, \dots, G$ und $\sum_{g=1}^G \pi_g = 1$).

- Die maximierenden Parameterwerte $\hat{\pi}$, $\hat{\mu}$ und $\hat{\sigma}$ sind die **ML-Schätzer**. Das kann man so ausdrücken:

$$(\hat{\pi}, \hat{\mu}, \hat{\sigma}) = \arg \max_{\pi, \mu, \sigma} \ell(\pi, \mu, \sigma|\mathbf{x})$$

Probleme mit Singularitäten bei Numerische Lösungen: Versucht man obiges Maximierungsproblem numerisch mit Hilfe des Computers zu lösen, wird man schnell merken, dass die Ergebnisse höchst instabil, unplausibel und wenig vertrauenswürdig sind. Der Grund für diese instabilen Schätzungen sind Probleme mit Singularitäten: Für echte GMMs ($G > 1$) treten während einer numerischen Maximierung sehr leicht Probleme mit Singularitäten auf. Dies geschieht immer dann, wenn eine der Normalverteilungskomponenten versucht den ganzen Datensatz \mathbf{x} zu beschreiben und die andere(n) Normalverteilungskomponente(n) versuchen lediglich einzelne Datenpunkte zu beschreiben. Eine Gaußsche Dichtefunktion, die sich um einen einzigen Datenpunkt x_i konzentriert (d.h. $\mu_g = x_i$ und $\sigma_g \rightarrow 0$) wird dabei sehr große Werte annehmen (d.h. $f(x_i|\mu_g = x_i, \sigma_g) \rightarrow \infty$ für $\sigma_g \rightarrow 0$) und so die Log-Likelihood auf unerwünschte Weise maximieren (siehe Abbildung 1.2). Solch **unerwünschte, triviale Maximierungslösungen** führen i.d.R. zu unplausiblen Schätzergebnissen.

Analytische Lösung: Es ist zwar etwas mühsam, aber man kann versuchen die Log-Likelihood analytisch zu maximieren. Tut man sich das an, kommt man zu folgenden Ausdrücken:

$$\begin{aligned}\hat{\pi}_g &= \frac{1}{n} \sum_{i=1}^n p_{ig} \\ \hat{\mu}_g &= \sum_{i=1}^n \frac{p_{ig}}{(\sum_{j=1}^n p_{jg})} x_i \\ \hat{\sigma}_g &= \sqrt{\sum_{i=1}^n \frac{p_{ig}}{(\sum_{j=1}^n p_{jg})} (x_i - \hat{\mu}_g)^2}\end{aligned}$$

für $g = 1, \dots, G$.

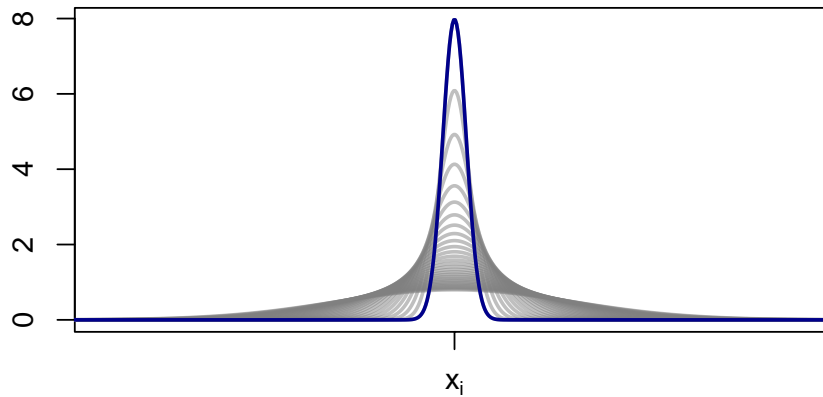


Abbildung 1.2: Normalverteilung mit $\mu_g = x_i$ für $\sigma_g \rightarrow 0$.

Die Herleitung der Ausdrücke für $\hat{\mu}_g$, $\hat{\sigma}_g$ und $\hat{\pi}_g$ ist wirklich etwas lästig (mehrfache Anwendungen der Kettenregel, Produktregel, etc., sowie Anwendung des Lagrange-Multiplikator Verfahrens zur Optimierung unter Nebenbedingungen) aber machbar. In einer der Übungsaufgaben dürfen Sie den Ausdruck für $\hat{\mu}_g$ herleiten.

Aber: Diese Ausdrücke für $\hat{\pi}_g$, $\hat{\mu}_g$ und $\hat{\sigma}_g$ hängen von den **unbekannten** Parametern $\pi = (\pi_1, \dots, \pi_G)$, $\mu = (\mu_1, \dots, \mu_G)$ und $\sigma = (\sigma_1, \dots, \sigma_G)$, denn:

$$p_{ig} = \frac{\pi_g f(x_i | \mu_g, \sigma_g)}{f_G(x_i | \pi, \mu, \sigma)}$$

für $i = 1, \dots, n$ und $g = 1, \dots, G$. Die Ausdrücke für $\hat{\pi}_g$, $\hat{\mu}_g$, und $\hat{\sigma}_g$ erlauben also keine direkte Schätzung der unbekannten Parameter π_g , μ_g und σ_g .

Lösung: Der EM Algorithmus

1.2.3 Der EM Algorithmus für GMMs

Die Ausdrücke für $\hat{\pi}_g$, $\hat{\mu}_g$ und $\hat{\sigma}_g$ legen jedoch ein einfaches, iteratives ML-Schätzverfahren nahe: Nämlich einer alternierenden Schätzung von p_{ig} und $(\hat{\pi}_g, \hat{\mu}_g, \hat{\sigma}_g)$.

Der EM Algorithmus:

1. Setze Startwerte $\pi^{(0)}$, $\mu^{(0)}$ und $\sigma^{(0)}$
2. Für $r = 1, 2, \dots$

- **(Expectation)** Berechne:

$$p_{ig}^{(r)} = \frac{\pi_g^{(r-1)} f(x_i | \mu_g^{(r-1)}, \sigma_g^{(r-1)})}{f_G(x_i | \pi^{(r-1)}, \mu^{(r-1)}, \sigma^{(r-1)})}$$

- **(Maximization)** Berechne:

$$\hat{\pi}_g^{(r)} = \frac{1}{n} \sum_{i=1}^n p_{ig}^{(r)}, \quad \hat{\mu}_g^{(r)} = \sum_{i=1}^n \frac{p_{ig}^{(r)}}{\left(\sum_{j=1}^n p_{jg}^{(r)}\right)} x_i$$

$$\hat{\sigma}_g^{(r)} = \sqrt{\sum_{i=1}^n \frac{p_{ig}^{(r)}}{\left(\sum_{j=1}^n p_{jg}^{(r)}\right)} \left(x_i - \hat{\mu}_g^{(r)}\right)^2}$$

3. Prüfe Konvergenz:

- Stoppe, falls sich der Wert der maximierten Log-Likelihood Funktion, $\ell(\pi^{(r)}, \mu^{(r)}, \sigma^{(r)} | \mathbf{z})$, nicht mehr ändert.

Der obige pseudo-Code wird im Folgenden Code-Chunck umgesetzt:

```
library("MASS")
library("mclust")

## Daten:
x <- cbind(Penguin_Flosse) # Daten [n x d]-Dimensional.
d <- ncol(x)                # Dimension (d=1: univariat)
n <- nrow(x)                # Stichprobenumfang
G <- 2                      # Anzahl Gruppen

## Weitere Deklarationen:
llk      <- matrix(NA, n, G)
p        <- matrix(NA, n, G)
loglikOld <- 1e07
tol       <- 1e-05
it        <- 0
check     <- TRUE

## EM Algorithmus

## 1. Startwerte für pi, mu und sigma:
pi      <- rep(1/G, G)      # Naive pi
sigma   <- array(diag(d), c(d,d,G)) # Varianz = 1
mu      <- t(MASS::mvrnorm(G, colMeans(x), sigma[,1]*4) )

while(check){
```



```

## 2.a Expectation-Schritt
for(g in 1:G){
  p[,g] <- pi[g] * mclust::dmvnorm(x, mu[,g], sigma[,g])
}
p <- sweep(p, 1, STATS = rowSums(p), FUN = "/")

## 2.b Maximization-Schritt
par <- mclust::covw(x, p, normalize = FALSE)
mu <- par$mean
sigma <- par$S
pi <- colMeans(p)

## 3. Prüfung der Konvergenz
for(g in 1:G) {
  llk[,g] <- pi[g] * mclust::dmvnorm(x, mu[,g], sigma[,g])
}
loglik <- sum(log(rowSums(llk))) # aktueller max. Log-Likelihood Wert
##
diff <- abs(loglik - loglikOld)/abs(loglik) # Änderungsrate
loglikOld <- loglik
it <- it + 1
## Änderungsrate noch groß genug (> tol)?
check <- diff > tol
}

## Schätz-Resultate:
results <- matrix(c(pi, mu, sqrt(sigma)),
                  nrow = 3, ncol = 2, byrow = TRUE,
                  dimnames = list(
                    c("Gewichte", "Mittelwerte", "Standardabweichungen"),
                    c("Gruppe 1", "Gruppe 2")))
##
results %>% round(., 2)
#>
#> Gewichte      Gruppe 1 Gruppe 2
#> Mittelwerte   216.20  194.26
#> Standardabweichungen 7.32  6.27

```

Das Clusterverfahren basierend auf Gaußschen Mischverteilungen:

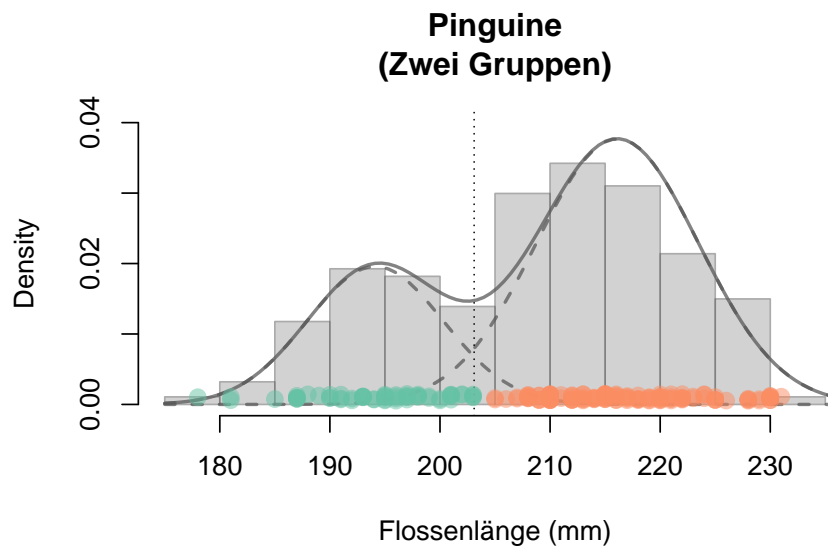
1. Gaußsche Mischverteilung (**per EM Algorithmus**) schätzen
2. Die Datenpunkte x_i derjenigen Gruppe g zuordnen, welche die **a-posteriori-Wahrscheinlichkeit** p_{ig} maximiert.

Das finale Schätzergebnis erlaubt es uns, Abbildung 1.1 zu replizieren.

18KAPITEL 1. DER EXPECTATION MAXIMIZATION (EM) ALGORITHMUS

```
## Auswerten der Gaußsche Mischungs-Dichtefunktion
np      <- 100 # Anzahl der Auswertungspunkte
xxd     <- seq(min(Penguin_Flosse)-3, max(Penguin_Flosse)+5, length.out = np)
## Mischungs-Dichte
yyd     <- dnorm(xx, mu[1,1], sqrt(sigma)[,1])*pi[1] +
          dnorm(xx, mu[1,2], sqrt(sigma)[,2])*pi[2]
## Einzel-Dichten
yyd1    <- dnorm(xx, mu[1,1], sqrt(sigma)[,1])*pi[1]
yyd2    <- dnorm(xx, mu[1,2], sqrt(sigma)[,2])*pi[2]

## Plot
hist(x = Penguin_Flosse, xlab="Flossenlänge (mm)", main="Pinguine\n(Zwei Gruppen)",
     col=gray(.65,.5), border=gray(.35,.5), freq = FALSE, ylim=c(0, 0.04))
lines(x = xxd, y=yyd, lwd=2, col=gray(.35,.75))
lines(x = xxd, y=yyd1, lwd=2, col=gray(.35,.75), lty=2)
lines(x = xxd, y=yyd2, lwd=2, col=gray(.35,.75), lty=2)
abline(v=203.1, lty=3)
stripchart(Penguin_Flosse[class==1], method = "jitter", jitter = .0005, at = .001,
           pch = 21, col=alpha(col_v[1],.5), bg=alpha(col_v[1],.5), cex=1.3, add = TRUE)
stripchart(Penguin_Flosse[class==2], method = "jitter", jitter = .0005, at = .001,
           pch = 21, col=alpha(col_v[2],.5), bg=alpha(col_v[2],.5), cex=1.3, add = TRUE)
```



1.3 Der alternative (wahre) Blick auf den EM Algorithmus

Der EM Algorithmus ermöglicht es Maximum Likelihood Probleme zu vereinfachen, indem man die Daten durch nicht beobachtete („latente“) Variablen vervollständigt. Dieses Prinzip ist der eigentliche wahre Beitrag des EM Algorithmus. Es ermöglicht die Lösung verschiedener Maximum Likelihood Probleme - wir bleiben aber hier bei der Schätzung von GMMs.

Zur Erinnerung: Wir haben es ja nicht geschafft, die Log-Likelihood Funktion

$$\ell(\pi, \mu, \sigma | \mathbf{x}) = \sum_{i=1}^n \ln \left(\sum_{g=1}^G \pi_g f(x_i | \mu_g, \sigma_g) \right)$$

direkt zu maximieren. Die $\ln(\sum_{g=1}^G [\dots])$ -Konstruktion macht einem hier das Leben schwer.

1.3.1 Vervollständigung der Daten

In unseren Pinguin-Daten gibt zwei Gruppen ($g \in \{1, 2\}$). Es gäbe also im Prinzip $G = 2$ -dimensionale Zuordnungsvektoren (z_{i1}, z_{i2}) mit

$$(z_{i1}, z_{i2}) = \begin{cases} (1, 0) & \text{falls Pinguin } i \text{ zu Gruppe } g = 1 \text{ gehört.} \\ (0, 1) & \text{falls Pinguin } i \text{ zu Gruppe } g = 2 \text{ gehört.} \end{cases}$$

Im Fall von $G > 2$ Gruppen:

$$(z_{i1}, \dots, z_{ig}, \dots, z_{iG}) = \begin{cases} (1, 0, \dots, 0) & \text{falls Datenpunkt } i \text{ zu Gruppe } g = 1 \text{ gehört.} \\ (0, 1, \dots, 0) & \text{falls Datenpunkt } i \text{ zu Gruppe } g = 2 \text{ gehört.} \\ \vdots & \\ (0, 0, \dots, 1) & \text{falls Datenpunkt } i \text{ zu Gruppe } g = G \text{ gehört.} \end{cases}$$

Die Zuordnungen z_{ig} können also die Werte $z_{ig} \in \{0, 1\}$ annehmen, wobei aber gelten muss, dass $\sum_{g=1}^G z_{ig} = 1$.

Beachte: Für jeden Datenpunkt i (jeder Pinguin i) gibt es nur **eine** Gruppe (daher $\sum_{g=1}^G z_{ig} = 1$). Dies ist eine wichtige Restriktion von GMMs, welche bei den Pinguindaten unproblematisch ist. Bei Anwendungen mit hierarchischen Gruppierungen ist dies aber evtl. problematisch.

Die Zuordnungen z_{ig} sind leider unbekannt (latent). Wir wissen aber trotzdem etwas über diese Zuordnungen. Die Gewichte π_1, \dots, π_G der Gaußschen Mischverteilung

$$f_G(x|\pi, \mu, \sigma) = \sum_{g=1}^G \pi_g f(x|\mu_g, \sigma_g),$$

geben die Anteile der Einzelverteilungen $f(\cdot|\mu_g, \sigma_g)$ an der Gesamtverteilung f_G an. Im Schnitt kommen also $\pi_g \cdot 100\%$ der Datenpunkte von Gruppe g , $g = 1, \dots, G$. Somit können wir die (latente) Zuordnung z_{ig} als eine Realisation der Zufallsvariablen Z_{ig} mit Wahrscheinlichkeitsfunktion

$$P(Z_{ig} = 1) = \pi_g$$

auffassen.

Wegen der Bedingung $\sum_{g=1}^G z_{ig} = 1$, gilt dass

$$Z_{ig} = 1 \quad \Rightarrow \quad Z_{i1} = 0, \dots, Z_{ig-1} = 0, Z_{ig+1} = 0, \dots, Z_{iG} = 0.$$

1.3.2 A-priori und A-posteriori Wahrscheinlichkeiten: π_g und p_{ig}

A-priori-Wahrscheinlichkeit π_g : Man bezeichnet die Wahrscheinlichkeiten π_g als die *a-priori-Wahrscheinlichkeiten*. Wenn wir nichts über die Flossenlänge von Pinguin i wissen, dann bleiben uns nur die a-priori-Wahrscheinlichkeiten:

„Mit Wahrscheinlichkeit $\pi_g = P(Z_{ig} = 1)$ gehört Pinguin i zu Gruppe g .“

A-posteriori-Wahrscheinlichkeit p_{ig} : Falls wir die Flossenlänge x_i von Pinguin i erfahren, können wir die a-priori-Wahrscheinlichkeiten mit Hilfe des **Satzes von Bayes** aktualisieren. Dies führt dann zur *a-posteriori-Wahrscheinlichkeit*:

„Mit Wahrscheinlichkeit $p_{ig} = P(Z_{ig=1}|X_i = x_i)$ gehört ein Pinguin i mit Flossenlänge x_i zu Gruppe g .“

Satz von Bayes:

$$\begin{aligned} p_{ig} &= \frac{\pi_g f(x_i|\mu_g, \sigma_g)}{f_G(x_i|\pi, \mu, \sigma)} \\ &= \frac{\overbrace{P(Z_{ig} = 1)}^{\text{„A-priori-Wahrs.“}} f(x_i|\mu_g, \sigma_g)}{f_G(x_i|\pi, \mu, \sigma)} = \overbrace{P(Z_{ig} = 1|X_i = x_i)}^{\text{„A-posteriori-Wahrs.“}} = p_{ig} \end{aligned}$$

1.3.3 Der (bedingte) Mittelwert: p_{ig}

Beachte: Die a-posteriori-Wahrscheinlichkeiten p_{ig} sind tatsächlich (bedingte) Erwartungswerte:

$$p_{ig} = 1 \cdot P(Z_{ig} = 1 | X_i = x_i) + 0 \cdot P(Z_{ig} = 0 | X_i = x_i) = E(Z_{ig} | X_i = x_i)$$

Damit ist die Berechnung von p_{ig} im **(Expectation)**-Schritt des EM Algorithmus (siehe Kapitel 1.2.3) also tatsächlich eine Erwartungswertberechnung.

1.4 Das Große Ganze

Hätten wir neben den Datenpunkten $\mathbf{x} = (x_1, \dots, x_n)$ auch die Gruppenzuordnungen $\mathbf{z} = (z_{11}, \dots, z_{nG})$ beobachtet, dann könnten wir folgende **Likelihood** ($\tilde{\mathcal{L}}$) bzw. **Log-Likelihood** ($\tilde{\ell}$) **Funktion** aufstellen:

$$\tilde{\mathcal{L}}(\pi, \mu, \sigma | \mathbf{x}, \mathbf{z}) = \prod_{i=1}^n \prod_{g=1}^G (\pi_g f(x_i | \mu_g, \sigma_g))^{z_{ig}}$$

$$\tilde{\ell}(\pi, \mu, \sigma | \mathbf{x}, \mathbf{z}) = \sum_{i=1}^n \sum_{g=1}^G z_{ig} \{ \ln(\pi_g) + \ln(f(x_i | \mu_g, \sigma_g)) \}$$

- Im Gegensatz zur ursprünglichen Log-Likelihood Funktion (ℓ), wäre die neue Log-Likelihood Funktion $\tilde{\ell}$ **einfach zu maximieren**, da hier keine Summe innerhalb der Logarithmusfunktion steckt, sodass wir direkt den Logarithmus der Normalverteilung berechnen können. Dies vereinfacht das Maximierungsproblem deutlich, da die Normalverteilung zur Exponentialfamilie gehört.
- Aber: Wir beobachten die Realisationen $\mathbf{z} = (z_{11}, \dots, z_{nG})$ nicht, sondern kennen lediglich die Verteilung der Zufallsvariablen $\mathbf{Z} = (Z_{11}, \dots, Z_{nG})$. Dies führt zu einer stochastischen Version der Log-Likelihood Funktion:

$$\tilde{\ell}(\pi, \mu, \sigma | \mathbf{x}, \mathbf{Z}) = \sum_{i=1}^n \sum_{g=1}^G Z_{ig} \{ \ln(\pi_g) + \ln(f(x_i | \mu_g, \sigma_g)) \}$$

- Von dieser können jedoch den bedingten Erwartungswert berechnen:

$$E(\tilde{\ell}(\pi, \mu, \sigma | \mathbf{x}, \mathbf{Z}) | X_i = x_i) = \sum_{i=1}^n \sum_{g=1}^G p_{ig} \{ \ln(\pi_g) + \ln(f(x_i | \mu_g, \sigma_g)) \}$$

1.4.1 Der EM Algorithmus: *Die abstrakte Version*

Der folgende EM Algorithmus unterscheidet sich wieder lediglich in der Notation von den oben bereits besprochenen Versionen (siehe Kapitel 1.2.3). Die hier gewählte Notation verdeutlicht, dass der **Expectation**-Schritt die zu maximierende Log-Likelihood Funktion aktualisiert und diese dann im **(Maximization)**-Schritt maximiert wird. Darüber hinaus ist die gewählte Notation abstrakt genug, um die Grundidee des EM Algorithmuses auf andere Maximum Likelihood Probleme zu übertragen. Im Folgenden wird der Parametervektor (π, μ, σ) der Einfachheit halber auch mit θ bezeichnet.

1. Setze Startwerte $\theta^{(0)} = (\pi^{(0)}, \mu^{(0)}, \sigma^{(0)})$
2. Für $r = 1, 2, \dots$

- **(Expectation)** Berechne:

$$\begin{aligned} Q(\theta, \theta^{(r-1)}) &= E_{\theta^{(r-1)}}(\tilde{\ell}(\pi, \mu, \sigma | \mathbf{x}, \mathbf{z}) | X_i = x_i) \\ &= \sum_{i=1}^n \sum_{k=1}^K p_{ig}^{(r-1)} \{ \ln(\pi_g) + \ln(f(x_i | \mu_g, \sigma_g)) \} \end{aligned}$$

- **(Maximization)** Berechne:

$$\theta^{(r)} = \arg \max_{\theta} Q(\theta, \theta^{(r-1)})$$

3. Prüfe Konvergenz:

- Stoppe, falls sich der Wert der maximierten Log-Likelihood Funktion, $Q(\theta^{(r)}, \theta^{(r-1)})$, nicht mehr ändert.

Ende

Dem gemeinen Pinguin ist der EM Algorithmus egal.

Kapitel 2

Regressionsmodelle im Kontext des Maschinellen Lernens

Lineare Regressionsmodelle gehören zu den erfolgreichsten statistischen Modellen, da sie

- vergleichsweise **einfach zu interpretieren** sind und
- zugleich **äußerst flexibel** sind.

In diesem Kapitel betrachten wir das multivariate (oder multiple) lineare Regressionsmodell als **Prädiktionsmodell** im Kontext des maschinellen Lernens.

Lernziele für dieses Kapitel

Sie können ...

- ein **Anwendungsfeld** des linearen Regressionsmodell als Prädiktionsmodell **benennen**.
- die **Probleme** der Auswahl eines geeigneten Prädiktionsmodells am Beispiel der Polynomregression **benennen und erläutern**.
- die **Grundidee** der Validierungsdaten-Methode **erläutern**.
- die **Grundidee** der k-fachen Kreuzvalidierung **erläutern**.

Begleitlektüren

Zur Vorbereitung der Klausur ist es grundsätzlich ausreichend dieses Kapitel durcharbeiten - aber Lesen hat ja noch nie geschadet. Empfehlenswerte weiterführende Literatur:

- Kapitel 3 in **An Introduction to Statistical Learning, with Applications in R** (James et al., 2021). Die pdf-Version des Buches ist hier frei erhältlich: www.statlearning.com
- Kapitel 3 in **Pattern Recognition and Machine Learning** (Bishop, 2006). Die pdf-Version des Buches ist frei erhältlich: [pdf-Version](#)
- Kapitel 6 in **Introduction to Econometrics with R** (Hanck et al., 2021). Freies Online-Buch: www.econometrics-with-r.org

R-Pakete und Datenbeispiel für dieses Kapitel

Folgende Pakete werden in diesem Kapitel benötigt.

- **tidyverse**: Viele nützliche Pakete zur Datenverarbeitung.
- **GGally**: Enthält die Funktion `ggpairs()` zur Erzeugung von Pairs-Plots
- **ISLR**: Enthält die Auto Daten

Falls noch nicht geschehen, müssen diese Pakete installiert und geladen werden:

```
## Installieren
install.packages("tidyverse")
install.packages("GGally")
install.packages("ISLR")
## Laden
library("tidyverse") # Viele nützliche Pakete zur Datenverarbeitung
library("GGally")    # Pairs-Plot
library("ISLR")      # Enthält die Auto-Daten
data(Auto)           # Auto-Daten abrufbar machen
```

Als Datenbeispiel für diese Kapitel betrachten wir den **Auto** Datensatz im R-Paket **ISLR**. Wir betrachten folgende Auswahl der Variablen im Datensatz **Auto**:

- **Zielvariable:**
 - **Verbrauch (km/Liter)**
- **Prädiktorvariablen:**
 - **Gewicht (kg)**: Schwerere Autos verbrauchen wahrscheinlich mehr.

- **Leistung (PS):** Höhere Leistung geht wohl auch mit höherem Verbrauch einher.
- **Hubraum (ccm):** Großer Hubraum ... höherer Verbrauch?

Achtung: Es gibt sicherlich noch weitere relevante Prädiktorvariablen. Obige Auswahl ist jedoch relativ einfach zu erheben und ermöglicht eventuell bereits eine **gute Prädiktion des Verbrauches** im Rahmen eines **Regressionsmodells**. Falls dem so ist, könnte unser Prädiktionsmodell dazu dienen, nach Auffälligkeiten bei den herstellerseitigen Verbrauchsangaben zu suchen. Besonders große Abweichung zwischen Modellprädiktion und Herstellerangabe wäre ein mögliches Indiz auf unlautere Zahlenschönung.

Aufbereitung der Daten:

```
## Auswahl und Aufbereitung der Variablen
Auto_df <- Auto %>%
  mutate(Verbrauch = mpg * (1.60934/3.78541), # Verbrauch (km/Liter)
         Gewicht    = weight * 0.45359,      # Gewicht (kg)
         PS         = horsepower,            # Pferdestärken (PS)
         Hubraum    = displacement * 2.54^3  # Hubraum (ccm)
        ) %>%
  dplyr::select("Verbrauch", "Gewicht", "PS", "Hubraum")

n <- nrow(Auto_df) # Stichprobenumfang
```

Insgesamt enthält der betrachtete Datensatz also fünf Variablen zu $n = 392$ verschiedenen Autos. Dies sind die ersten sechs Zeilen des Datensatzes:

Verbrauch (km/Liter)

Gewicht (kg)

Pferdestärken (PS)

Hubraum (ccm)

7.65

1589.38

130

5030.83

6.38

1675.11

165

5735.47

7.65

1558.54

150

5211.09

6.80

1557.17

150

4981.67

7.23

1564.43

140

4948.89

6.38

1969.03

198

7030.05

Um sich einen Überblick zu den Beziehungen zwischen den Variablen zu verschaffen, eignet sich ein **Pairs-Plot** sehr gut (siehe Abbildung 2.1):

```
ggpairs(Auto_df,
  upper = list(continuous = "density", combo = "box_no_facet"),
  lower = list(continuous = "points", combo = "dot_no_facet"))
```

Der Pairs-Plot veranschaulicht alle paarweisen Zusammenhänge zwischen den Variablen im Datensatz `Auto_df`. Uns interessieren hierbei in erster Linie die Zusammenhänge zwischen der Zielvariable **Verbrauch** und den **Prädiktorvariablen**:

- $Y = \text{Verbrauch}$ und ...
 - $G = \text{Gewicht}_i$: haben einen nicht linearen, negativen Zusammenhang.
 - $P = \text{PS}$: haben einen nicht linearen, negativen Zusammenhang.
 - $H = \text{Hubraum}$: haben einen nicht linearen, negativen Zusammenhang.

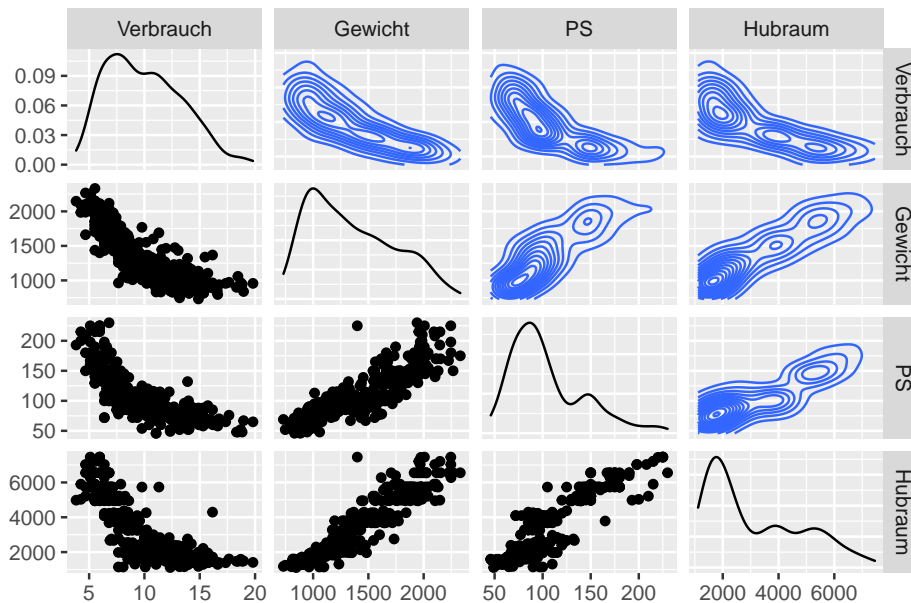


Abbildung 2.1: Pairs-Plot zur Veranschaulichung der paarweisen Zusammenhänge zwischen den Variablen.

2.1 Das allgemeine Regressionsmodell

Die einzelnen Prädiktorvariablen werden gerne kompakt zu einer multivariaten Prädiktorvariablen $X = (X_1, X_2, \dots, X_p)$ zusammengefasst; in unserem Benzinverbrauchsbeispiel also $X = (G, P, H, B)$. So lässt sich das **allgemeine Regressionsmodell** schreiben als

$$Y = f(X) + \varepsilon$$

wobei

- f den **systematischen Zusammenhang** zwischen der Zielvariable Y und den Prädiktorvariablen X beschreibt und
- ε ein **Fehlerterm** ist, der unabhängig von X ist und Mittelwert $E(\varepsilon) = 0$ Null hat.

Daraus ergibt sich folgender Zusammenhang zwischen der **allgemeinen Regressionsfunktion** f und dem bedingten Mittelwert von Y gegeben X :

$$E(Y|X) = f(X)$$

Die Funktion f beschreibt also den bedingten Mittelwert von Y gegeben X . Ziel ist es nun, die Regressionsfunktion f aus den Daten zu schätzen (lernen).

Achtung: Die Annahme der Unabhängigkeit zwischen ε und X kann in der Praxis verletzt sein. Die Verletzung dieser Unabhängigkeitsannahme erlaubt insbesondere keine kausale Interpretation der Ergebnisse, daher betrachtet die Literatur zur Kausalinferenz viele Möglichkeiten diese Unabhängigkeitsannahme durch eine weniger strikte Annahmen zu ersetzen. In der Literatur zur prädiktiven Inferenzen wird eine Verletzung der Unabhängigkeitsannahme weniger kritisch gesehen, da eine Prädiktion trotz verletzter Unabhängigkeitsannahme sehr gut sein kann. Eine schöne und gut lesbare Übersicht zu den Unterschieden zwischen der Kausalinferenz und der prädiktiven Inferenzen findet man, z.B., im Artikel To Explain or To Predict? (Shmueli, 2010).

Abbildung 2.2 zeigt ein Beispiel von 50 simulierten Daten (künstlich erzeugte Fake-Daten). Der Plot legt nahe, dass man das Einkommen mit Hilfe der Ausbildungsjahre vorhersagen kann. Normalerweise ist die wahre Funktion f , welche die Verbindung zwischen Y und X beschreibt, unbekannt und muss aus den Daten geschätzt werden. Da es sich hier um simulierte Daten handelt, können wir den Graph der Funktion f als blaue Linie plotten. Einige der 50 Beobachtungspunkte (X, Y) liegen über der Regressionsfunktion $f(X)$, andere darunter. Im Großen und Ganzen haben die Fehlerterme einen Mittelwert von Null.

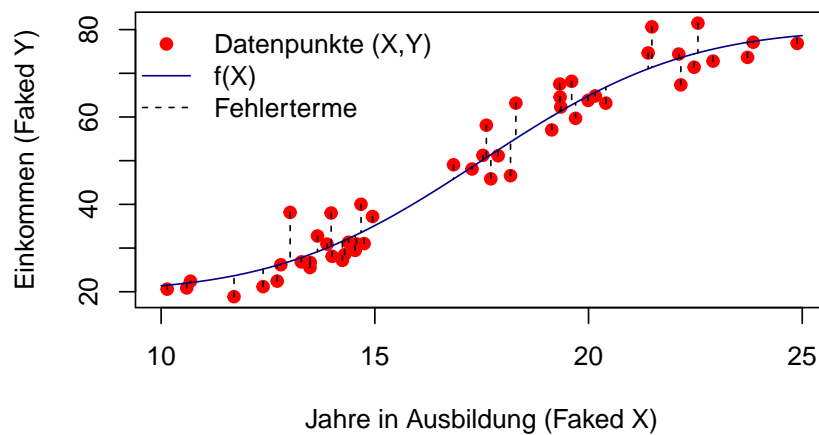


Abbildung 2.2: Simulierte (künstlich erzeugte) Daten zur Veranschaulichung einer allgemeinen, univariaten Regressionsbeziehung.

Abbildung 2.3 zeigt ein simuliertes Beispiel einer allgemeinen, bivariaten Regressionsbeziehung

$$Y = f(X) + \varepsilon \quad \text{mit} \quad X = (X_1, X_2).$$

2.1.1 Der Prädiktionsfehler (zwischen Y und \hat{Y})

In vielen Datenproblemen sind zwar die Prädiktorvariablen X bekannt (z.B. Gewicht, PS und Hubraum eines neuen Autos), aber die dazugehörige Zielvariable Y ist unbekannt. Da sich der Fehlerterm zu Null mittelt, lässt sich in solch einem Fall das unbekannte Y durch

$$\hat{Y} = \hat{f}(X)$$

vorhersagen, wobei

- \hat{f} für unsere Schätzung von f steht und
- \hat{Y} die Vorhersage von Y für gegebene Prädiktorvariablen X ist.

Die Genauigkeit der Vorhersage von \hat{Y} für Y hängt von zwei verschiedenen Prädiktionsfehlergrößen ab:

- **Reduzierbarer Prädiktionsfehler** aufgrund des Schätzfehlers in \hat{f} . Eine genauere Schätzung kann diesen Fehler reduzieren.
- **Nicht reduzierbarer Prädiktionsfehler** aufgrund des Fehlerterms ε . Das ist der Fehler, den wir selbst bei perfekter Schätzung von f nicht reduzieren können.

Der **nicht reduzierbare Fehler** ε enthält alle nicht messbaren und nicht gemessenen Variablen, die ebenfalls einen Einfluss auf Y haben. Und da wir diese Variablen nicht messen können, können wir sie auch nicht verwenden, um f zu schätzen.

Sei nun \hat{f} eine gegebene Schätzung von f und seien X gegebene Werte der Prädiktorvariablen welche die Vorhersage $\hat{Y} = \hat{f}(X)$ ergeben. Nehmen wir nun für einen Moment an, dass \hat{f} und X gegeben und fest (also nicht zufällig) sind, dann

$$\begin{aligned} E[(Y - \hat{Y})^2] &= E\left[\overbrace{(f(X) + \varepsilon)}^{=Y} - \overbrace{\hat{f}(X)}^{=\hat{Y}}\right]^2 \\ &= E\left[\left((f(X) - \hat{f}(X))^2 + 2(f(X) - \hat{f}(X))\varepsilon + \varepsilon^2\right)\right] \\ &= \underbrace{\left((f(X) - \hat{f}(X))^2\right)}_{\text{reduzierbar}} + \underbrace{\text{Var}(\varepsilon)}_{\text{nicht reduzierbar}} \end{aligned}$$

Der mittlere quadratische Prädiktionsfehler $E[(Y - \hat{Y})^2]$ lässt sich also in eine reduzierbare und eine nicht reduzierbare Fehlerkomponente zerlegen.

2.2 Das multivariate lineare Regressionsmodell

Um die allgemeine Regressionsfunktion $f(X) = E(Y|X)$ mit Hilfe der Daten zu schätzen (lernen), gibt es sehr viele verschiedenen Möglichkeiten. Eine der erfolgreichsten und am häufigsten verwendete Möglichkeit ist das **multivariaten lineare Regressionsmodell**. Dieses Modell ist die **strukturelle Modellannahme**, dass sich die unbekannte Regressionsfunktion f als lineare Funktion (linear in den Modellparametern $\beta_0, \beta_1, \dots, \beta_p$) schreiben lässt:

$$f(X) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p.$$

Unter dieser Modellannahme wird das allgemeine Regressionsmodell $Y = f(X) + \varepsilon$ zum multivariaten (multiplen) linearen Regressionsmodell

$$Y = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \varepsilon.$$

Zusammen mit der Annahme, dass ε unabhängig von X ist, und dass $E(\varepsilon) = 0$, können wir mit dieser Modellannahme den unbekannten bedingten Mittelwert $E(Y|X) = f(X)$ vereinfacht schreiben als

$$E(Y|X) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p.$$

Vorteile des **multivariaten linearen Regressionsmodells**:

- Anstatt eine gänzlich unbekannte Funktion f schätzen (lernen) zu müssen, muss man lediglich die unbekannten Parameterwerte $\beta_0, \beta_1, \dots, \beta_p$ schätzen.
- Die Modellstruktur ist **keine Black Box**, sondern gibt Aufschluss über die **assoziativen Zusammenhänge** zwischen den Prädiktorvariablen und der Zielvariablen.
- Die lineare Modellstruktur ist **extrem flexibel**, da Transformationen der Prädiktorvariablen grundsätzlich erlaubt sind.

Gerade die große Flexibilität linearer Modelle werden wir nutzen müssen, um die **nicht linearen Zusammenhänge** zwischen den Prädiktorvariablen und der Zielvariablen in unserem Benzinverbrauchsbeispiel berücksichtigen zu können (siehe Abbildung 2.1).

2.2.1 Schätzung

Wir wollen nun diejenige Funktion

$$\hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \dots + \hat{\beta}_p X_p$$

finden, sodass $Y \approx \hat{f}(X)$ für alle Datenpunkte (Y, X) .

Zur Berechnung von \hat{f} können wir die **beobachteten Daten** als **Trainingsdaten** verwenden:

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\} \quad \text{wobei} \quad x_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T.$$

Im Folgenden werden wir oft die Notation

$$x_{ij}, \quad i = 1, \dots, n, \quad j = 1, \dots, p$$

verwenden, um die j te Prädiktorvariable der i ten Beobachtung zu bezeichnen. Der Laufindex $j = 1, \dots, p$ repräsentiert die einzelnen Prädiktorvariablen (z.B. Verbrauch, Gewicht, Pferdestärken, und Hubraum im **Auto_df** Datensatz) und der Laufindex $i = 1, \dots, n$ repräsentiert die einzelnen Beobachtungen (z.B. gespeichert als Zeilen im **Auto_df** Datensatz).

Idee: Die Trainingsdaten $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ enthalten Information zum unbekannten Regressionsmodell f , da (so die Grundidee) die Daten von eben diesem Modell erzeugt wurden. Ziel ist also die unbekannte Regressionsfunktion f mit Hilfe der Trainingsdaten zu schätzen (erlernen).

Für jede mögliche Schätzung \hat{f} von f können wir die beobachteten Werte der Zielvariablen y_i mit den vorhergesagten Werten

$$\hat{y}_i = \hat{f}(x_i) = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \dots + \hat{\beta}_p x_{ip}$$

vergleichen, indem wir die **Residuen**

$$e_i = y_i - \hat{y}_i \quad i = 1, \dots, n$$

betrachten.

Die gängigste Methode zur Schätzung der unbekannten Modellparameter $\beta_0, \beta_1, \dots, \beta_p$ ist die **Methode der kleinsten Quadrate**. Wir definieren die **Residuenquadratsumme** RSS (Residual Sum of Squares) als:

$$\text{RSS} = e_1^2 + e_2^2 + \dots + e_n^2$$

oder äquivalent als

$$\text{RSS} = (y_1 - \hat{\beta}_0 + \hat{\beta}_1 x_{11} + \dots + \hat{\beta}_p x_{1p})^2 + \dots + (y_n - \hat{\beta}_0 + \hat{\beta}_1 x_{n1} + \dots + \hat{\beta}_p x_{np})^2$$

Die Methode der kleinsten Quadrate bestimmt die Parameterschätzungen $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p)^T$ durch **Minimierung der Residuenquadratsumme RSS**. Nach ein paar Rechnungen kann man zeigen, dass

$$\begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_p \end{pmatrix} = \left(\begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ \vdots & & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix}^T \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ \vdots & & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix} \right)^{-1} \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ \vdots & & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix}^T \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}$$

2.2.2 Polynomregression

Die **Polynomregression** ist eine Möglichkeit, die nicht linearen Beziehungen zwischen der Zielvariablen und den Prädiktorvariablen in unserem Benzinverbrauchsproblem (siehe Abbildung 2.1) berücksichtigen zu können. So kann, zum Beispiel, der nicht lineare Zusammenhang zwischen **Verbrauch** und **Leistung PS** sehr flexibel als Polynomfunktion modelliert werden:

$$\text{Verbrauch} = \beta_0 + \beta_1 \text{PS} + \beta_2 \text{PS}^2 + \dots + \beta_p \text{PS}^p$$

Je höher der Grad p des Polynoms, desto flexibler ist ein Polynomregressionsmodell und ermöglicht so auch die Modellierung nicht linearen Zusammenhänge. Das Polynomregressionsmodell ist jedoch für alle Polynomgrade p ein **(multivariate) lineares Regressionsmodell**, denn es ist linear bezüglich der Modellparameter $\beta_0, \beta_1, \dots, \beta_p$.

```
## Polynom Regressionen
polreg_1 <- lm(Verbrauch ~ poly(PS, degree = 1, raw=TRUE), data = Auto_df)
polreg_2 <- lm(Verbrauch ~ poly(PS, degree = 2, raw=TRUE), data = Auto_df)
polreg_5 <- lm(Verbrauch ~ poly(PS, degree = 5, raw=TRUE), data = Auto_df)
## Data-Frame zum Abspeichern der Prädiktionen
plot_df <- tibble("PS" = seq(45, 250, len=50))
## Abspeichern der Prädiktionen
plot_df$fit_1 <- predict(polreg_1, newdata = plot_df)
plot_df$fit_2 <- predict(polreg_2, newdata = plot_df)
plot_df$fit_5 <- predict(polreg_5, newdata = plot_df)
## Ploten
plot(Verbrauch ~ PS, data = Auto_df, ylim=c(2,20),
     xlab="Leistung (PS)", pch=21, col="gray", bg="gray", cex=1.5)
with(plot_df, lines(x = PS, y = fit_1, lwd=2, col="orange"))
with(plot_df, lines(x = PS, y = fit_2, lwd=2, col="blue"))
with(plot_df, lines(x = PS, y = fit_5, lwd=2, col="darkgreen"))
legend("topright", lty=c(NA,1,1,1), pch=c(21,NA,NA,NA),
     col=c("gray","orange","blue","darkgreen"), pt.bg="gray", pt.cex=1.5,
     legend=c("Datenpunkte", "Grad 1", "Grad 2", "Grad 5"), bty="n")
```

Überanpassung

Zusätzlich zur Wahl der Modellparameter $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$ besteht hier nun das Problem der Wahl des Grades p des Polynoms als weiteren Modellparameter

$$y_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2}^2 + \dots + \hat{\beta}_p x_{ip}^p + e_i$$

Wenn man jedoch versucht, alle Modellparameter (also $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$ und p) durch Minimieren der Trainingsdaten-RSS

$$\text{RSS} \equiv \text{RSS}(\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p, p) = e_1^2 + e_2^2 + \dots + e_n^2$$

zu schätzen, so ergibt sich ein Problem das als **Überanpassung (Overfitting)** bekannt ist (siehe Abbildung 2.5). Das Polynomregressionsmodell ist so flexibel, dass es den einzelnen Trainingsdaten (x_i, y_i) folgen kann. Eine Überangepassung an die Trainingsdaten führt jedoch notwendigerweise zu einer Verschlechterung der Vorhersagegüte bezüglich *neuer* Daten.

2.3 Modellauswahl

Maschinelles Lernen versus Strukturelle Modelle

Das oben veranschaulichte Problem der Überanpassung (Overfitting) ist eng damit verbunden, dass wir hier ein sehr flexibles Regressionsmodell (Polynomregression) betrachten. Viele der möglichen Polynomfunktionen sind unsinnig, da sie nicht die strukturellen Einschränkungen des betrachteten Datenproblems berücksichtigen. Falls ein gesichertes Wissen zu den zugrundeliegenden, strukturellen Zusammenhängen zwischen der Zielvariable Y und den Prädiktorvariablen X existiert, sollte man diese strukturellen Zusammenhängen auch im statistischen Modell berücksichtigen. (Immer mit den Expert*Innen des Faches sprechen!) Im besten Falle gibt es ein **strukturelles Modell** zu den systematischen Zusammenhängen f zwischen Y und X , welches genügend Einschränkungen bietet, sodass alle unsinnigen Modellierungen vermieden werden können. In solchen Idealfällen führt die Minimierung der Trainingsdaten-RSS zu keinem Problem der Überanpassung.

Falls jedoch kein (vertrauenswürdige) strukturelles Modell vorliegt, ist die Verwendung von sehr flexiblen Regressionsmodellen wie der Polynomregression eine grundsätzlich sehr gute Idee, da wir so, ohne große Einschränkungen, nach den unbekannten richtigen Zusammenhängen f suchen können. Dies ist der Ansatz des **maschinellen Lernens** und die **Polynomregression mit unbekanntem Polynomgrad p** ist lediglich eine von sehr vielen Methoden, welche im Kontext des maschinellen Lernens verwendet werden.

Fazit: Methoden des **maschinellen Lernens** sind typischerweise sehr flexibel und bauen nicht auf strukturellen Modellen auf. Daher benötigen diese Methoden spezielle Verfahren der **Modellauswahl**, um eine Überanpassung an die Trainingsdaten zu vermeiden. Richtig angewandt, können Methoden des maschinellen Lernens unbekannte Zusammenhänge richtig erkennen.

2.3.1 Die Validierungsdaten-Methode

Da die Minimierung der Trainingsdaten-RSS schnell zu einem Problem der Überanpassung führt, benötigen wir eine alternative Methode, um die Güte des

geschätzten Modells zu prüfen. Die einfachste Idee ist dabei die beobachteten Daten in einen Satz von Trainingsdaten

$$\text{Trainingsdaten} = \{(x_1^{Train}, y_1^{Train}), (x_2^{Train}, y_2^{Train}), \dots, (x_{n_{Train}}^{Train}, y_{n_{Train}}^{Train})\}$$

und einen **separaten** (disjunkten) Satz von Validierungsdaten

$$\text{Validierungsdaten} = \{(x_1^{Valid}, y_1^{Valid}), (x_2^{Valid}, y_2^{Valid}), \dots, (x_{n_{Valid}}^{Valid}, y_{n_{Valid}}^{Valid})\}$$

zu teilen mit

- $n = n_{Train} + n_{Valid}$
- $\text{Trainingsdaten} \cap \text{Validierungsdaten} = \emptyset$

Folgender Code-Schnipsel ermöglicht solch eine (zufällige) Aufteilung der Daten in Trainings- und Validierungsdaten:

```
n          <- nrow(Auto_df) # Stichprobenumfang
n_Train    <- 200           # Stichprobenumfang der Trainingsdaten
n_Valid    <- n - n_Train   # Stichprobenumfang der Validierungsdaten

## Index-Mengen zur Auswahl der
## Trainings- und Validierungsdaten
I_Train    <- sample(x = 1:n, size = n_Train, replace = FALSE)
I_Valid    <- c(1:n)[-I_Train]

## Trainingsdaten
Auto_Train_df <- Auto_df[I_Train, ]
## Validierungsdaten
Auto_Valid_df <- Auto_df[I_Valid, ]
```

Obschon die Validierungsdaten-Methode auf alle Regressionsmodelle angewandt werden kann, veranschaulichen wir im Folgenden die Methode anhand der Polynomregression.

Die Aufteilung der Daten in Trainings- und Validierungsdaten ermöglicht uns nun ein zweistufiges Verfahren:

Schritt 1: Mit Hilfe der **Trainingsdaten** wird das Polynomregressionsmodell **geschätzt**:

$$y_i^{Train} = \hat{\beta}_0^{Train} + \hat{\beta}_1^{Train} x_i^{Train} + \hat{\beta}_2^{Train} (x_i^{Train})^2 + \dots + \hat{\beta}_p^{Train} (x_i^{Train})^p + e_i^{Train}$$

Code-Schnipsel Beispiel:

```
Train_polreg <- lm(Verbrauch ~ poly(PS, degree = p, raw=TRUE), data = Auto_Train_df)
```

Schritt 2: Mit Hilfe der **Validierungsdaten** wird das geschätzte Polynomregressionsmodell **validiert**:

$$\hat{y}_i^{Valid} = \hat{\beta}_0 + \hat{\beta}_1^{Train} x_i^{Valid} + \hat{\beta}_2^{Train} (x_i^{Valid})^2 + \dots + \hat{\beta}_p^{Train} (x_i^{Valid})^p,$$

indem man den **mittleren quadratischen Prädiktionsfehler** (Mean Squared Prediction Error **MSPE**) berechnet:

$$\begin{aligned} \text{MSPE} &= \frac{1}{n_{Valid}} \text{RSS}_{Valid} \\ &= \frac{1}{n_{Valid}} \left((y_1^{Valid} - \hat{y}_1^{Valid})^2 + \dots + (y_{n_{Valid}}^{Valid} - \hat{y}_{n_{Valid}}^{Valid})^2 \right) \end{aligned}$$

Code-Schnipsel Beispiel:

```
y_fit_Valid <- predict(Train_polreg, newdata = Auto_Valid_df)
RSS_Valid <- sum( (Auto_Valid_df$Verbrauch - y_fit_Valid)^2 )
MSPE <- RSS_Valid / n_Valid
```

Man wiederholt obige Schritte für eine Auswahl von verschiedenen Polynomgraden $p = 1, 2, \dots, p_{\max}$, z.B. $p_{\max} = 10$, und berechnet für jeden dieser Fälle den MSPE, also:

$$\text{MSPE} \equiv \text{MSPE}(\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p, p), \quad \text{für jedes } p = 1, 2, \dots, p_{\max}$$

Der MSPE ist eine Schätzung des wahren, unbekannten mittleren quadratischen Prädiktionsfehlers $E[(Y - \hat{Y})^2]$,

$$\text{MSPE}(\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p, p) \approx E[(Y - \hat{Y})^2].$$

Die Minimierung des MSPE über verschiedene Werte des Polynomgrades $p = 1, 2, \dots$ erlaubt es uns den **reduzierbaren Prädiktions-Fehler** der Polynomregression zu minimieren.

Folgender R-Code verbindet nun alle Schritte und berechnet den MSPE für verschiedene Werte des Polynomgrades p . Dasjenige Modell, welches den MSPE minimiert, ist laut der Daten das beste Prädiktionsmodell.

```
set.seed(31)
##
n <- nrow(Auto_df) # Stichprobenumfang
n_Train <- 200 # Stichprobenumfang der Trainingsdaten
n_Valid <- n - n_Train # Stichprobenumfang der Validierungsdaten
```

```

## Index-Mengen zur Auswahl der
## Trainings- und Validierungsdaten
I_Train <- sample(x = 1:n, size = n_Train, replace = FALSE)
I_Valid <- c(1:n)[-I_Train]

## Trainingsdaten
Auto_Train_df <- Auto_df[I_Train, ]
## Validierungsdaten
Auto_Valid_df <- Auto_df[I_Valid, ]

p_max      <- 6
MSPE       <- numeric(p_max)
fit_plot   <- matrix(NA, 50, p_max)
for(p in 1:p_max){
  ## Schritt 1
  Train_polreg <- lm(Verbrauch ~ poly(PS, degree = p, raw=TRUE),
                    data = Auto_Train_df)

  ## Schritt 2
  y_fit_Valid <- predict(Train_polreg, newdata = Auto_Valid_df)
  RSS_Valid   <- sum( (Auto_Valid_df$Verbrauch - y_fit_Valid)^2 )
  MSPE[p]     <- RSS_Valid / n_Valid

  ## Daten für's plotten
  fit_plot[,p] <- predict(Train_polreg, newdata = plot_df)
}

```

Achtung: Auch eine Modellauswahl ist fehlerhaft und stellt lediglich eine Schätzung (mit Schätzfehlern) des besten Prädiktionsmodelles innerhalb der betrachteten Klasse von Prädiktionsmodellen (hier Polynomregressionen) dar.

Abbildung 2.7 zeigt jedoch ein Problem der Validierungsdaten-Methode. Die Trainingsdaten und die Validierungsdaten haben kleinere Stichprobenumfänge ($n_{Train} < n$ und $n_{Valid} < n$) was zu einer **erhöhten Schätzgenauigkeit in der MSPE-Schätzung** führt.

2.3.2 k-Fache Kreuzvalidierung

Die k -fache (z.B. $k = 5$ oder $k = 10$) Kreuzvalidierung ist eine Vorgehensweise zur Bewertung der Leistung einer Schätzprozedur (Algorithmus) im Kontext des maschinellen Lernens. Als Schätzprozedur verwenden wir wieder das Beispiel der Polynomregression mit unbekanntem Polynomgrad p , welcher zusammen mit den Modellparametern $\beta_0, \beta_1, \dots, \beta_p$ aus den Daten erlernt werden muss.

Die k -fache Kreuzvalidierung stellt eine Verbesserung der Validierungsdaten-Methode dar, da sie faktisch die Stichprobenumfänge in den Trainingsdaten und Validierungsdaten erhöht. Wie bei der Validierungsdaten-Methode wird der Datensatz in Trainings- und Validierungsdaten aufgeteilt – jedoch k -fach. Abbildung ?? zeigt ein Beispiel der Datenaufteilung bei der 5-fachen Kreuzvalidierung.

Folgender Code-Schnipsel ermöglicht eine (zufällige) Aufteilung der Daten in k verschiedene Trainings- und Validierungsdaten:

```
n      <- nrow(Auto_df) # Stichprobenumfang
k      <- 5             # 5-fache Kreuzvalidierung

## Index zur Auswahl k verschiedener
## Trainings- und Validierungsdaten:
folds  <- sample(x = 1:k, size = n, replace=TRUE)

## Trainingsdaten im j-ten (j=1,2,...,k) Durchgang
Auto_df[folds != j,]
## Validierungsdaten im j-ten (j=1,2,...,k) Durchgang
Auto_df[folds == j,]
```

Für jede der k Datenaufteilungen wird der MSPE berechnet. Der Mittelwert dieser MSPE-Werte wird häufig als $CV_{(k)}$ Wert (crossvalidation score) bezeichnet

$$CV_{(k)} = \frac{1}{k} \sum_{j=1}^k MSPE_j$$

Der $CV_{(k)}$ -Wert stellt eine im Vergleich zur Validierungsdaten-Methode verbesserte Schätzung des unbekannten mittleren quadratischen Prädiktionsfehlers $CV_{(k)} \approx E[(Y - \hat{Y})^2]$ dar. Die Modellauswahl folgt also auch hier mittels Minimierung des $CV_{(k)}$ -Wertes über die verschiedene Werte des Polynomgrades $p = 1, 2, \dots$

Wahl von k : In der Praxis haben sich die Werte $k = 5$ und $k = 10$ etabliert, da diese Größenordnungen einen guten Kompromiss zwischen der Varianz und der Verzerrung des Schätzers $CV_{(k)}$ für $E[(Y - \hat{Y})^2]$ darstellen.

2.4 Anwendung: Vorhersage des Benzinverbrauchs

Nun haben wir das Werkzeug, um die nicht linearen Zusammenhänge zwischen der **Zielvariable** Y =Verbrauch und den **Prädiktorvariablen** G =Gewicht,

P =PS und H =Hubraum im Datensatz `Auto_df` zu berücksichtigen (siehe Abbildung 2.1) und allein mit Hilfe der Daten zu erlernen. Wir folgen hier der Herangehensweise des **maschinellen Lernens** und lassen die **Daten für sich selbst sprechen**.

Da Abbildung 2.1 sehr ähnliche Zusammenhänge zwischen der Zielvariable Y =Verbrauch und den Prädiktorvariablen G =Gewicht, P =PS und H =Hubraum vermuten lässt, betrachten wir zunächst ein vereinfachtest Polynomregressionsmodell, bei dem für alle Prädiktorvariablen der gleiche Polynomgrad p verwendet wird.

$$Y_i = \beta_0 + \beta_1^G G_i + \beta_2^G G_i^2 + \dots + \beta_p^G G_i^p + \beta_1^P P_i + \beta_2^P P_i^2 + \dots + \beta_p^P P_i^p + \beta_1^H H_i + \beta_2^H H_i^2 + \dots + \beta_p^H H_i^p + \varepsilon_i$$

Folgender R-Code (Algorithmus) erlernt aus den Daten, mit Hilfe der 5-fachen Kreuzvalidierung $CV_{(5)} \approx E[(Y - \hat{Y})^2]$, den optimalen Polynomgrad p .

```
set.seed(8)                # Seed für den Zufallsgenerator

n      <- nrow(Auto_df) # Stichprobenumfang
k      <- 5             # 5-fache Kreuzvalidierung
p_max  <- 5             # Maximaler Polynomgrad

folds   <- sample(x = 1:k, size = n, replace=TRUE)

## Container für die MSPE-Werte
## für alle j=1,...,k Kreuzvalidierungen und
## für alle p=1,...,p_max Polynomgrade
MSPE <- matrix(NA, nrow = k, ncol = p_max,
               dimnames=list(NULL, paste0("p=", 1:p_max)))

for(p in 1:p_max){
  for(j in 1:k){
    ## Modellschätzung auf Basis j-ten Trainingsdaten Auto_df[folds != j,]
    poly_fit <- lm(Verbrauch ~
                  poly(Gewicht, degree = p, raw = TRUE) +
                  poly(PS, degree = p, raw = TRUE) +
                  poly(Hubraum, degree = p, raw = TRUE),
                  data=Auto_df[folds != j,])
    ## Prädiktion auf Basis j-ten Validierungsdaten Auto_df[folds == j,]
    pred <- predict(poly_fit, newdata = Auto_df[folds == j,])
    ##
    MSPE[j,p] <- mean( (Auto_df$Verbrauch[folds==j] - pred)^2 )
  }
}
```

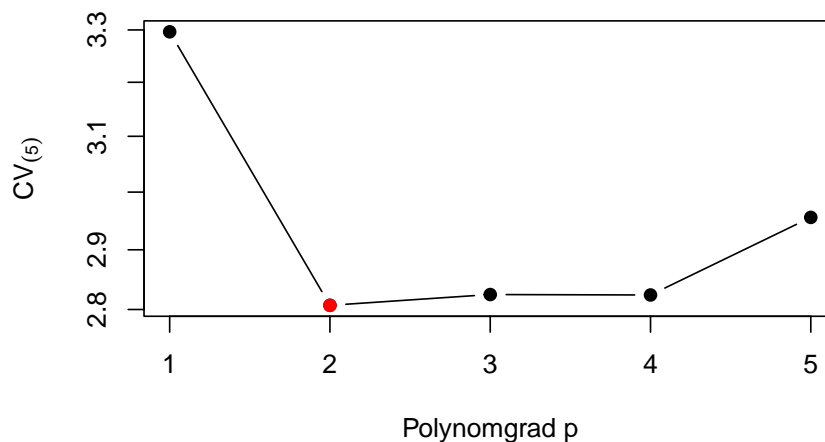
```

    }
  }

  ## CV-Wert für alle p=1,...,p_max Polynomgrade
  CV_k <- colMeans(MSPE)

  ## Plotten
  plot(y = CV_k, x = 1:length(CV_k), pch=21, col="black", bg="black",
       type='b', xlab="Polynomgrad p", ylab=expression(CV[(5)]), log="y")
  points(y = CV_k[which.min(CV_k)],
        x = c(1:length(CV_k))[which.min(CV_k)],
        col = "red", bg = "red", pch = 21)

```



Auch der 5-fache Kreuzvalidierungswert $CV_{(5)}$ ist lediglich eine zufallsbehaftete Schätzung des unbekannten mittleren quadratischen Prädiktionsfehlers $E[(Y - \hat{Y})^2]$. Um eine Idee von der Präzision und Stabilität der Modellauswahl mittels der Minimierung von $CV_{(5)}$ zu bekommen, können wir die zufälligen, 5-fachen Aufteilungen der Daten in Trainins- und Validierungsdaten wiederholen und den Effekt alternativer Datenaufteilungen betrachten. Abbildung 2.8 zeigt, dass die Minimierung des Kreuzvalidierungswertes $CV_{(5)}$ auch in Wiederholungen häufig das Modell mit Polynomgrad $p = 2$ auswählt. Der Polynomgrad $p = 2$ scheint also eine vertrauenswürdige Modellauswahl darzustellen.

Das Polynomregressionsmodell mit $p = 2$ stellt also ein gutes Prädiktionsmodell dar. Wir verwenden nun dieses Modell, um nach auffälligen Unterschieden in den herstellerseitigen Verbrauchsangaben y_i und unseren Prädiktionen zu suchen. Gerade **stark negative Residuen** $y_i - \hat{y}_i$ sind verdächtig, da es auf eine Schönung der Verbrauchsangaben hindeuten könnte.

Folgender R-Code schätzt zunächst das Polynomregressionsmodell mit $p = 2$, berechnet dann die Residuen $y_i - \hat{y}_i$ und veranschaulicht die größte negative Abweichung in Abbildung 2.9.

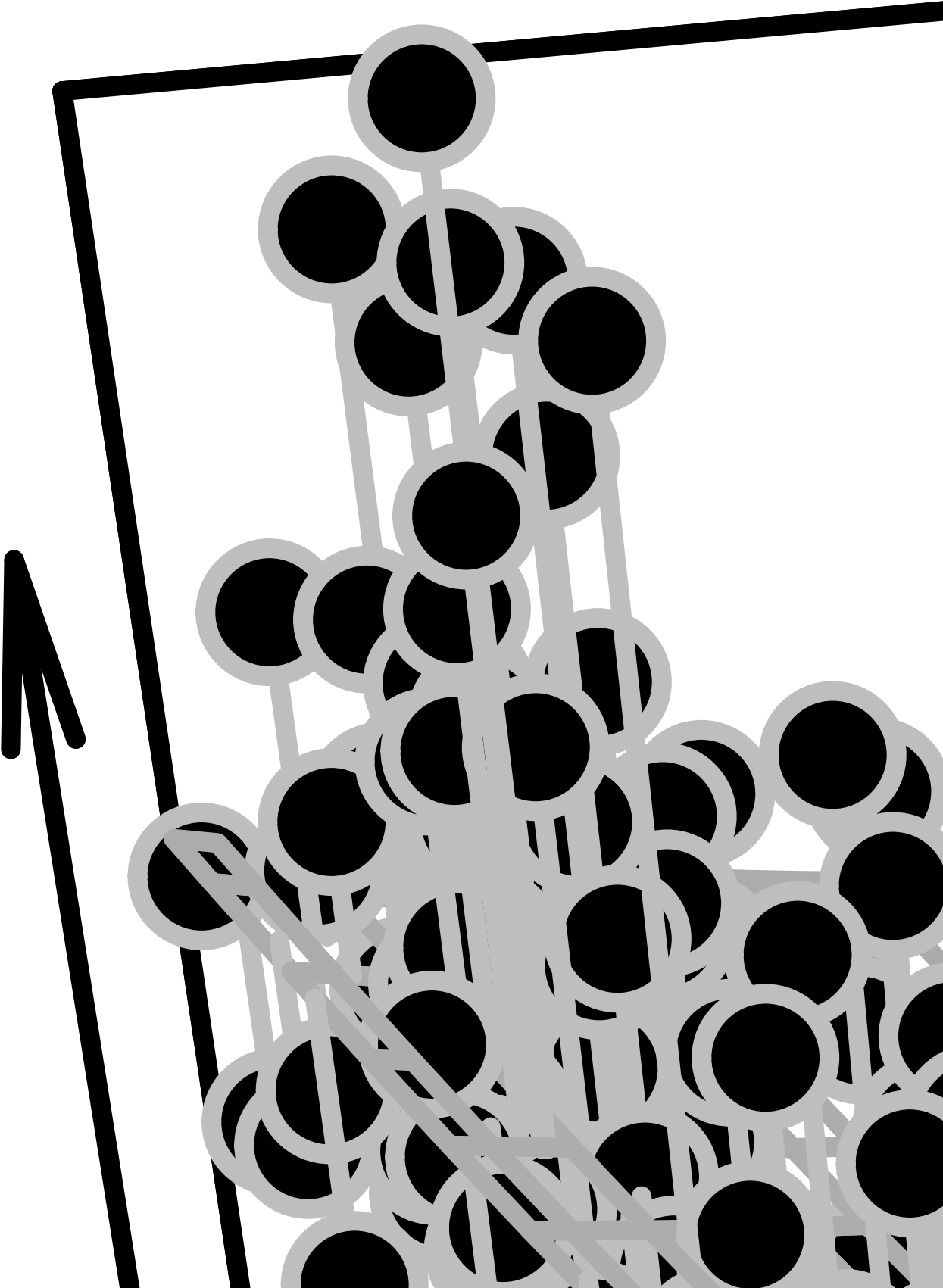
```
p <- 2
poly_fit <- lm(Verbrauch ~
               poly(Gewicht,      degree = p, raw = TRUE) +
               poly(PS,          degree = p, raw = TRUE) +
               poly(Hubraum,     degree = p, raw = TRUE),
               data=Auto_df)

par(mar=c(5.1, 5.1, 4.1, 2.1))
plot(y = resid(poly_fit), x = fitted(poly_fit),
     ylab = expression("Residuen:"~y[i] - hat(y)[i]),
     xlab = expression("Prädiktionen:"~hat(y)[i]),
     main="Größte negative Abweichung der Verbrauchsangabe")
slct <- order(abs(resid(poly_fit)), decreasing = TRUE)[4]
points(y = resid(poly_fit)[slct], x = fitted(poly_fit)[slct],
       col = "red", bg = "red", pch = 21)
text(y = resid(poly_fit)[slct], x = fitted(poly_fit)[slct],
     labels = "Mazda RX-3 (1973)", pos = 2)
```

```
par(mar=c(5.1, 4.1, 4.1, 2.1))
```

Wir haben hier tatsächlich einen besonderen Fall gefunden. Der Mazda RX-3 (1973) (Abbildung ??) lief mit einem ungewöhnlich sparsamen Wankelmotor. Dieser Motor war so ungewöhnlich, dass es vielerlei Streitigkeiten um die Verbrauchsangaben gab.

2.5 Ende



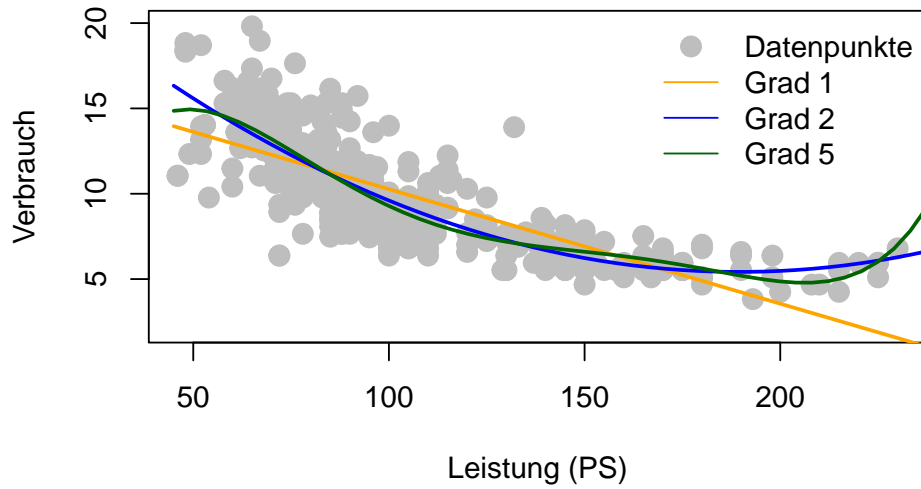


Abbildung 2.4: Polynom Regression bei verschiedenen Polynomgraden p .

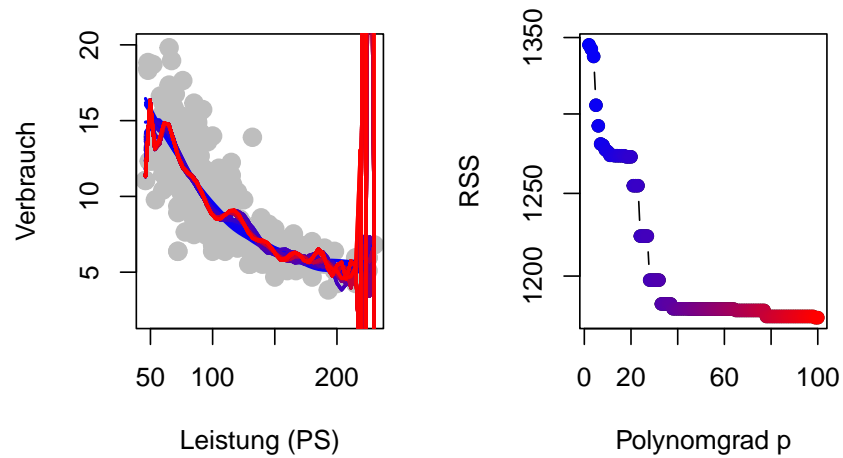


Abbildung 2.5: Polynom Regression und die Wahl des Polynomgrades p durch Minimierung der Trainingsdaten-RSS. (Eine schlechte Idee).

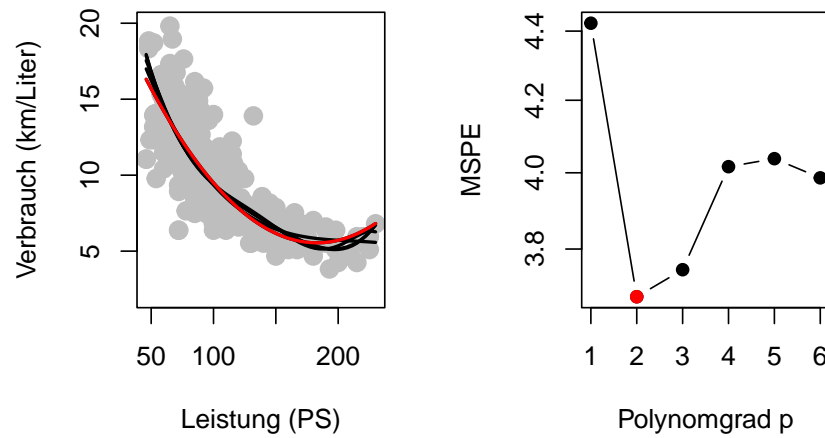


Abbildung 2.6: Polynom Regression und die Wahl des Polynomgrades p durch Minimierung des mittleren quadratischen Prädiktionsfehler MSPE.

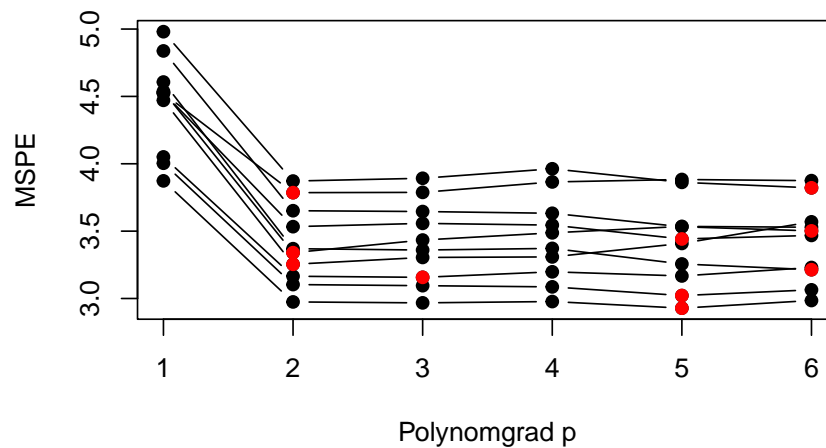


Abbildung 2.7: Zehn verschiedene MSPE-Berechnungen basierend auf zehn verschiedenen, zufälligen Aufteilungen der Daten in Trainings- und Validierungsdaten.

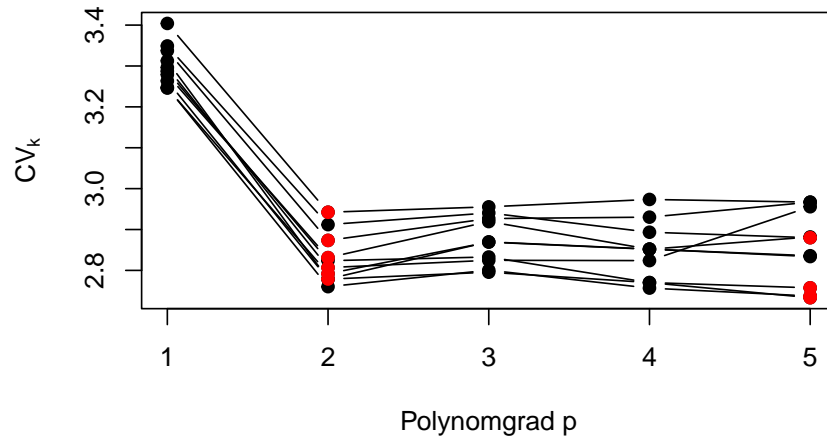


Abbildung 2.8: Zehn verschiedene $CV_{(k)}$ -Berechnungen basierend auf zehn verschiedenen, zufälligen Wiederholungen der 5-fachen Kreuzvalidierung.

Größte negative Abweichung der Verbrauchsangabe

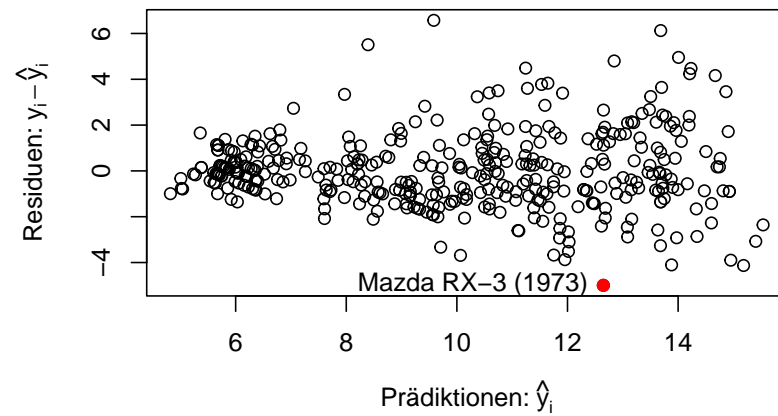


Abbildung 2.9: Polynomregression im Anwendungsbeispiel zum Benzinverbrauch. Die größte negative Abweichung ist der Mazda RX-3 von 1973.

Literaturverzeichnis

- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer Science & Business Media.
- Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1):1–22.
- Hanck, C., Arnold, M., Gerber, A., and Schmelzer, M. (2021). *Introduction to Econometrics with R*. www.econometrics-with-r.org.
- Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer Science.
- Horst, A. M., Hill, A. P., and Gorman, K. B. (2020). *palmerpenguins: Palmer Archipelago (Antarctica) Penguin Data*. R Package Version 0.1.0.
- James, G., Witten, D., Hastie, T., and Tibshirani, R. (2021). *An Introduction to Statistical Learning: With Applications in R*. Springer Science.
- Shmueli, G. (2010). To Explain or to Predict? *Statistical Science*, 25(3):289 – 310.