ASMNW - Lösung 3

Peter von Rohr

2018-03-15

Kontrollfrage 1

1. Aus welchen Komponenten besteht ein lineares Modell?
2. Welcher Zusammenhang besteht zwischen diesen Komponenten

Lösung

1. Zielgrösse, erklärende Variablen und Resteffekte
2. Zielgrösse soll als lineare Funktion der erklärenden Variablen dargestellt werden

Kontrollfrage 2

1. Welche fixen und welche zufälligen Effekte gibt es in einer multiplen linearen Regression
2. Wie lauten Erwartungswerte und Varianzen der zufälligen Effekte

Lösung

1. erklärende Variablen sind fix und Resteffekte sind zufällig
2. Erwartungswert $E(\epsilon) = \mathbf{0}$ und die Co-Varianz-Matrix beträgt: $Var(\epsilon) = \mathbf{I} * \sigma^2$

Kontrollfrage 3

Welche vier Ziele wollen wir mit einer multiplen linearen Regression erreichen?

Lösung

1. gute **Anpassung** des Modells, so dass Abweichungen zu beobachteten Zielgrössen möglichst klein
2. gute **Schätzung** der Parameter, Änderungen bei den erklärenden Variablen sollen möglichst eng mit Änderungen in Zielgrössen zusammenhängen
3. gute **Voraussage** für neue Werte der erklärenden Variablen innerhalb des Wertebereichs des Datensatzens, keine Extrapolation
4. Unsicherheiten sollen durch Tests und Vertrauensintervalle quantifiziert werden

Kontrollfrage 4

1. Was bedeutet der Ausdruck
$$\hat{\beta} = argmin_{\beta}||\mathbf{y} - \mathbf{X}\beta||^2$$
2. Welcher Schätzer resultiert aus dem Ausdruck unter 1?

Lösungen

1. Der Ausdruck $||\mathbf{y} - \mathbf{X}\beta||^2$ wird nach β abgeleitet und 0 gesetzt und der Wert für β an diesem Minimum wird als Schätzer $\hat{\beta}$ verwendet.
2. $$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Aufgabe 1: Lineare Modelle

In der Vorlesung wurden die Eigenschaften von linearen Modellen besprochen. Die folgende Tabelle enthält vier verschiedene Modelle. Tragen Sie in die Kolonne ganz rechts ein, ob es sich jeweilen um ein lineares oder nicht-lineares Modell handelt.

Nr	Model	Typ(linear/nicht-linear)
1	$y_i = \beta_1^2 + \sqrt{\beta_2}x_{i,2} + \epsilon_i$	
2	$y_i = \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3} + \epsilon_i$	
3	$y_i = \beta_1 x_{i,1} + \beta_2 \log x_{i,2} + \beta_3 sin(x_{i,3}) + \epsilon_i$	
4	$y_i = \beta_1^2 \log x_{i,1} + \sqrt{\beta_2}x_{i,2} + \epsilon_i$	

Lösung

Nr	Model	Typ(linear/nicht-linear)
1	$y_i = \beta_1^2 + \sqrt{\beta_2}x_{i,2} + \epsilon_i$	nicht linear
2	$y_i = \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3} + \epsilon_i$	linear
3	$y_i = \beta_1 x_{i,1} + \beta_2 \log x_{i,2} + \beta_3 sin(x_{i,3}) + \epsilon_i$	linear
4	$y_i = \beta_1^2 \log x_{i,1} + \sqrt{\beta_2}x_{i,2} + \epsilon_i$	nicht linear

Aufgabe 2: Regressionsanalyse

Während der Vorlesung habe wir das Beispiel mit der Zunahme nach dem Absetzen als Zielgrösse betrachtet. Wir haben aber nirgends die Koeffizienten und die geschätzte Restvarianz berechnet. Stellen Sie für dieses kleine Beispiel das lineare Modell gemäss Vorlesungsunterlagen auf und berechnen Sie die Koeffizienten des linearen Modells und die geschätzte Restvarianz.

Lösung

Zuerst werden die Daten in ein `data.frame` abgelegt.

```
dfBwWgData <- data.frame(BW = c(35.0, 25.3, 34.2, 31.2, 38.7),
                         WWG = c(0.90, 0.58, 0.78, 0.70, 1.00),
                         PWG = c(1.36, 1.00, 1.36, 1.20, 1.50))
```

Mit diesem `data.frame` können wir schon ein lineares Modell anpassen.

```
lmBwWg <- lm(PWG ~ ., data = dfBwWgData)
summary(lmBwWg)
```

```
##
## Call:
## lm(formula = PWG ~ ., data = dfBwWgData)
##
## Residuals:
##          1          2          3          4          5
## -9.919e-05  6.950e-03  2.004e-02 -2.439e-02 -2.497e-03
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.008119   0.101168   0.080   0.9433
## BW           0.041413   0.008749   4.733   0.0419 *
## WWG         -0.108294   0.266154  -0.407   0.7235
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02292 on 2 degrees of freedom
## Multiple R-squared:  0.9928, Adjusted R-squared:  0.9856
## F-statistic: 137.9 on 2 and 2 DF,  p-value: 0.007201
```

Die Modellanpassung ohne Achsenabschnitt

```
lmBwWgNI <- lm(PWG ~ -1 + BW + WWG, data = dfBwWgData)
summary(lmBwWgNI)
```

```
##
## Call:
## lm(formula = PWG ~ -1 + BW + WWG, data = dfBwWgData)
##
## Residuals:
##          1          2          3          4          5
##  0.0002699  0.0084706  0.0191781 -0.0246233 -0.0028784
##
## Coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## BW   0.041997   0.003965  10.592   0.0018 **
## WWG -0.122413   0.163336  -0.749   0.5080
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01875 on 3 degrees of freedom
## Multiple R-squared:  0.9999, Adjusted R-squared:  0.9998
## F-statistic: 1.194e+04 on 2 and 3 DF,  p-value: 1.409e-06
```

Anstatt das Modell ohne Achsenabschnitt ganz neu anzugeben, gibt es auch die Möglichkeit ein bestehendes Modell mit der Funktion `update()` anzupassen. Dadurch können wir mit folgendem Befehl das ursprüngliche Modell in `lmBwWg` anpassen. Dabei ist das erste Argument von `update()` das ursprüngliche Objekt, welches das lineare Modell enthält (bei uns ist das `lmBwWg`). Das zweite Argument ist die neue Modellformel. Im nachfolgenden Aufruf von `update()` wird eine abgekürzte Schreibweise verwendet, wobei die Punkte als Platzhalter für die Komponenten im schon bestehenden Modell in `lmBwWg` stehen. Somit heisst die Abkürzung `. ~ . -1`: nimm das bestehenede Modell und entferne den Achsenabschnitt.

```r
lmBwGwUpd <- update(lmBwWg, . ~ . -1)
summary(lmBwGwUpd)
```

```
##
## Call:
## lm(formula = PWG ~ BW + WWG - 1, data = dfBwWgData)
##
## Residuals:
##          1          2          3          4          5
##  0.0002699  0.0084706  0.0191781 -0.0246233 -0.0028784
##
## Coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## BW   0.041997   0.003965  10.592   0.0018 **
## WWG -0.122413   0.163336  -0.749   0.5080
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01875 on 3 degrees of freedom
## Multiple R-squared:  0.9999, Adjusted R-squared:  0.9998
## F-statistic: 1.194e+04 on 2 and 3 DF,  p-value: 1.409e-06
```

Die Resultate der Modellanpassung mit und ohne Achsenabschnitte mögen auf den ersten Blick verwirrend aussehen. Aufgrund des höheren R^2-Wertes und basierend auf der tieferen Restvarianz des Modells ohne Achsenabschnitt, würde man dieses Modell ohne Achsenabschnitt als das bessere Modell bezeichnen. Aufgrund der biologischen Zusammenhänge macht aber ein Modell ohne Achsenabschnitt wenig Sinn. Als Erklärung dafür gilt wohl, dass nur fünf Beobachtungen wohl zu wenig sind um zwischen zwei Modellen sicher unterschieden zu können. Des Weiteren sind die Unterschiede zwischen den Modellen sehr gering und würden eine Rangierung der Modelle kaum erlauben.

Aufgabe 3: GBLUP

Gegeben sind die Genotypen von 10 Tieren an 1000 SNP-Genorten. Die Genotypen können von der Datei asmas_w04_u03_genotypes.txt gelesen werden. Da wir nach dem Einlesen die Genotypen in Form einer Matrix haben wollen, werden die Inputdaten am einfachsten mit den Funktionen `matrix(scan())` eingelesen und einer Variablen zugewiesen, wie dies im folgenden Statement gezeigt wird (Die beiden '#' gehören nicht zum Einlese-Statement und deshalb müssen Sie diese weglassen).

```r
## mat_data <- matrix(scan("asmas_w04_u03_genotypes.txt"), ncol=1000,  byrow = TRUE)
```

Ihre Aufgabe

Aus den eingelesenen SNP-Genotypen soll die genomische Verwandtschaftsmatrix G berechnet werden. Dazu müssen Sie zuerst herausfinden, in welcher Codierung die SNP-Genotypen vorliegen. In der Vorlesung haben wir gesehen, dass wir zwischen den Codierungen (0, 1, 2) und (-1, 0, 1) unterscheiden können. In den Vorlesungsunterlagen wurde die Herleitung von G für die zweite Codierung gezeigt. Sie können die Codierung der eingelesenen Genotypen mit der Funktion `table()` bestimmen. Dabei geben Sie die Matrix der eingelesenen Genotypen als Argument für die Funktion an.

Damit sollten Sie alle nötigen Angaben haben, damit Sie die Matrix G berechnen können.

Zusatzaufgabe

Da wir die Matrix G anschliessend in den Mischmodellgleichungen von GBLUP verwenden wollen, muss diese Matrix positiv-definit sein, da wir ihre Inverse G^{-1} benötigen. Für die Überprüfung ob die Matrix G positiv-definit ist, können Sie ihre Eigenwerte mit der Funktion `eigen()` berechnen. Die Funktion `eigen()` gibt als Resultat eine Liste zurück, wobei die Eigenwerte als Vektor unter dem Namen `values` abgelegt sind.

Lösung

- Zuerst lesen wir die Genotypen mit dem angegebenen Statement in eine Matrix ein.

```
# store name of genotype inputfile in a variable
s_genotypes_filename <- "asmas_w04_u03_genotypes.txt"
# fix the number of snp locations
n_nr_snp <- 1000
# reading the data
mat_data <- matrix(scan(s_genotypes_filename), ncol = n_nr_snp, byrow = TRUE)
```

Als Kontrolle überprüfen wir die Dimension der Matrix mit den eingelesenen Genotypen

```
dim(mat_data)
```

```
## [1]   10 1000
```

- Wie in der Aufgabe erwähnt, überprüfen wir die Codierung mit der Funktion `table()`.

```
table(mat_data)
```

```
## mat_data
##    0    1    2
## 9780  219    1
```

- Zur Berechnung der genomischen Verwandtschaftsmatrix verwenden wir die im Skript beschriebene Funktion, welche hier nochmals aufgeführt wird.

```
#' Compute genomic relationship matrix based on data matrix
computeMatGrm <- function(pmatData) {
  # Allele frequencies, column vector of P and sum of frequency products
  freq <- apply(pmatData, 2, mean) / 2
  P <- 2 * (freq - 0.5)
  sumpq <- sum(freq*(1-freq))
  # Changing the coding from (0,1,2) to (-1,0,1) and subtract matrix P
  Z <- pmatData - 1 - matrix(P, nrow = nrow(pmatData),
                             ncol = ncol(pmatData),
                             byrow = TRUE)
  # Z%*%Zt is replaced by tcrossprod(Z)
  return(tcrossprod(Z)/(2*sumpq))
}
```

Die Matrix G wird dann mit dem folgenden Aufruf berechnet

```
(mat_grm <- computeMatGrm(pmatData = mat_data))
```

```
##              [,1]        [,2]        [,3]        [,4]        [,5]
##  [1,]  0.76325258 -0.07147997 -0.10985848 -0.10985848 -0.11945311
##  [2,] -0.07147997  0.91676661 -0.08107460 -0.12904773 -0.13864236
##  [3,] -0.10985848 -0.08107460  1.07987527 -0.11945311 -0.12904773
##  [4,] -0.10985848 -0.12904773 -0.11945311  0.88798273 -0.03310146
```

```
##  [5,] -0.11945311 -0.13864236 -0.12904773 -0.03310146  1.10865915
##  [6,] -0.05229072 -0.07147997 -0.10985848 -0.10985848 -0.11945311
##  [7,] -0.09546654 -0.11465579 -0.15303430 -0.10506117 -0.16262893
##  [8,] -0.05708803 -0.12425042 -0.16262893 -0.16262893 -0.12425042
##  [9,] -0.08587191 -0.10506117 -0.14343967 -0.04749340 -0.15303430
## [10,] -0.06188534 -0.08107460 -0.07147997 -0.07147997 -0.12904773
##              [,6]        [,7]        [,8]        [,9]       [,10]
##  [1,] -0.05229072 -0.09546654 -0.05708803 -0.08587191 -0.06188534
##  [2,] -0.07147997 -0.11465579 -0.12425042 -0.10506117 -0.08107460
##  [3,] -0.10985848 -0.15303430 -0.16262893 -0.14343967 -0.07147997
##  [4,] -0.10985848 -0.10506117 -0.16262893 -0.04749340 -0.07147997
##  [5,] -0.11945311 -0.16262893 -0.12425042 -0.15303430 -0.12904773
##  [6,]  0.81122571 -0.09546654 -0.10506117 -0.08587191 -0.06188534
##  [7,] -0.09546654  1.06068602 -0.14823699 -0.08107460 -0.10506117
##  [8,] -0.10506117 -0.14823699  1.13744303 -0.13864236 -0.11465579
##  [9,] -0.08587191 -0.08107460 -0.13864236  0.93595586 -0.09546654
## [10,] -0.06188534 -0.10506117 -0.11465579 -0.09546654  0.79203646
```

- **Zusatzaufgabe**: Die Eigenwerte userer berechneten Matrix G betragen

```
eigen_mat_grm <- eigen(mat_grm)
eigen_mat_grm$values
```

```
##  [1] 1.297038e+00 1.281418e+00 1.236900e+00 1.095874e+00 1.039083e+00
##  [6] 9.450845e-01 9.263851e-01 8.499629e-01 8.221379e-01 8.881784e-16
```

Auch wenn der kleinste Eigenwert zwar positiv aber sehr nahe bei 0 ist, lässt sich keine zuverlässige Inverse berechnen. Wir müssen die Matrix mit einer Prozedur namens Bending anpassen. Mehr dazu folgt später.