LAB Prostate LinReg 4.R Focus on the description of the shrinkage meth- ods Ridge and Lasso.

Behrooz Filzadeh

2025-05-15

```
# Analyse der Prostata-Daten mit Ridge und Lasso Regression

# ====== Bibliotheken laden ======
# Diese Pakete brauchen wir für Datenanalyse, Visualisierung und Modellierung.
library(data.table)
library(ggplot2)
library(leaps)        # Für Auswahl von Variablen (Subset Selection)
library(glmnet)       # Für Ridge und Lasso Regression
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(psych)
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
library(DataExplorer)

# ====== Daten laden ======
# Wir laden die Prostata-Daten und machen daraus ein data.table.
prostateData <- read.table(file="prostate_data.csv")
prostateData <- as.data.table(prostateData)
table(prostateData$train)  # Zeigt wie viele Trainings- und Testdaten es gibt
```

```
##
## FALSE  TRUE
##    30    67
```

Erklärung: Hier laden wir die Daten aus einer Datei. Die Daten beinhalten Informationen über Prostata-Krebs. Wir schauen auch, wie viele Daten in Trainings- und Testgruppe sind.

```
# ====== Aufteilen in Trainings- und Testdaten ======
# Wir trennen die Daten in Trainings- und Testsets.
prostateData_train <- prostateData[train==TRUE]
prostateData_test <- prostateData[train==FALSE]
prostateData_train$train <- NULL
prostateData_test$train <- NULL
```

Erklärung: Wir trennen die Daten: ein Teil zum Trainieren des Modells, ein anderer zum Testen. Die Spalte train brauchen wir danach nicht mehr.

```
# ====== Standardisieren der Prädiktoren ======
# Standardisieren heißt: alle Variablen haben gleichen Maßstab (Mittelwert 0, SD 1).
# Das ist wichtig für Ridge und Lasso Regression.

# Trainingsdaten standardisieren
prostateData_train_scaled <- scale(prostateData_train[, 1:8])
prostateData_train_scaled <- as.data.table(prostateData_train_scaled)
prostateData_train_scaled[,lpsa:=prostateData_train$lpsa]

# Testdaten auch standardisieren (wir nehmen hier gleiche Methode)
prostateData_test_scaled <- scale(prostateData_test[, 1:8])
prostateData_test_scaled <- as.data.table(prostateData_test_scaled)
prostateData_test_scaled[,lpsa:=prostateData_test$lpsa]
```

Erklärung: Wir bringen alle numerischen Variablen auf dieselbe Skala. Das ist nötig, weil Ridge und Lasso sensibel auf verschiedene Größenordnungen sind.

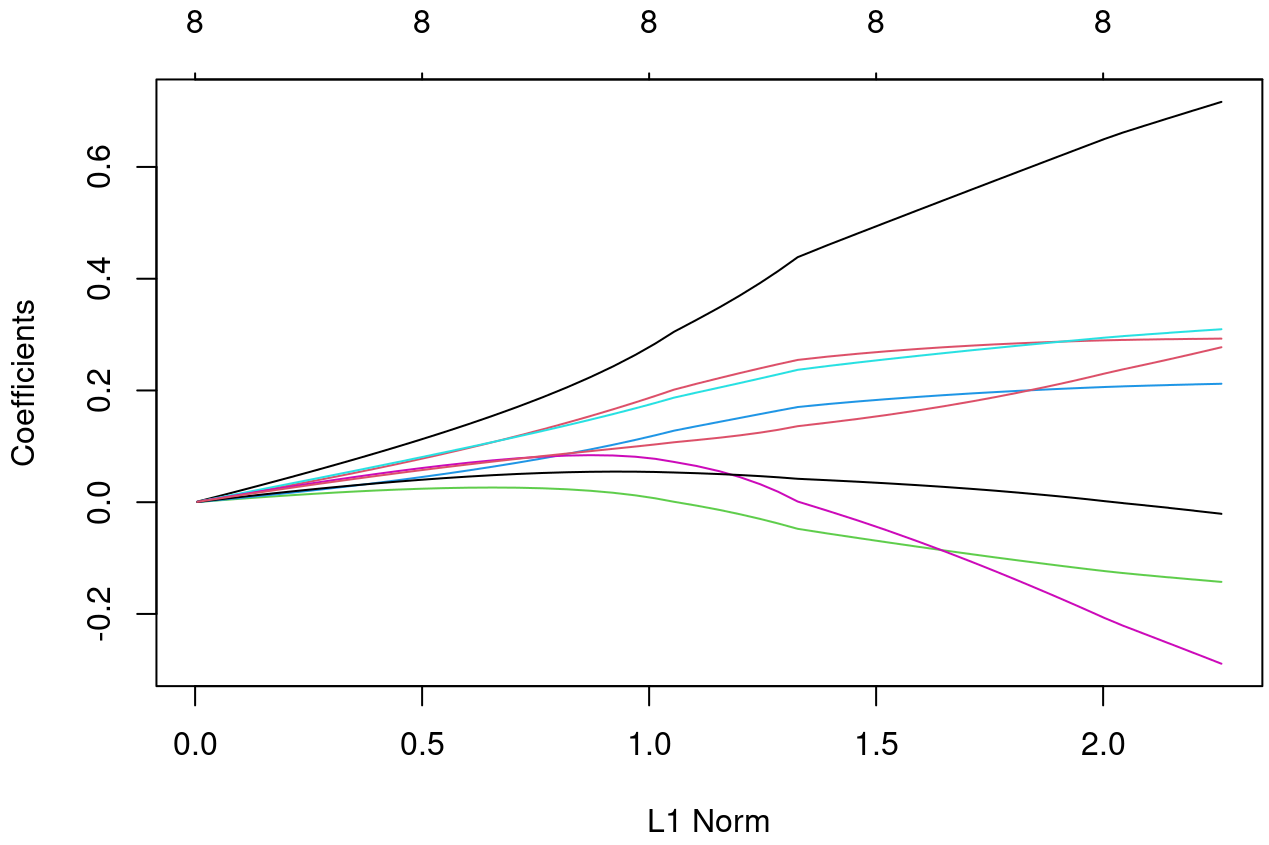
```
# ====== Ridge Regression ======
# Ridge benutzt alpha = 0 (kein Lasso-Effekt).
# Es schrumpft die Koeffizienten, aber setzt sie nicht auf 0.
# Ziel: Überanpassung (Overfitting) vermeiden und Stabilität erhöhen.

grid <- round(10^seq(3, -4, length = 100),2)
x <- as.matrix(prostateData_train_scaled[,1:8])
y <- prostateData_train_scaled$lpsa

ridgeModel <- glmnet(x, y, alpha = 0, standardize = FALSE, intercept = TRUE, lambda = grid)

plot(ridgeModel)  # Zeigt wie die Koeffizienten sich mit Lambda ändern
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```




```
print(ridgeModel)
```

```
##
## Call:  glmnet(x = x, y = y, alpha = 0, lambda = grid, standardize = FALSE,      intercept = T
RUE)
##
##     Df  %Dev  Lambda
## 1    8  0.41 1000.00
## 2    8  0.49  849.80
## 3    8  0.57  722.10
## 4    8  0.67  613.60
## 5    8  0.79  521.40
## 6    8  0.93  443.10
## 7    8  1.09  376.50
## 8    8  1.28  319.90
## 9    8  1.50  271.90
## 10   8  1.76  231.00
## 11   8  2.06  196.30
## 12   8  2.41  166.80
## 13   8  2.82  141.80
## 14   8  3.30  120.40
## 15   8  3.85  102.40
## 16   8  4.49   86.97
## 17   8  5.23   73.91
## 18   8  6.08   62.80
## 19   8  7.05   53.37
## 20   8  8.16   45.35
## 21   8  9.42   38.54
## 22   8 10.83   32.75
## 23   8 12.42   27.83
## 24   8 14.20   23.64
## 25   8 16.15   20.09
## 26   8 18.28   17.07
## 27   8 20.58   14.51
## 28   8 23.05   12.33
## 29   8 25.66   10.48
## 30   8 28.40    8.90
## 31   8 31.21    7.56
## 32   8 34.05    6.43
## 33   8 36.92    5.46
## 34   8 39.74    4.64
## 35   8 42.50    3.94
## 36   8 45.13    3.35
## 37   8 47.61    2.85
## 38   8 49.97    2.42
## 39   8 52.13    2.06
## 40   8 54.13    1.75
## 41   8 56.01    1.48
## 42   8 57.65    1.26
## 43   8 59.14    1.07
## 44   8 60.48    0.91
## 45   8 61.71    0.77
## 46   8 62.72    0.66
## 47   8 63.69    0.56
```

```
## 48   8 64.50    0.48
## 49   8 65.34    0.40
## 50   8 65.99    0.34
## 51   8 66.55    0.29
## 52   8 67.01    0.25
## 53   8 67.48    0.21
## 54   8 67.83    0.18
## 55   8 68.18    0.15
## 56   8 68.40    0.13
## 57   8 68.62    0.11
## 58   8 68.84    0.09
## 59   8 68.94    0.08
## 60   8 69.03    0.07
## 61   8 69.12    0.06
## 62   8 69.21    0.05
## 63   8 69.28    0.04
## 64   8 69.28    0.04
## 65   8 69.34    0.03
## 66   8 69.34    0.03
## 67   8 69.39    0.02
## 68   8 69.39    0.02
## 69   8 69.39    0.02
## 70   8 69.43    0.01
## 71   8 69.43    0.01
## 72   8 69.43    0.01
## 73   8 69.43    0.01
## 74   8 69.43    0.01
## 75   8 69.43    0.01
## 76   8 69.44    0.00
## 77   8 69.44    0.00
## 78   8 69.44    0.00
## 79   8 69.44    0.00
## 80   8 69.44    0.00
## 81   8 69.44    0.00
## 82   8 69.44    0.00
## 83   8 69.44    0.00
## 84   8 69.44    0.00
## 85   8 69.44    0.00
## 86   8 69.44    0.00
## 87   8 69.44    0.00
## 88   8 69.44    0.00
## 89   8 69.44    0.00
## 90   8 69.44    0.00
## 91   8 69.44    0.00
## 92   8 69.44    0.00
## 93   8 69.44    0.00
## 94   8 69.44    0.00
## 95   8 69.44    0.00
## 96   8 69.44    0.00
## 97   8 69.44    0.00
## 98   8 69.44    0.00
```

```
## 99   8 69.44    0.00
## 100  8 69.44    0.00
```

```
coef(ridgeModel, s = 0.21)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  2.45234509
## lcavol       0.52194400
## lweight      0.27414004
## age         -0.07943987
## lbph         0.18834838
## svi          0.26172337
## lcp         -0.06953063
## gleason      0.03056797
## pgg45        0.16404778
```

```
predict_model3 <- predict(ridgeModel, as.matrix(prostateData_test_scaled[,1:8]), s = 0.21)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```

```
mse_model3 <- mean((predict_model3 - prostateData_test_scaled$lpsa)^2)
mse_model3
```

```
## [1] 0.5055012
```

```
sErr <- sqrt(var((prostateData_test_scaled$lpsa - predict_model3)^2) / 30)
sErr
```

```
##           s1
## s1 0.1563597
```

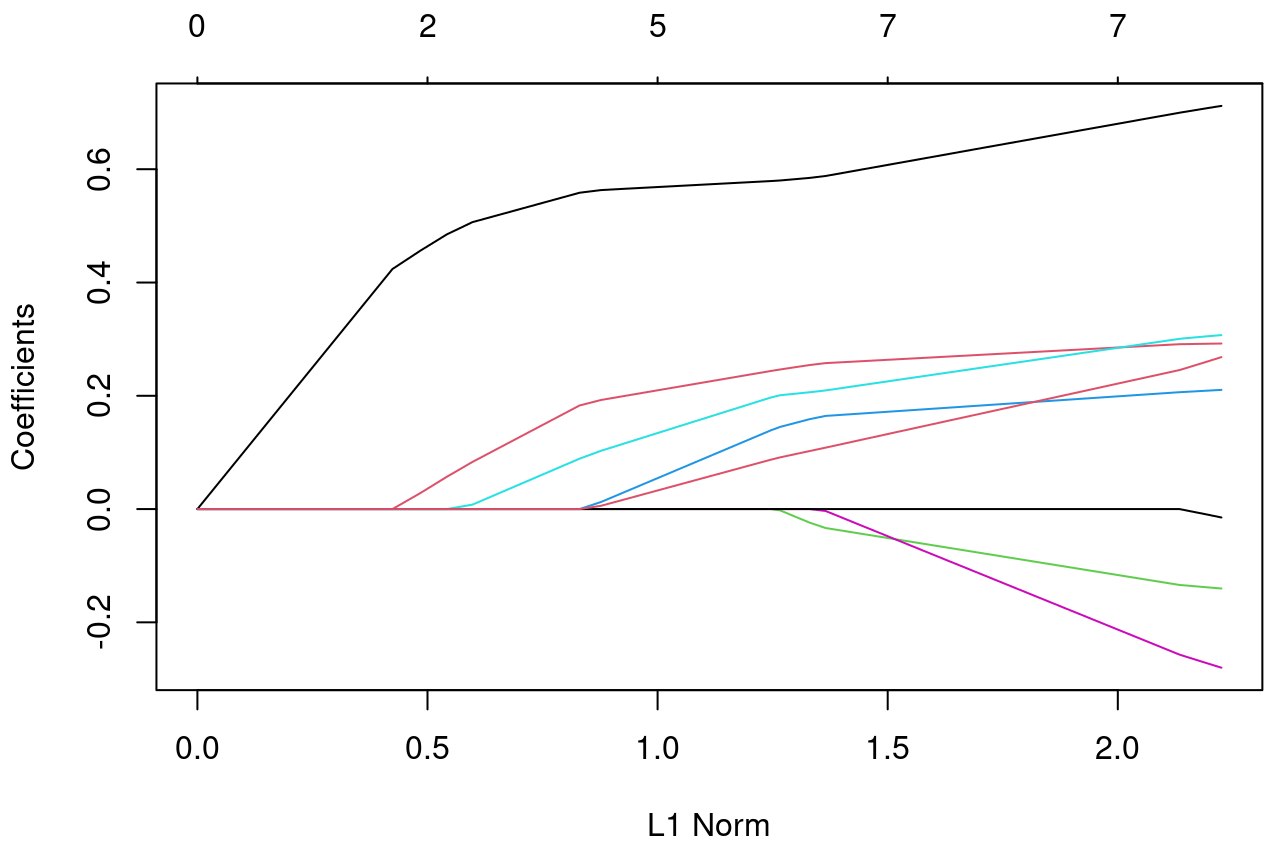
Erklärung: Ridge Regression hilft uns, ein stabileres Modell zu machen, besonders wenn es viele Prädiktoren gibt. Mit Lambda können wir steuern, wie stark die Koeffizienten “geschrumpft” werden. Hier testen wir das Modell auf Testdaten und berechnen den Fehler (MSE).

```
# ====== Lasso Regression ======
# Lasso benutzt alpha = 1 (volle Lasso-Penalisierung).
# Es kann manche Koeffizienten auf genau 0 setzen => Variable-Selection.
# Gut, wenn man nur wichtige Variablen behalten will.

grid <- 10^seq(3, -4, length = 100)
x <- as.matrix(prostateData_train_scaled[,1:8])
y <- prostateData_train_scaled$lpsa

lassoModel <- glmnet(x, y, alpha = 1, standardize = FALSE, intercept = TRUE)

plot(lassoModel)  # Zeigt wie viele Koeffizienten auf 0 gehen
```




```
print(lassoModel)
```

```
##
## Call:  glmnet(x = x, y = y, alpha = 1, standardize = FALSE, intercept = TRUE)
##
##    Df  %Dev  Lambda
## 1   0  0.00 0.87230
## 2   1  9.13 0.79480
## 3   1 16.70 0.72420
## 4   1 22.99 0.65990
## 5   1 28.22 0.60120
## 6   1 32.55 0.54780
## 7   1 36.15 0.49920
## 8   1 39.14 0.45480
## 9   2 42.81 0.41440
## 10  2 45.98 0.37760
## 11  3 48.77 0.34410
## 12  3 51.31 0.31350
## 13  3 53.42 0.28560
## 14  3 55.18 0.26030
## 15  3 56.63 0.23710
## 16  3 57.84 0.21610
## 17  5 59.17 0.19690
## 18  5 60.45 0.17940
## 19  5 61.51 0.16350
## 20  5 62.39 0.14890
## 21  5 63.12 0.13570
## 22  5 63.72 0.12360
## 23  5 64.23 0.11270
## 24  5 64.65 0.10270
## 25  5 64.99 0.09353
## 26  5 65.28 0.08522
## 27  5 65.52 0.07765
## 28  5 65.72 0.07075
## 29  5 65.89 0.06447
## 30  6 66.05 0.05874
## 31  6 66.32 0.05352
## 32  6 66.53 0.04877
## 33  7 66.76 0.04444
## 34  7 67.21 0.04049
## 35  7 67.59 0.03689
## 36  7 67.90 0.03361
## 37  7 68.16 0.03063
## 38  7 68.37 0.02791
## 39  7 68.55 0.02543
## 40  7 68.70 0.02317
## 41  7 68.82 0.02111
## 42  7 68.93 0.01924
## 43  7 69.01 0.01753
## 44  7 69.08 0.01597
## 45  7 69.14 0.01455
## 46  7 69.19 0.01326
## 47  7 69.23 0.01208
## 48  7 69.26 0.01101
```

```
## 49  7 69.29 0.01003
## 50  7 69.31 0.00914
## 51  7 69.33 0.00833
## 52  7 69.35 0.00759
## 53  7 69.36 0.00691
## 54  7 69.37 0.00630
## 55  7 69.38 0.00574
## 56  7 69.39 0.00523
## 57  8 69.39 0.00476
## 58  8 69.40 0.00434
## 59  8 69.41 0.00396
## 60  8 69.41 0.00360
## 61  8 69.42 0.00328
## 62  8 69.42 0.00299
## 63  8 69.42 0.00273
## 64  8 69.43 0.00248
## 65  8 69.43 0.00226
## 66  8 69.43 0.00206
## 67  8 69.43 0.00188
## 68  8 69.43 0.00171
## 69  8 69.43 0.00156
## 70  8 69.43 0.00142
## 71  8 69.43 0.00130
```

```
coef(lassoModel, s = 0.3)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                     s1
## (Intercept) 2.45234509
## lcavol      0.52446582
## lweight     0.11756707
## age         .
## lbph        .
## svi         0.03580105
## lcp         .
## gleason     .
## pgg45       .
```

```
predict_model4 <- predict(lassoModel, as.matrix(prostateData_test_scaled[,1:8]), s = 0.5)
mse_model4 <- mean((predict_model3 - prostateData_test_scaled$lpsa)^2)  # Achtung: predict_model
3 statt 4!
mse_model4
```

```
## [1] 0.5055012
```

```
sErr <- sqrt(var((prostateData_test_scaled$lpsa - predict_model4)^2) / 30)
sErr
```

```
##           s1
## s1 0.2400819
```

Erklärung: Lasso ist sehr hilfreich, wenn wir viele unnötige Variablen haben. Es hilft, ein sparsames Modell zu bauen. Der Unterschied zu Ridge ist, dass es wirklich Variablen entfernt. Hinweis: In der Zeile mse_model4 wird fälschlich predict_model3 verwendet – sollte predict_model4 sein.