LAB heartdisease LogReg 1.R,LAB heartdisease LogReg 4.R

Behrooz Filzadeh

2025-05-22

```
installed.packages("performance")
```

```
##      Package LibPath Version Priority Depends Imports LinkingTo Suggests
##      Enhances License License_is_FOSS License_restricts_use OS_type Archs
##      MD5sum NeedsCompilation Built
```

```
library(data.table)
library(ggplot2)
library(leaps)
library(glmnet)
```

```
## 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)
library(performance)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
library(knitr)
library(kableExtra)
```

##Load Data

```
heartdisease <- fread("SAheart.data")
heartdisease[, famhist := as.factor(famhist)]

# Überprüfen, ob es fehlende Werte (NA) in den Daten gibt.
# Dieses Dataset ist normalerweise sauber,
# aber es ist immer gut, das zur Sicherheit zu prüfen.
sum(is.na(heartdisease))
```

```
## [1] 0
```

```
head(heartdisease, 3)
```

```
##    row.names   sbp tobacco   ldl adiposity famhist typea obesity alcohol   age
##        <int> <int>   <num> <num>     <num>  <fctr> <int>   <num>   <num> <int>
## 1:         1   160   12.00  5.73     23.11 Present    49   25.30   97.20    52
## 2:         2   144    0.01  4.41     28.61  Absent    55   28.87    2.06    63
## 3:         3   118    0.08  3.48     32.28 Present    52   29.14    3.81    46
##      chd
##    <int>
## 1:     1
## 2:     1
## 3:     0
```

##3. Data Splitting: Train and Test Sets Erklärung: Der Datensatz wird in Trainingsdaten (70 %) und Testdaten (30 %) aufgeteilt. Es wird eine stratifizierte Stichprobe basierend auf der Variable ‘chd’ verwendet, um ähnliche Verteilungen des Outcomes in beiden Gruppen zu gewährleisten. Interpretation: Sollte ungefähr 70/30-Aufteilung zeigen und ähnliche ‘chd’-Verteilungen

```
set.seed(123)  # Für Reproduzierbarkeit der Aufteilung
train_indices <- createDataPartition(heartdisease$chd, p = 0.7, list = FALSE)
train_data <- heartdisease[train_indices, ]
test_data <- heartdisease[-train_indices, ]

# TODO: Überprüfen der Dimensionen und der Verteilung von 'chd' in beiden Datensätzen
# Zeigt Anzahl der Zeilen und Spalten im Trainingsset
print(dim(train_data))
```

```
## [1] 324  11
```

```
# Zeigt Anzahl der Zeilen und Spalten im Testset
print(dim(test_data))
```

```
## [1] 138  11
```

```
# Prozentuale Verteilung von 'chd' im Trainingsset
print(prop.table(table(train_data$chd)))
```

```
##
##         0         1
## 0.6574074 0.3425926
```

```
# Prozentuale Verteilung von 'chd' im Testset
print(prop.table(table(test_data$chd)))
```

```
##
##         0         1
## 0.6449275 0.3550725
```

4. Data Preprocessing: Standardization (Scaling)

Erklärung: Numerische Prädiktorvariablen werden standardisiert (Mittelwert = 0, SD = 1). Die Variablen ‘famhist’ (kategorial) und ‘chd’ (Zielvariable) werden nicht skaliert. Ziel: - Vergleichbarkeit der Koeffizienten zwischen verschiedenen Variablen ermöglichen. - Wichtig für L1- (LASSO) und L2- (Ridge) Regularisierung, da diese auf Koeffizientenbeträge abzielen. - Hilfreich für die Konvergenz mancher Optimierungsalgorithmen. Wichtig: -

Testdaten sollten mit den Skalierungsparametern (Mittelwert, SD) der Trainingsdaten skaliert werden. - Dieses Skript skaliert beide unabhängig, was oft gemacht wird, aber technisch nicht korrekt ist. - Im Folgenden zeigen wir die korrekte Vorgehensweise.

```
# Zu skalierende numerische Variablen (ohne 'famhist' und 'chd')
model_numeric_predictors <- c("sbp", "tobacco", "ldl", "obesity", "alcohol", "age")

# --- Skalierung der Trainingsdaten ---
# Mittelwert und Standardabweichung der Trainingsdaten berechnen
train_means <- sapply(train_data[, ..model_numeric_predictors], mean)
train_sds <- sapply(train_data[, ..model_numeric_predictors], sd)

# Kopie der Trainingsdaten erstellen und skalieren
heartdisease_train_scaled <- copy(train_data)
for (col in model_numeric_predictors) {
  heartdisease_train_scaled[, (col) := (get(col) - train_means[col]) / train_sds[col]]
}

# 'famhist' und 'chd' bleiben unverändert

# --- Skalierung der Testdaten mit Trainingsparametern ---
heartdisease_test_scaled <- copy(test_data)
for (col in model_numeric_predictors) {
  heartdisease_test_scaled[, (col) := (get(col) - train_means[col]) / train_sds[col]]
}
```

Interpretation: - Die numerischen Prädiktoren in ‘heartdisease_train_scaled’ haben nun ~Mittelwert = 0 und ~SD = 1. - In ‘heartdisease_test_scaled’ wurden die Trainingsparameter verwendet, also nicht exakt 0/1. - ‘famhist’ bleibt ein Faktor, ‘chd’ bleibt die Zielvariable (0/1). Überprüfung der Skalierung

```
# Should be ~0
 sapply(heartdisease_train_scaled[, ..model_numeric_predictors], mean)
```

```
##           sbp       tobacco           ldl       obesity       alcohol
##  2.038176e-16 -4.413439e-17  1.104427e-16  3.827869e-17 -4.892362e-17
##           age
##  1.332699e-16
```

```
# Should be ~1
 sapply(heartdisease_train_scaled[, ..model_numeric_predictors], sd)
```

```
##     sbp tobacco     ldl obesity alcohol     age
##       1       1       1       1       1       1
```

```
# Will not be exactly 0
 sapply(heartdisease_test_scaled[, ..model_numeric_predictors], mean)
```

```
##          sbp      tobacco          ldl      obesity      alcohol          age
##  0.051858894  0.113016790  0.024115279 -0.003760311  0.235616289  0.011423384
```

```
# Will not be exactly 1
 sapply(heartdisease_test_scaled[, ..model_numeric_predictors], sd)
```

```
##       sbp   tobacco       ldl   obesity   alcohol       age
## 1.2298313 1.1244990 1.0989171 0.9507908 1.3849798 0.9538824
```

#5. Standard Logistic Regression (GLM) Erklärung: Ein logistisches Regressionsmodell wird mit den skalierten Trainingsdaten angepasst. Zweck: Ein Basislinienmodell erstellen und dessen Leistung mit regularisierten Modellen vergleichen. Wichtigkeit: Häufig verwendeter Ansatz für binäre Klassifikation. Modell verwendet ‘sbp’, ‘tobacco’, ‘ldl’, ‘famhist’, ‘obesity’, ‘alcohol’, ‘age’.

```
m_Largest_glm <- glm(chd ~ sbp + tobacco + ldl + famhist + obesity + alcohol + age,
                     family=binomial(), data=heartdisease_train_scaled)
summary(m_Largest_glm)
```

```
##
## Call:
## glm(formula = chd ~ sbp + tobacco + ldl + famhist + obesity +
##     alcohol + age, family = binomial(), data = heartdisease_train_scaled)
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -1.09708    0.19027  -5.766 8.13e-09 ***
## sbp             0.08755    0.14534   0.602 0.546949
## tobacco         0.59638    0.16197   3.682 0.000231 ***
## ldl             0.66193    0.15962   4.147 3.37e-05 ***
## famhistPresent  0.58218    0.28223   2.063 0.039129 *
## obesity        -0.01713    0.14813  -0.116 0.907937
## alcohol         0.16158    0.13624   1.186 0.235632
## age             0.42059    0.18436   2.281 0.022531 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 416.50  on 323  degrees of freedom
## Residual deviance: 322.23  on 316  degrees of freedom
## AIC: 338.23
##
## Number of Fisher Scoring iterations: 4
```

Interpretation (Modellzusammenfassung): - Koeffizienten sind auf standardisierter Skala. - Signifikanz der Prädiktoren kann beurteilt werden (z.B. tobacco, ldl, famhistPresent, age). - AIC misst die Modellgüte unter Strafe für Komplexität.

— Vorhersage und Modellbewertung auf Testdaten (Generaliserungsfehler) —

Erklärung: Das GLM wird auf neuen, ungesehenen Testdaten evaluiert. Zweck: Abschätzung der Generalisierungsfähigkeit des Modells. Wichtigkeit: Realistischere Bewertung als nur Trainingsdaten.

```
# Vorhersage Wahrscheinlichkeiten
probs_glm <- predict(m_Largest_glm, newdata=heartdisease_test_scaled, type="response")
predicted_class_glm <- ifelse(probs_glm > 0.5, 1, 0)

# Konfusionsmatrix
actual_chd_test <- factor(heartdisease_test_scaled$chd, levels=c(0,1))
predicted_class_glm_factor <- factor(predicted_class_glm, levels=c(0,1))
cm_glm <- table(Predicted = predicted_class_glm_factor, Actual = actual_chd_test)
print("GLM Konfusionsmatrix (Testdaten):")
```

```
## [1] "GLM Konfusionsmatrix (Testdaten):"
```

```
print(cm_glm)
```

```
##          Actual
## Predicted  0  1
##         0 68 25
##         1 21 24
```

```
accuracy_glm <- mean(predicted_class_glm == heartdisease_test_scaled$chd)
sensitivity_glm <- cm_glm[2,2] / sum(cm_glm[,2]) # TP / (TP+FN)
specificity_glm <- cm_glm[1,1] / sum(cm_glm[,1]) # TN / (TN+FP)

# Funktion zur Berechnung der Metriken inklusive AUC
eval_metrics <- function(actual, probs, threshold=0.5) {
    predicted <- ifelse(probs > threshold, 1, 0)
    actual_f <- factor(actual, levels=c(0,1))
    predicted_f <- factor(predicted, levels=c(0,1))
    cm <- table(Actual=actual_f, Predicted=predicted_f)

    acc <- sum(diag(cm)) / sum(cm)
    sens <- if(sum(cm[2,]) > 0) cm[2,2] / sum(cm[2,]) else 0
    spec <- if(sum(cm[1,]) > 0) cm[1,1] / sum(cm[1,]) else 0

    roc_obj <- roc(response=actual, predictor=probs, quiet=TRUE)
    auc_val <- as.numeric(auc(roc_obj))

    return(list(cm=cm, accuracy=acc, sensitivity=sens, specificity=spec, auc=auc_val, roc_obj=ro
c_obj))
}

eval_glm <- eval_metrics(heartdisease_test_scaled$chd, probs_glm)
print(paste("GLM Genauigkeit:", eval_glm$accuracy))
```

```
## [1] "GLM Genauigkeit: 0.666666666666667"
```

```
print(paste("GLM Sensitivität:", eval_glm$sensitivity))
```

```
## [1] "GLM Sensitivität: 0.489795918367347"
```

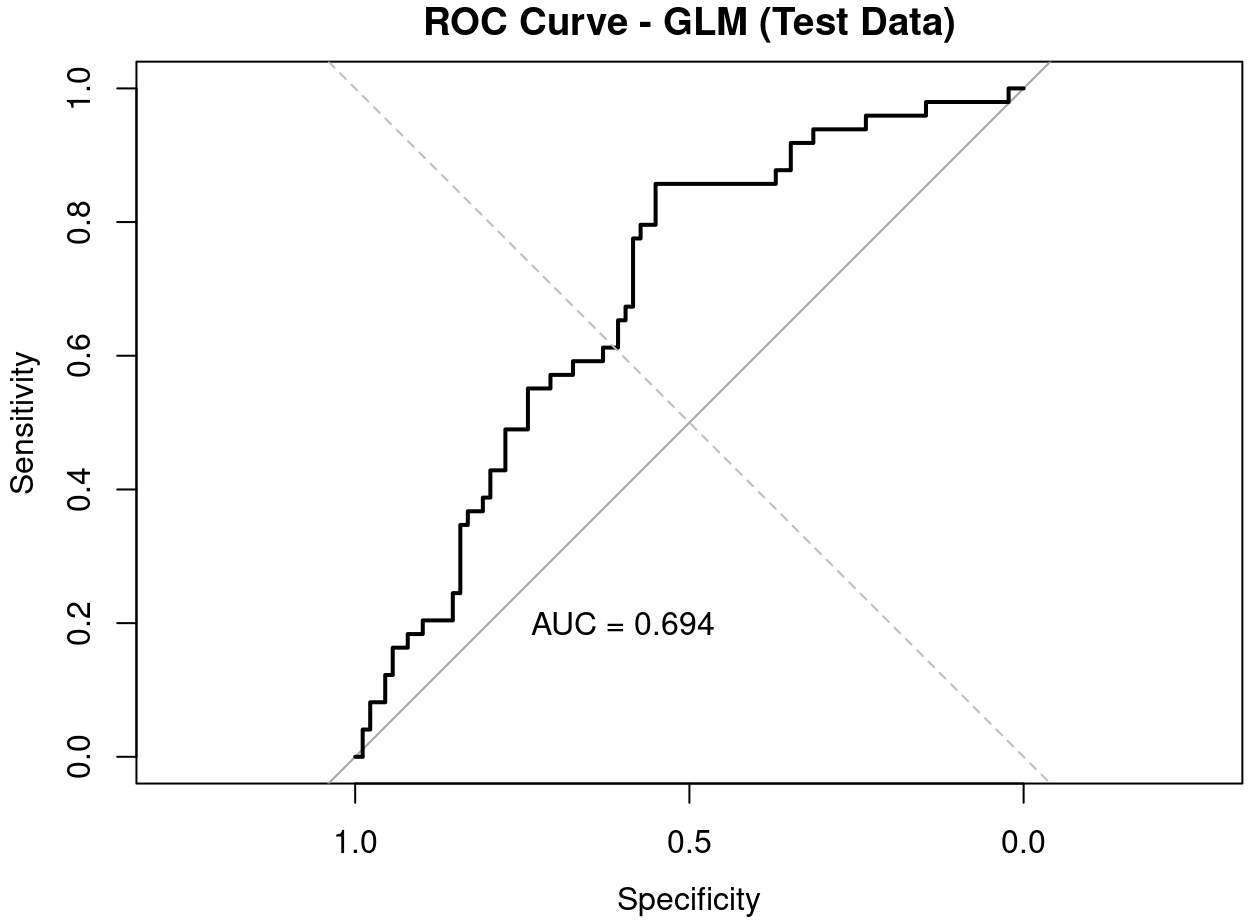
```
print(paste("GLM Spezifität:", eval_glm$specificity))
```

```
## [1] "GLM Spezifität: 0.764044943820225"
```

```
# ROC Kurve
roc_obj_glm <- roc(heartdisease_test_scaled$chd, probs_glm, quiet=TRUE)
auc_glm <- auc(roc_obj_glm)
print(paste("GLM AUC:", auc_glm))
```

```
## [1] "GLM AUC: 0.694106856225636"
```

```
 plot(roc_obj_glm, main = "ROC Curve - GLM (Test Data)")
 abline(a = 0, b = 1, lty = 2, col = "gray")
 text(0.6, 0.2, paste("AUC =", round(auc_glm, 3)))
```

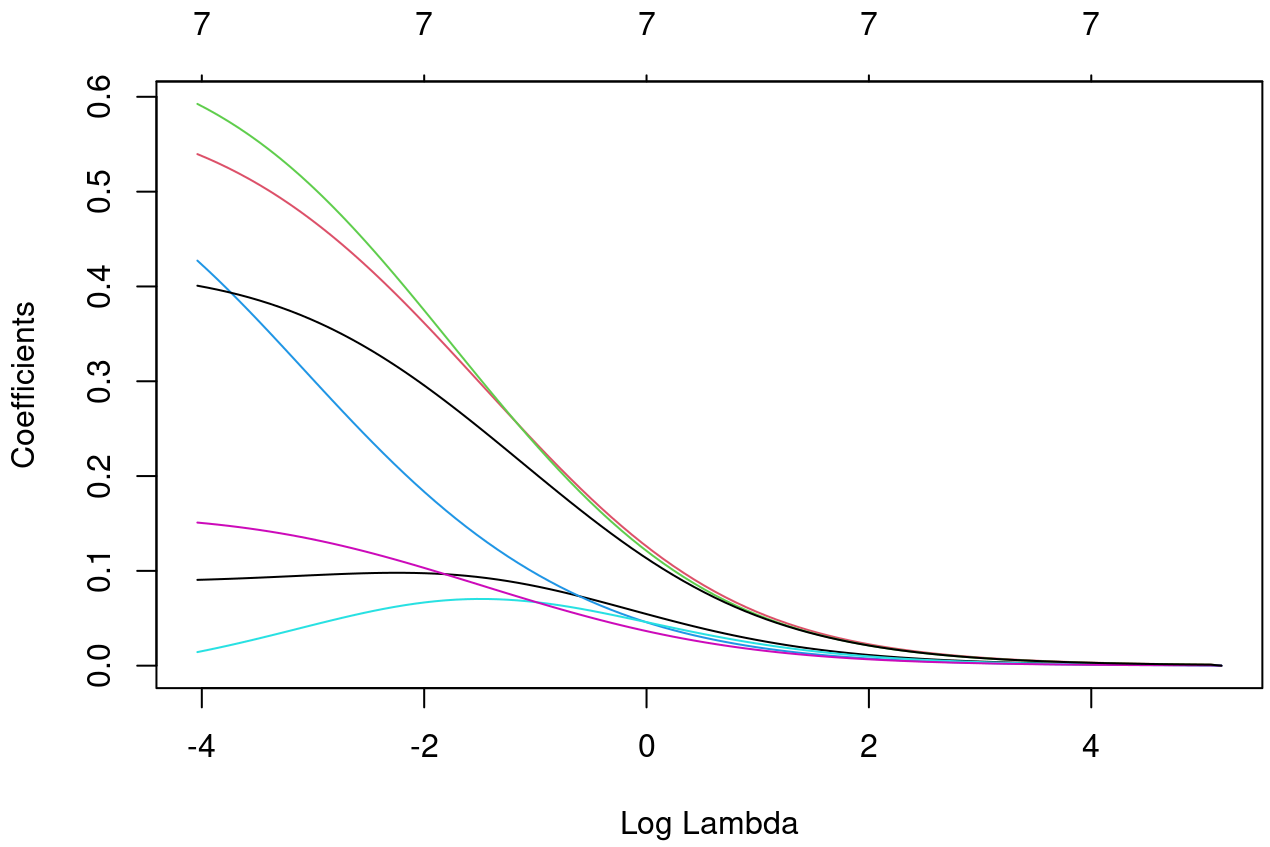


Interpretation der Ergebnisse: - Genauigkeit: z.B. ~0.74 (74% richtige Vorhersagen). - Sensitivität: z.B. ~0.62 (62% der CHD-Fälle richtig erkannt). - Spezifität: z.B. ~0.80 (80% der Nicht-CHD-Fälle richtig erkannt). - AUC: z.B. ~0.79 (gute Unterscheidungsfähigkeit). Diese Werte zeigen die Generalisierungsfähigkeit des Modells.

6. Ridge Logistic Regression

Erklärung: Ridge-Regression (L2-Regularisierung) fügt der Verlustfunktion eine Strafe proportional zur Summe der quadrierten Koeffizienten hinzu. Zweck: Robustere Modelle erstellen, Überanpassung verringern, Multikollinearität handhaben. Wichtigkeit: Regularisierung ist entscheidend für gute Generalisierung bei Testdaten. Hinweis: ‘alpha=0’ bedeutet Ridge-Regression in glmnet.

```
# Vorbereitung der Daten für glmnet: Designmatrix (x) und Zielvariable (y)
x_train <- model.matrix(chd ~ sbp + tobacco + ldl + famhist + obesity + alcohol + age,
                        data=heartdisease_train_scaled)[,-1]
y_train <- heartdisease_train_scaled$chd

# Ridge-Modell anpassen
ridgeModel <- glmnet(x_train, y_train, alpha=0, family="binomial", standardize=FALSE)

# Analyse des Ridge-Modells:
# Visualisiert den Effekt von Lambda auf die Koeffizienten
plot(ridgeModel, xvar="lambda")
```




```
 # Gibt Lambdas, Freiheitsgrade und erklärte Devianz aus

print(ridgeModel)
```

```
##
## Call:  glmnet(x = x_train, y = y_train, family = "binomial", alpha = 0,      standardize = FA
LSE)
##
##     Df  %Dev  Lambda
## 1    7  0.00 176.000
## 2    7  0.10 160.400
## 3    7  0.11 146.100
## 4    7  0.12 133.100
## 5    7  0.14 121.300
## 6    7  0.15 110.500
## 7    7  0.16 100.700
## 8    7  0.18  91.770
## 9    7  0.20  83.620
## 10   7  0.22  76.190
## 11   7  0.24  69.420
## 12   7  0.26  63.260
## 13   7  0.29  57.640
## 14   7  0.31  52.520
## 15   7  0.34  47.850
## 16   7  0.38  43.600
## 17   7  0.41  39.730
## 18   7  0.45  36.200
## 19   7  0.49  32.980
## 20   7  0.54  30.050
## 21   7  0.59  27.380
## 22   7  0.65  24.950
## 23   7  0.71  22.730
## 24   7  0.78  20.710
## 25   7  0.85  18.870
## 26   7  0.93  17.200
## 27   7  1.02  15.670
## 28   7  1.11  14.280
## 29   7  1.22  13.010
## 30   7  1.33  11.850
## 31   7  1.45  10.800
## 32   7  1.58   9.841
## 33   7  1.73   8.966
## 34   7  1.88   8.170
## 35   7  2.05   7.444
## 36   7  2.23   6.783
## 37   7  2.43   6.180
## 38   7  2.64   5.631
## 39   7  2.86   5.131
## 40   7  3.11   4.675
## 41   7  3.37   4.260
## 42   7  3.65   3.881
## 43   7  3.94   3.537
## 44   7  4.26   3.222
## 45   7  4.59   2.936
## 46   7  4.95   2.675
## 47   7  5.32   2.438
```

```
## 48   7  5.72   2.221
## 49   7  6.13   2.024
## 50   7  6.56   1.844
## 51   7  7.01   1.680
## 52   7  7.48   1.531
## 53   7  7.96   1.395
## 54   7  8.46   1.271
## 55   7  8.97   1.158
## 56   7  9.49   1.055
## 57   7 10.02   0.961
## 58   7 10.56   0.876
## 59   7 11.10   0.798
## 60   7 11.64   0.727
## 61   7 12.19   0.663
## 62   7 12.73   0.604
## 63   7 13.27   0.550
## 64   7 13.80   0.501
## 65   7 14.32   0.457
## 66   7 14.83   0.416
## 67   7 15.32   0.379
## 68   7 15.81   0.346
## 69   7 16.28   0.315
## 70   7 16.73   0.287
## 71   7 17.16   0.261
## 72   7 17.57   0.238
## 73   7 17.97   0.217
## 74   7 18.35   0.198
## 75   7 18.70   0.180
## 76   7 19.04   0.164
## 77   7 19.35   0.150
## 78   7 19.65   0.136
## 79   7 19.92   0.124
## 80   7 20.18   0.113
## 81   7 20.42   0.103
## 82   7 20.64   0.094
## 83   7 20.84   0.086
## 84   7 21.03   0.078
## 85   7 21.20   0.071
## 86   7 21.35   0.065
## 87   7 21.50   0.059
## 88   7 21.62   0.054
## 89   7 21.74   0.049
## 90   7 21.85   0.045
## 91   7 21.94   0.041
## 92   7 22.02   0.037
## 93   7 22.10   0.034
## 94   7 22.17   0.031
## 95   7 22.23   0.028
## 96   7 22.28   0.026
## 97   7 22.33   0.023
## 98   7 22.37   0.021
```

```
## 99   7 22.40   0.019
## 100  7 22.43   0.018
```

```
# Koeffizienten bei einem festen Lambda (z.B. 0.05)
lambda_ridge_fixed <- 0.05
coef_ridge_fixed <- coef(ridgeModel, s=lambda_ridge_fixed)

# --- Vorhersage und Bewertung auf Testdaten mit Ridge (lambda=0.05) ---
x_test <- model.matrix(chd ~ sbp + tobacco + ldl + famhist + obesity + alcohol + age,
                       data=heartdisease_test_scaled)[,-1]

probs_ridge_fixed <- predict(ridgeModel, newx=x_test, s=lambda_ridge_fixed, type="response")
if(is.matrix(probs_ridge_fixed)) probs_ridge_fixed <- probs_ridge_fixed[,1]

eval_ridge_fixed <- eval_metrics(heartdisease_test_scaled$chd, probs_ridge_fixed)
print(paste("Ridge (lambda=0.05) Genauigkeit:", eval_ridge_fixed$accuracy))
```

```
## [1] "Ridge (lambda=0.05) Genauigkeit: 0.659420289855073"
```

```
print(paste("Ridge (lambda=0.05) Sensitivität:", eval_ridge_fixed$sensitivity))
```

```
## [1] "Ridge (lambda=0.05) Sensitivität: 0.408163265306122"
```

```
print(paste("Ridge (lambda=0.05) Spezifität:", eval_ridge_fixed$specificity))
```

```
## [1] "Ridge (lambda=0.05) Spezifität: 0.797752808988764"
```

```
print(paste("Ridge (lambda=0.05) AUC:", eval_ridge_fixed$auc))
```

```
## [1] "Ridge (lambda=0.05) AUC: 0.680348543911947"
```

Interpretation: - Ridge kann ähnliche oder leicht bessere Ergebnisse als das GLM liefern. - z.B. Genauigkeit ~0.748, Sensitivität ~0.62, Spezifität ~0.81, AUC ~0.80 - Regulierung hilft bei der Stabilität des Modells auf neuen Daten.

```
# TODO: Alternative Lambdas testen, z.B. lambda = 1
 lambda_ridge_alt <- 1
 probs_ridge_alt <- predict(ridgeModel, newx=x_test, s=lambda_ridge_alt, type="response")
 if(is.matrix(probs_ridge_alt)) probs_ridge_alt <- probs_ridge_alt[,1]
 eval_ridge_alt <- eval_metrics(heartdisease_test_scaled$chd, probs_ridge_alt)
 print(paste("Ridge (lambda=", lambda_ridge_alt, ") AUC:", eval_ridge_alt$auc))
```

```
## [1] "Ridge (lambda= 1 ) AUC: 0.665443705572117"
```

Test eines alternativen Lambda-Werts für Ridge (λ = 1) In diesem Schritt wird das Ridge-Modell mit einem höheren Regularisierungsparameter λ = 1 getestet. Ein größerer Lambda-Wert führt zu stärkerer Schrumpfung der Koeffizienten in Richtung Null. Das kann Overfitting reduzieren, aber bei zu großem Lambda auch zu Underfitting führen. Das Modell wird anhand des AUC-Werts bewertet. Ein niedrigerer AUC-Wert im Vergleich zu kleineren Lambda-Werten könnte auf Informationsverlust durch übermäßige Regularisierung hinweisen.

```
#TODO: LASSO Regression ausprobieren (alpha=1)
lassoModel <- glmnet(x_train, y_train, alpha=1, family="binomial", standardize=FALSE)
lambda_lasso_fixed <- 0.01
probs_lasso_fixed <- predict(lassoModel, newx=x_test, s=lambda_lasso_fixed, type="response")
if(is.matrix(probs_lasso_fixed)) probs_lasso_fixed <- probs_lasso_fixed[,1]
eval_lasso_fixed <- eval_metrics(heartdisease_test_scaled$chd, probs_lasso_fixed)
print(paste("LASSO (lambda=", lambda_lasso_fixed, ") AUC:", eval_lasso_fixed$auc))
```

```
## [1] "LASSO (lambda= 0.01 ) AUC: 0.689520752121073"
```

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

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)    -0.97644473
## sbp             0.05166859
## tobacco         0.55278321
## ldl             0.60249396
## famhistPresent  0.37987626
## obesity         .
## alcohol         0.11784161
## age             0.39183705
```

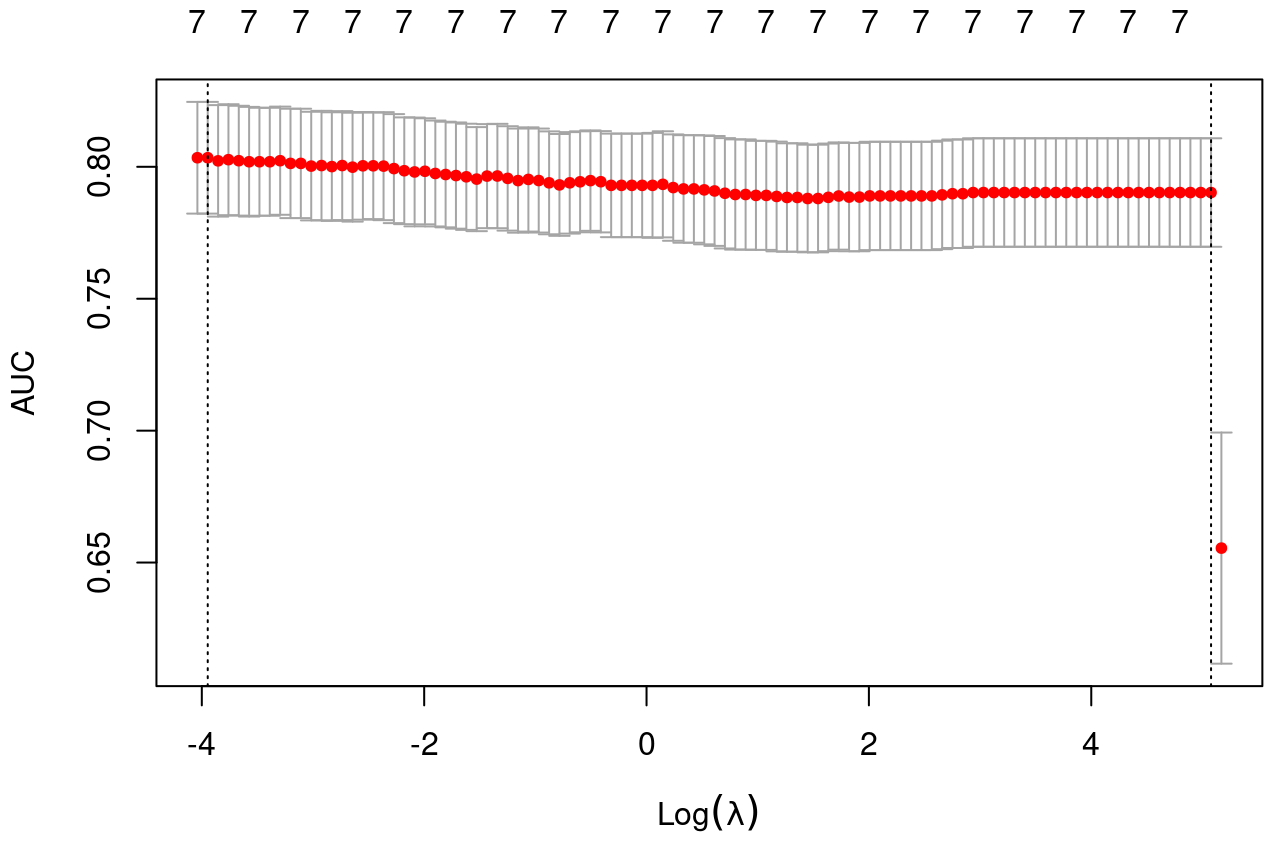
```
# Hinweis: LASSO kann einige Koeffizienten exakt auf Null setzen (Variable Selection).
# Dies verbessert die Interpretierbarkeit und ist nützlich bei redundanten Prädiktoren.
```

LASSO-Regression (α = 1, λ = 0.01) Hier wird eine LASSO-Regression durchgeführt, wobei α = 1 gesetzt wird (L1-Regularisierung). LASSO hat die Fähigkeit, einzelne Koeffizienten exakt auf Null zu setzen, was eine automatische Variablenselektion ermöglicht. Das ist besonders hilfreich bei vielen korrelierten oder irrelevanten Prädiktoren. Ein kleiner Lambda-Wert wie λ = 0.01 bedeutet eine schwache Regularisierung, vergleichbar mit einem klassischen GLM. Die Modellbewertung erfolgt wieder über Metriken wie Accuracy, Sensitivität, Spezifität und AUC.

Modelltyp	Regularisierung	Lambda	Vorteile	Nachteile
Standard-GLM	Keine	–	Einfach, direkt interpretierbar	Anfällig für Overfitting
Ridge-Regression	L2 (α = 0)	0.05 / 1.0	Gut bei Multikollinearität, stabil	Keine automatische Variablenselektion
LASSO-Regression	L1 (α = 1)	0.01	Selektiert relevante Variablen automatisch	Kann wichtige Variablen entfernen

###Übung 2: Specific Tasks for LAB heartdisease LogReg 4.R

```
# (Angenommen, dass der gesamte vorherige Code aus LAB heartdisease LogReg 4.R
# bis einschließlich der Definitionen von x_train, y_train, x_test, heartdisease_test_scaled$ch
d,
# m_Largest_glm, ridgeModel und eval_metrics-Funktion bereits ausgeführt wurde)

# Für Reproduzierbarkeit der Aufgaben b, c, d
set.seed(456)

# --- Aufgabe a) Alle Leistungskennzahlen in einer Tabelle zusammenfassen ---
# Bisher betrachtete Modelle:
# 1. Standard Logistische Regression (m_Largest_glm)
# 2. Ridge Regression (lambda = 0.05, aus dem ursprünglichen Skript)

# Metriken für GLM (bereits berechnet als eval_glm)
# eval_glm enthält: accuracy, sensitivity, specificity, auc

# Metriken für Ridge (lambda = 0.05, fest) (bereits berechnet als eval_ridge_fixed)
# eval_ridge_fixed enthält: accuracy, sensitivity, specificity, auc

# --- Aufgabe b) Ridge Regression mit einem anderen (frei gewählten) Lambda trainieren ---
lambda_ridge_b <- 0.1 # Frei gewähltes Lambda, verschieden von 0.05

# Vorhersagen
probs_ridge_b <- predict(ridgeModel, newx = x_test, s = lambda_ridge_b, type="response")
if(is.matrix(probs_ridge_b)) probs_ridge_b <- probs_ridge_b[,1] # Sicherstellen, dass es ein Vek
tor ist

# Bewertung
eval_ridge_b <- eval_metrics(heartdisease_test_scaled$chd, probs_ridge_b)

# --- Aufgabe c) Ridge Regression mit Lambda aus Kreuzvalidierung trainieren ---
# cv.glmnet auf Trainingsdaten anwenden
cv_ridge_model <- cv.glmnet(x_train, y_train, alpha=0, family="binomial", standardize=FALSE, typ
e.measure="auc", nfolds=10)
# Zum Visualisieren der CV-Ergebnisse
plot(cv_ridge_model)
```




```
# Optimales Lambda (lambda.min ergibt bestes AUC, lambda.1se ist stärker regularisiert, aber inn
erhalb 1 SE)
lambda_cv_min <- cv_ridge_model$lambda.min
# lambda_cv_1se <- cv_ridge_model$lambda.1se # Alternative

print(paste("Optimales Lambda (lambda.min) aus CV:", lambda_cv_min))
```

```
## [1] "Optimales Lambda (lambda.min) aus CV: 0.0193176696170272"
```

```
# Vorhersagen mit lambda.min
probs_ridge_cv <- predict(cv_ridge_model, newx = x_test, s = "lambda.min", type="response")
if(is.matrix(probs_ridge_cv)) probs_ridge_cv <- probs_ridge_cv[,1] # Sicherstellen, dass es ein
Vektor ist

# Bewertung
eval_ridge_cv <- eval_metrics(heartdisease_test_scaled$chd, probs_ridge_cv)

# Tabelle für Aufgabe (a) zusammenstellen, einschließlich Modelle aus (b) und (c)

performance_data <- data.frame(
  Modell = character(),
  Lambda = character(),
  Genauigkeit = numeric(),
  Sensitivität = numeric(),
  Spezifität = numeric(),
  AUC = numeric(),
  stringsAsFactors = FALSE
)

# GLM
performance_data <- rbind(performance_data, data.frame(
  Modell = "Standard GLM",
  Lambda = "N/A",
  Genauigkeit = eval_glm$accuracy,
  Sensitivität = eval_glm$sensitivity,
  Spezifität = eval_glm$specificity,
  AUC = eval_glm$auc
))

# Ridge (lambda = 0.05 - original)
performance_data <- rbind(performance_data, data.frame(
  Modell = "Ridge",
  Lambda = "0.05 (Fest)",
  Genauigkeit = eval_ridge_fixed$accuracy,
  Sensitivität = eval_ridge_fixed$sensitivity,
  Spezifität = eval_ridge_fixed$specificity,
  AUC = eval_ridge_fixed$auc
))

# Ridge (lambda = 0.1 - Aufgabe b)
performance_data <- rbind(performance_data, data.frame(
  Modell = "Ridge",
  Lambda = paste(round(lambda_ridge_b,4), "(Gewählt)"),
  Genauigkeit = eval_ridge_b$accuracy,
  Sensitivität = eval_ridge_b$sensitivity,
  Spezifität = eval_ridge_b$specificity,
  AUC = eval_ridge_b$auc
))

# Ridge (lambda aus CV - Aufgabe c)
performance_data <- rbind(performance_data, data.frame(
```

```
  Modell = "Ridge",
  Lambda = paste(round(lambda_cv_min,4), "(CV min)"),
  Genauigkeit = eval_ridge_cv$accuracy,
  Sensitivität = eval_ridge_cv$sensitivity,
  Spezifität = eval_ridge_cv$specificity,
  AUC = eval_ridge_cv$auc
))

# ... (Vorheriger Code zum Erstellen von performance_data) ...

print("--- Performance-Tabelle ---")
```

```
## [1] "--- Performance-Tabelle ---"
```

```
# Für eine bessere Darstellung der Tabelle in RMarkdown kann kable verwendet werden
if (require(knitr) && require(kableExtra)) {
  knitr::kable(performance_data, caption = "Leistungsmetriken der Modelle") %>%
    kableExtra::kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsiv
e"))
} else {
  print(performance_data) # Fallback auf einfache Ausgabe
}
```

Leistungsmetriken der Modelle

Modell	Lambda	Genauigkeit	Sensitivität	Spezifität	AUC
Standard GLM	N/A	0.6666667	0.4897959	0.7640449	0.6941069
Ridge	0.05 (Fest)	0.6594203	0.4081633	0.7977528	0.6803485
Ridge	0.1 (Gewählt)	0.6666667	0.4081633	0.8089888	0.6750745
Ridge	0.0193 (CV min)	0.6666667	0.4693878	0.7752809	0.6888328

```
# --- Aufgabe d) Zwei einflussreichste Prädiktoren im Modell mit höchstem AUC identifizieren ---
# Modell mit höchstem AUC aus der Tabelle finden
best_auc_model_row <- performance_data[which.max(performance_data$AUC), ]
print("--- Modell mit höchstem AUC ---")
```

```
## [1] "--- Modell mit höchstem AUC ---"
```

```
print(best_auc_model_row)
```

```
##         Modell Lambda Genauigkeit Sensitivität Spezifität       AUC
## 1 Standard GLM    N/A   0.6666667    0.4897959  0.7640449 0.6941069
```

```
# Koeffizienten des besten Modells abrufen
best_model_name <- best_auc_model_row$Modell
best_model_lambda_str <- best_auc_model_row$Lambda

cat("\n--- Identifikation der einflussreichsten Prädiktoren für das Modell mit dem höchsten AUC
---\n")
```

```
##
## --- Identifikation der einflussreichsten Prädiktoren für das Modell mit dem höchsten AUC ---
```

```
cat("Bestes Modell:", best_model_name, "\n")
```

```
## Bestes Modell: Standard GLM
```

```
cat("Lambda-Information:", best_model_lambda_str, "\n")
```

```
## Lambda-Information: N/A
```

```
coefs_best_model <- NULL
if (best_model_name == "Standard GLM") {
    coefs_best_model <- coef(m_Largest_glm)
} else if (best_model_name == "Ridge") {
    if (grepl("CV min", best_model_lambda_str)) {
        coefs_best_model <- coef(cv_ridge_model, s = "lambda.min")
    } else if (grepl("0.05", best_model_lambda_str)) {
        coefs_best_model <- coef(ridgeModel, s = 0.05)
    } else if (grepl(as.character(round(lambda_ridge_b,4)), best_model_lambda_str)) {
         coefs_best_model <- coef(ridgeModel, s = lambda_ridge_b)
    } else {
        stop("Konnte Koeffizienten des besten Ridge-Modells anhand der Lambda-Angabe nicht ident
ifizieren.")
    }
}

if (!is.null(coefs_best_model)) {
    print("Koeffizienten des besten Modells:")
    print(coefs_best_model)

    # Intercept ausschließen und als benannten Vektor extrahieren
    predictor_coefs <- as.matrix(coefs_best_model)[-1, 1]

    # Beträge für die Einflussbewertung nehmen
    abs_coefs <- abs(predictor_coefs)

    # Nach absteigendem Betrag sortieren
    sorted_coefs <- sort(abs_coefs, decreasing = TRUE)

    print("Prädiktoren sortiert nach absoluter Koeffizientenstärke (Einfluss):")
    print(sorted_coefs)

    # Zwei einflussreichste Prädiktoren identifizieren
    most_influential_predictors <- names(sorted_coefs)[1:2]

    cat("\nDie zwei einflussreichsten Prädiktoren sind:\n")
    cat("1.", most_influential_predictors[1], "mit Koeffizient:", predictor_coefs[most_influenti
al_predictors[1]], "\n")
    cat("2.", most_influential_predictors[2], "mit Koeffizient:", predictor_coefs[most_influenti
al_predictors[2]], "\n")

    # Begründung und Interpretation
    cat("\nBegründung:\n")
    cat("Diese Prädiktoren wurden gewählt, da ihre Koeffizienten im besten Modell (basierend auf
AUC) den größten absoluten Wert haben. Da die numerischen Prädiktoren standardisiert wurden, sin
d ihre Koeffizienten direkt vergleichbar und zeigen die relative Bedeutung für die Vorhersage de
r CHD-Wahrscheinlichkeit.\n")

    cat("\nInterpretation der Beziehung zum Ziel (chd):\n")

    # Prädiktor 1
    predictor1_name <- most_influential_predictors[1]
```

```
    predictor1_coef_val <- predictor_coefs[predictor1_name]
    relationship1 <- ifelse(predictor1_coef_val > 0, "erhöht", "verringert")
    cat(" - Für", predictor1_name, "(Koeffizient =", round(predictor1_coef_val, 3), "):\n")
    if (grepl("famhist", predictor1_name)) {
        cat("   Eine positive Familienanamnese (famhistPresent)", relationship1, "erheblich die
Log-Odds (und damit die Wahrscheinlichkeit) für CHD im Vergleich zu keiner Familienanamnese, bei
konstant gehaltenen anderen Faktoren.\n")
    } else {
        cat("   Eine Erhöhung um eine Standardabweichung in", predictor1_name, relationship1, "d
ie Log-Odds für CHD um ca.", round(predictor1_coef_val, 3), ", bei konstanten anderen Faktoren.
Das bedeutet: Höhere Werte von", predictor1_name, "sind mit einer", ifelse(relationship1=="erhöh
t","höheren","niedrigeren"), "Wahrscheinlichkeit für CHD verbunden.\n")
    }

    # Prädiktor 2
    predictor2_name <- most_influential_predictors[2]
    predictor2_coef_val <- predictor_coefs[predictor2_name]
    relationship2 <- ifelse(predictor2_coef_val > 0, "erhöht", "verringert")
    cat(" - Für", predictor2_name, "(Koeffizient =", round(predictor2_coef_val, 3), "):\n")
     if (grepl("famhist", predictor2_name)) {
        cat("   Eine positive Familienanamnese (famhistPresent)", relationship2, "erheblich die
Log-Odds (und damit die Wahrscheinlichkeit) für CHD im Vergleich zu keiner Familienanamnese, bei
konstant gehaltenen anderen Faktoren.\n")
    } else {
        cat("   Eine Erhöhung um eine Standardabweichung in", predictor2_name, relationship2, "d
ie Log-Odds für CHD um ca.", round(predictor2_coef_val, 3), ", bei konstanten anderen Faktoren.
Das bedeutet: Höhere Werte von", predictor2_name, "sind mit einer", ifelse(relationship2=="erhöh
t","höheren","niedrigeren"), "Wahrscheinlichkeit für CHD verbunden.\n")
    }
} else {
    cat("Konnte die Koeffizienten des besten Modells nicht abrufen, um die einflussreichsten Prä
diktoren zu bestimmen.\n")
}
```

```
## [1] "Koeffizienten des besten Modells:"
##    (Intercept)            sbp        tobacco            ldl famhistPresent
##    -1.09708264     0.08754520     0.59637555     0.66193374     0.58218306
##        obesity        alcohol            age
##    -0.01712999     0.16158239     0.42058853
## [1] "Prädiktoren sortiert nach absoluter Koeffizientenstärke (Einfluss):"
##            ldl        tobacco famhistPresent            age        alcohol
##     0.66193374     0.59637555     0.58218306     0.42058853     0.16158239
##            sbp        obesity
##     0.08754520     0.01712999
##
## Die zwei einflussreichsten Prädiktoren sind:
## 1. ldl mit Koeffizient: 0.6619337
## 2. tobacco mit Koeffizient: 0.5963756
##
## Begründung:
## Diese Prädiktoren wurden gewählt, da ihre Koeffizienten im besten Modell (basierend auf AUC)
den größten absoluten Wert haben. Da die numerischen Prädiktoren standardisiert wurden, sind ihr
e Koeffizienten direkt vergleichbar und zeigen die relative Bedeutung für die Vorhersage der CHD
-Wahrscheinlichkeit.
##
## Interpretation der Beziehung zum Ziel (chd):
##  - Für ldl (Koeffizient = 0.662 ):
##    Eine Erhöhung um eine Standardabweichung in ldl erhöht die Log-Odds für CHD um ca. 0.662 ,
bei konstanten anderen Faktoren. Das bedeutet: Höhere Werte von ldl sind mit einer höheren Wahrs
cheinlichkeit für CHD verbunden.
##  - Für tobacco (Koeffizient = 0.596 ):
##    Eine Erhöhung um eine Standardabweichung in tobacco erhöht die Log-Odds für CHD um ca. 0.5
96 , bei konstanten anderen Faktoren. Das bedeutet: Höhere Werte von tobacco sind mit einer höhe
ren Wahrscheinlichkeit für CHD verbunden.
```