(11) Lineare Diskriminanzanalyse

Ziel dieser Sitzung ist es, die lineare Diskriminanzanalyse eines simulierten Beispieldatensatzes auf der Implementationsebene nachzuvollziehen und die resultierenden Ergebnisse im Vergleich mit dem zugrundeliegenden Modell darzustellen.

Datensatz generieren

Untenstehender **R**-Code erzeugt dazu zunächst einen Datensatz aus zweidimensionalen Zufallsvektoren ($m = 2$), für den jeder Datenpunkt ($n = 60$) zu einer von zwei Klassen gehört ($y = 0$ vs. $y = 1$) und in dem sich die beiden Klassen hinsichtlich ihres multivariaten Erwartungswerts unterscheiden (μ_0 vs. μ_1), aber nicht hinsichtlich ihrer Kovarianzmatrix (Σ).

```
# Modellformulierung
library(mvtnorm)                                  # multivariate Normalverteilung
set.seed(0)                                       # Zufallszahlengenerator
m      = 2                                        # Featurevektordimension
n      = 60                                       # Anzahl Trainingsdatenpunkte
mu     = 0.5                                      # w.a.u. Bernoulliparameter \mu
mu_0   = c(1,1)+0.2                               # w.a.u. Normalverteilungsparameter \mu_0
mu_1   = c(2,2)-0.2                               # w.a.u. Normalverteilungsparameter \mu_1
Sigma  = matrix(c( 0.50, -0.15,                   # Kovarianzmatrixparameter \Sigma
                  -0.15,  0.50),
                nrow  = m,
                byrow = TRUE)

# Modellsampling
y       = matrix(rep(NaN,n)  , nrow = 1)          # Labeldatenarray
x       = matrix(rep(NaN,n*m), nrow = m)          # Featurevektorarray
for(i in 1:n){                                    # Iteration über Datenpunkte
    y[i]  = rbinom(1,1,mu)                        # y_i \sim Bern(\mu)
    x[,i] = ((rmvnorm(1, mu_0, Sigma)**(1-y[i]))# x_i \sim N(\mu_0,\Sigma)^{1-y} N(\mu_1,\Sigma)^y
            *(rmvnorm(1, mu_1, Sigma)**(y[i])))
}

# Datensatzspeicherung
D           = data.frame(t(rbind(x,y)))
colnames(D) = c("BDI", "GLU", "RES")
write.csv(D, "Lineare_Diskriminanzanalyse.csv", row.names = F)
```

Kreuzvalidierte Modellschätzung

Folgender **R**-Code führt dann mithilfe der in der Vorlesung hergeleiteten Formeln eine kreuzvalidierte Schätzung der Parameter der linearen Diskriminanzanalyse durch, wobei das Schema der *leave-one-out*-Kreuzvalidierung angewendet wird.

```
D       = read.csv("Lineare_Diskriminanzanalyse.csv")                    # Datensatz einlesen
K       = nrow(D)                                                        # Anzahl Kreuzvalidierungsschritte
p_y     = matrix(rep(NaN, K)  , nrow = 1)                                # p(y = 1|x)
y_pred  = matrix(rep(NaN, K*2), nrow = K)                                # Prädiktionsarray
for(k in 1:K){                                                           # K-fold LOOCV
    x_train     = t(D[-k,1:2])                                           # Trainingsdatensatzfeatures
    y_train     = t(D[-k,3  ])                                           # Trainingsdatensatzlabels
    x_test      = t(D[ k,1:2])                                           # Testdatensatzfeaturevektor
    y_pred[k,1] = t(D[ k,3])                                             # Testdatensatzlabel (wahr)
    n           = ncol(x_train)                                          # n
    m           = nrow(x_train)                                          # m
    mu_hat      = mean(y_train)                                          # \hat{\mu}
    mu_0_hat    = rowMeans(x_train[,y_train == 0])                       # \hat{\mu}_0
    mu_1_hat    = rowMeans(x_train[,y_train == 1])                       # \hat{\mu}_1
    Sigma_hat   = matrix(rep(0,m^2), nrow = m)                           # \hat{\Sigma}
    for(i in 1:n){
        Sigma_hat= (Sigma_hat + (1/n)*
                   ((y_train[i] == 0)*(x_train[,i]-mu_0_hat) %*% t((x_train[,i]-mu_0_hat))
                   +(y_train[i] == 1)*(x_train[,i]-mu_1_hat) %*% t((x_train[,i]-mu_1_hat))))
    }
    beta_hat    = matrix(c((1/2)*( t(mu_0_hat) %*% solve(Sigma_hat) %*% mu_0_hat
                                 - t(mu_1_hat) %*% solve(Sigma_hat) %*% mu_1_hat)
                                 + log(mu_hat/(1-mu_hat)),
                           -solve(Sigma_hat) %*% (mu_0_hat-mu_1_hat)), nrow = m+1)
    x_test_tilde = rbind(1, x_test)                                      # \tilde{x}
    p_y[k]      = 1/(1+exp(-t(x_test_tilde) %*% beta_hat))               # p(y = 1|x)
    y_pred[k,2] = as.numeric(p_y[k] >= 0.5)                              # Testdatensatzlabel (prädiziert)
}
```

Auf Grundlage der in jedem Kreuzvalidierungsschritt prädizierten Labels lassen sich Prädiktionsperformancemaße für die LDA-basierte Klassifikation bestimmen und ausgeben, wie aus folgendem **R**-Code hervorgeht.

```
rp      = sum(y_pred[y_pred[,1] == 1, 2] == 1)                           # |(1,1)|
rn      = sum(y_pred[y_pred[,1] == 0, 2] == 0)                           # |(0,0)|
fp      = sum(y_pred[y_pred[,1] == 0, 2] == 1)                           # |(0,1)|
fn      = sum(y_pred[y_pred[,1] == 1, 2] == 0)                           # |(1,0)|
ACC     = (rp+rn)/(rp+fp+rn+fn)                                          # Accuracy
SEN     = rp/(rp+fn)                                                     # Sensitivity
SPE     = rn/(rn+fp)                                                     # Specificity
cat(  "Accuracy   : ", ACC,                                              # Ergebnisausgabe
    "\nSensitivity: ", SEN,
    "\nSpecificity: ", SPE, "\n")
```

```
Accuracy   :  0.8666667
Sensitivity:  0.8823529
Specificity:  0.8461538
```

Ergebnisse visualisieren

Schließlich wollen wir die Ergebnisse der LDA-Schätzung visualisieren, indem wir die wahren und geschätzten Werte der Erwartungswerte und Kovarianzmatrizen beider Klassen gegenüberstellen. Dies wird durch folgenden **R**-Code erreicht, der Abbildung 1 erzeugt.

```
library(latex2exp)

# Abbildungsparameter
par(
    family      = "sans",
    mfcol       = c(1,1),
    pty         = "s",
    bty         = "l",
    lwd         = 1,
    las         = 1,
    mgp         = c(2,1,0),
    xaxs        = "i",
    yaxs        = "i",
    font.main   = 1,
    cex         = 1,
    cex.main    = 1)

# multivariate Isokonturen
library(ellipse)
iso_0           = ellipse(Sigma    , level = 0.5, centre = mu_0)
iso_1           = ellipse(Sigma    , level = 0.5, centre = mu_1)
iso_0_hat       = ellipse(Sigma_hat, level = 0.5, centre = mu_0_hat)
iso_1_hat       = ellipse(Sigma_hat, level = 0.5, centre = mu_1_hat)

# N(\mu_0,\Sigma)
plot(iso_0,
    type = "l",
    col  = "Red",
    xlim = c(0,3),
    ylim = c(0,3),
    xlab = TeX("$x_1$"),
    ylab = TeX("$x_2$"))
points(mu_0[1], mu_0[2],
    col  = "Red",
    pch  = 3,
    cex  = 1)

# N(\hat{\mu}_0,\hat{\Sigma})
lines(iso_0_hat[,1], iso_0_hat[,2],
    col  = "Red",
    lty  = 2)
points(mu_0_hat[1], mu_0_hat[2],
    col  = "Red",
    pch  = 2,
    cex  = 1)

# N(\mu_1,\Sigma)
lines(iso_1[,1], iso_1[,2],
    type = "l",
    col  = "Blue")
points(mu_1[1], mu_1[2],
    col  = "Blue",
    pch  = 3,
```

```
    cex  = 1)

# N(\hat{\mu}_1,\hat{\Sigma})
lines(iso_1_hat[,1], iso_1_hat[,2],
    col  = "Blue",
    lty  = 2)
points(mu_1_hat[1], mu_1_hat[2],
    col  = "Blue",
    pch  = 2,
    cex  = 1)

# Daten y_i = 0
points(D[D[,3] == 0,1], D[D[,3] == 0,2],
    col = "White",
    bg  = "Red",
    pch = 21)

# Daten y^{(i)} = 1
points(D[D[,3] == 1,1], D[D[,3] == 1,2],
    col = "White",
    bg  = "Blue",
    pch = 21)

# PDF-Speicherung
dev.copy2pdf(
    file        = "./Abbildungen/lda_lernen.pdf",
    width       = 5,
    height      = 5)
dev.off()
```

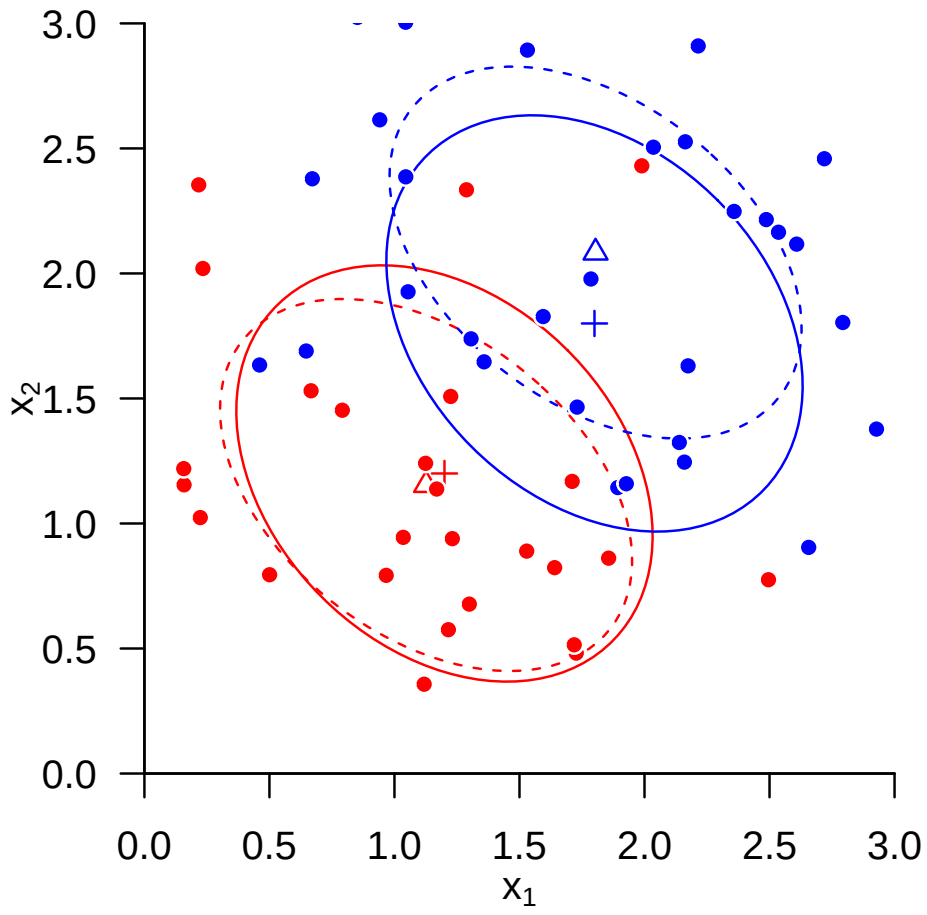

Abbildung 1. Zwei Klassen (rot, blau) mit wahren und geschätzten Erwartungswerten (Kreuze bzw. Dreiecke) sowie wahren und geschätzten Kovarianzmatrizen (durchgezogene bzw. gestrichelte Ellipsen).