(8) Prädiktion und Kreuzvalidierung

Ziel dieses Seminarblatts ist es, das Vorgehen typischer Analysen der prädiktiven Modellierung auf ihrer Implementierungsebene zu verdeutlichen. Dafür wollen wir das Prinzip der n-fachen Leave-one-out Crossvalidation mithilfe eines "Nearest-Neighbour-Klassifikationsansatzes" verdeutlichen: Zu zwei Klassen zweidimensionaler Trainingsfeaturevektoren bestimmen wir jeweils das Stichprobenmittel und die Stichprobenkovarianz und ordnen einen zweidimensionalen Testfeaturevektor dann derjenigen Klasse zu, dessen Stichprobenmittel es im Sinne der Mahalanobisdistanz in Bezug zur jeweiligen Stichprobenkovarianzmatrix näher liegt.

Datengeneration

Wir erstellen zunächst einen simulierten Datensatz mit $n = 40$ für dieses Vorgehen, den wir in Abbildung 1 zusammen mit den klassenpezifischen Stichprobenmitteln und Stichprobenkovarianzmatrizen visualisieren.

```r
# Modellparameter
library(mvtnorm)                                                    # R Paket für multivariate Normalverteilung
set.seed(0)                                                         # random number generator seed
m       = 2                                                         # Featurevektordimension
n       = 40                                                        # Anzahl Trainingsdatenpunkte
mu_0    = c(1,1)                                                    # w.a.u. Erwartungswertparameter von Klasse 0
mu_1    = c(2,2)                                                    # w.a.u. Erwartungswertparameter von Klasse 1
Sigma_0 = matrix(c( 0.5, -0.3,                                      # w.a.u. Kovarianzmatrixparameter von Klasse 0
                   -0.3,  0.5),
                 byrow = TRUE,
                  nrow = m)
Sigma_1 = matrix(c( 1.0,  0.5,                                      # w.a.u. Kovarianzmatrixparameter von Klasse 1
                    0.5,  1.0),
                 byrow = TRUE,
                  nrow = m)

# 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

  # Klassenabhängige Datengeneration
  if(i <= n/2){
    y[i]  = 0                                                       # Label
    x[,i] =  rmvnorm(1, mu_0, Sigma_0)                              # Featurevektor
    } else {
    y[i]  = 1                                                       # Label
    x[,i] =  rmvnorm(1, mu_1, Sigma_1)                              # Featurevektor
    }
}
D       = rbind(x,y)                                                # Datensatzkonkatenation
fname   = "./8_Daten_S/8_Prädiktion_und_Kreuzvalidierung.csv"       # Dateiname
write.csv(D, file = fname, row.names = FALSE)                       # Datenspeichern
```

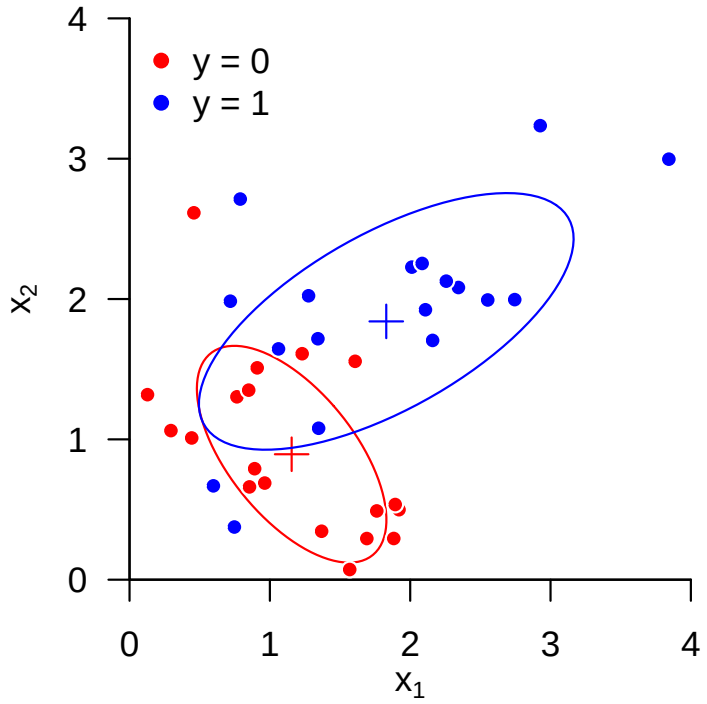

Abbildung 1. Beispieldatensatz

n-fache Leave-one-out Crossvalidation

Mit folgendem **R** Code führen wir im Sinne obigen Klassifizierungsansatzes eine n-fache Leave-one-out Crossvalidation durch.

```
# Datensatz
D       = read.csv("./8_Daten_S/8_Prädiktion_und_Kreuzvalidierung.csv")        # Datensatz
L       = c(0,1)                                                               # Klassenlabels
x       = as.matrix(D[1:2,])                                                   # Featurevektoren
y       = as.matrix(D[3,])                                                     # Label
n       = ncol(y)                                                              # Anzahl Datenpunkte
y_pred  = matrix(rep(NaN,n*2), nrow = n)                                       # Array wahrer und prädizierter Label

# n-fache Leave-One-Out Cross-Validation
for(i in 1:n){

    # Datensatzpartition
    x_train     = as.matrix(x[,-i])                                            # iter Featurevektor nicht im Trainingsdatendatz
    y_train     = as.matrix(y[,-i])                                            # ites Label nicht im Trainingsdatendatz
    x_test      = as.matrix(x[, i])                                            # iter Featurevektor als Testdatenpunkt
    y_test      = as.matrix(y[, i])                                            # ites Label als Testdatenpunkt
    y_pred[i,1] = y_test                                                       # wahres Label des iten Labels

    # Klassenspezfische Mahalanobisdistanz ("Training")
    D       = matrix(rep(NaN,2), ncol = 2)                                     # 1 x 2   Array für klassenspezifische Mahalanobisdistanzen
    for (l in L){                                                              # Iteration über Klassen
        Xl      = x[,y == l]                                                   # x^{(i)} für Label l
        nl       = ncol(Xl)                                                    # Anzahl Datenvektorealisierungen
        I_nl     = diag(nl)                                                    # Einheitsmatrix I_n
        J_nl     = matrix(rep(1,nl^2), nrow = nl)                              # 1_{nn}
        x_bar    = (1/nl)* Xl %*% J_nl[,1]                                     # Stichprobenmittel
        C        = (1/(nl-1))*(Xl %*% (I_nl-(1/nl)*J_nl) %*% t(Xl))            # Stichprobenkovarianzmatrix
        D[l+1]   = t(x_test - x_bar) %*% C %*% (x_test - x_bar)}               # Mahalanobisdistanz

    # Prädiktion ("Test")
    if (D[1] <= D[2]){y_pred[i,2] = 0} else {y_pred[i,2] = 1}                  # prädiziertes Label
}

# LOOCV Evaluation
rp      = sum(y_pred[y_pred[,1] == 1,2] == 1)                                  # Anzahl richtig positiver Prädiktionen (1,1)
rn      = sum(y_pred[y_pred[,1] == 0,2] == 0)                                  # Anzahl richtig negativer Prädiktionen (0,0)
fp      = sum(y_pred[y_pred[,1] == 0,2] == 1)                                  # Anzahl falsch  positiver Prädiktionen (0,1)
fn      = sum(y_pred[y_pred[,1] == 1,2] == 0)                                  # Anzahl falsch  positiver Prädiktionen (1,0)
ACC     = (rp+rn)/(rp+fp+rn+fn)                                                # Accuracy
SEN     = rp/(rp+fn)                                                           # Sensitivität
SPE     = rn/(rn+fp)                                                           # Spezifivität

# Ergebnisausgabe
cat("Accuracy    : "  , ACC,
    "\nSensitivity : " , SEN,
    "\nSpecificity : " , SPE)
```

```
Accuracy    :  0.65
Sensitivity :  0.35
Specificity :  0.95
```

Wir erhalten für den Beispieldatensatz eine Akkuratheit von 0.65, eine Sensitivität von 0.35 und eine Spezifizität von 0.95.