MRSA Klassifizierung in Schweinemastherden mit *glmnet*

Dr. Robert Opitz

2021-08-27

Der MRSA-Datensatz in Schweinemastherden wurde von Fromm *et al.* in PVM, **117**, 2014 veröffentlicht. Es handelt sich dabei um eine Metastudie. Diese wurde mittels logistischer Regression und generalized estiamtion equations ausgewertet. Es wurde eine Variablenselektion durch geführt: (i) durch univariates filtern (mit logistischer Regression) wurden alle Variablen entfernt, die keinen direkten Einfluss auf die Zielvariable haben, (ii) von paarweise korrelierten Variablen wurde eine der korrelierten Variablen entfernt. Es wurde versucht auch Interaktionsterme zubreücksichtigen. Es wurde nur einer gefunden, auch aufgrund der Anzahl der Datenpunkte Datensatzes. Dieser wurde im finalen Modell nicht weiter berücksichtigt.

# Notwendige Pakte und eigene Funktionen

Hier verwendete Pakete.

library(magrittr)  
library(caret)  
library(glmnet)  
library(pROC)  
library(foreach)  
library(doParallel)  
library(latex2exp)  
  
number\_of\_threads <- parallel::detectCores() # use max number of cores  
set.seed(42)  
  
# make a cluster of max. number of cores for foreach  
cl <- makeCluster(number\_of\_threads)  
registerDoParallel(cl)

Wir verwenden heir eine eigene Funktion zur Corss-Validierung von glmnet.

my\_glmnet\_cv <- function(formula,  
 data,  
 train\_index,  
 alpha\_seq = c(1), lambda\_seq = NULL,  
 type = 'ungrouped')   
{  
 # bestimme Response aus der Formel  
 response\_data <- data[[ all.vars(formula)[1] ]]  
   
 # lambda muss decreasing sein  
 lambda\_seq <- sort(lambda\_seq, decreasing = TRUE)  
  
 # length of train\_index, alpha and lambda sequence  
 number\_of\_train\_sets <- length(train\_index)  
 n <- length(alpha\_seq)  
 m <- length(lambda\_seq)  
  
 # umgekehrter index zur sortierung der AUC Werte  
 lambda\_index <- length(lambda\_seq):1  
   
 # get all the AUC for variations of alpha and lambda  
 # for all training and test data sets  
 result <- foreach(j = 1:number\_of\_train\_sets,   
 .packages = c('glmnet', 'pROC'),  
 .errorhandling = 'pass',  
 .verbose = FALSE) %dopar% {  
   
 # get the current index from the train index list  
 index <- train\_index[[j]]  
   
 # Datenaufteilung und Erstellung der Designmatrix  
 x\_train <- model.matrix(formula, data[ index, ])  
 x\_test <- model.matrix(formula, data[-index, ])  
  
 y\_train <- response\_data[ index]  
 y\_test <- response\_data[-index]  
   
 # intialize the grid matrix to collect all the computed AUC values  
 # for this particular training and test data set  
 auc\_test <- matrix(0.0, ncol = n, nrow = m)  
 acc\_test <- matrix(0.0, ncol = n, nrow = m)  
  
 # flag indicating, if glmnet failed, and the results for this   
 # training set will be discarded  
 error <- FALSE  
   
 # build the model for the current train data,  
 # with the current values of alpha and the used lambda sequence  
 for (i in seq\_len(n) )   
 {  
 # build the model  
 trained\_model <- glmnet(x = x\_train,  
 y = y\_train,  
 family = c("binomial"),  
 standardize = FALSE, # not necessary  
 intercept = FALSE, # we use a design matrix  
 alpha = alpha\_seq[i],  
 lambda = lambda\_seq,  
 maxit = 10^6)  
   
 if (trained\_model$jerr != 0)   
 {  
 error <- TRUE  
 break  
 }  
   
 # store AUC values for test data  
 auc\_test[,i] <- apply(predict(trained\_model,   
 newx = x\_test,   
 type = 'response'),   
 2,   
 function(x, y\_data)   
 {  
 tryCatch( auc( roc( y\_data, x ) ),  
 error = function(e) 0.5 )  
 },   
 y\_test )  
   
 acc\_test[,i] <- apply(predict(trained\_model,  
 newx = x\_test,  
 type = 'class'),  
 2,  
 function(pred\_class, y\_class)   
 {  
 conf\_table <- table(y\_class, pred\_class)  
 sum(diag(conf\_table)) / sum(conf\_table)  
 },  
 y\_test)  
   
 }  
 # output object for foreach  
 # the rows get re-orderd, as glmnet needs decreasing lambda values  
 if (error)  
 NULL  
 else  
 list(auc = auc\_test[lambda\_index,],  
 acc = acc\_test[lambda\_index,])  
 }  
   
 # clean result list of foreach from unsuccsessful trained models   
 # (they are NULL)  
 result <- Filter(Negate(is.null), result)  
 number\_of\_succesful\_models <- length(result)  
   
 if (number\_of\_succesful\_models != number\_of\_train\_sets)  
 warning(paste('Only', number\_of\_succesful\_models,   
 'successful runs out of', number\_of\_train\_sets, "\n"))  
   
 auc\_test\_list <- lapply(result, function(x) x$auc)  
 acc\_test\_list <- lapply(result, function(x) x$acc)  
   
 # average of Test AUC for all successful trained models  
 ave\_auc\_test <- Reduce('+', auc\_test\_list) / number\_of\_succesful\_models  
 ave\_acc\_test <- Reduce('+', acc\_test\_list) / number\_of\_succesful\_models  
   
 list(ave\_auc\_test = ave\_auc\_test,  
 ave\_acc\_test = ave\_acc\_test)  
}

Hilfsfunktion fürs Plotten.

plot\_func <- function(model\_fit\_response, reference, title = "")   
{  
 lev <- levels(reference)  
 threshold <- seq(0, 1, 0.01)  
 n <- length(threshold)  
 acc <- rep(NA, n)  
 sen <- rep(NA, n)  
 spe <- rep(NA, n)  
 bacc <- rep(NA, n)  
 for (i in seq(n))  
 {  
 pred\_class <- factor(ifelse(model\_fit\_response <= threshold[i],   
 lev[1], lev[2]),  
 levels = lev)  
 confMat <- caret::confusionMatrix(data = pred\_class,  
 reference = reference)  
   
 acc[i] <- confMat$overall[["Accuracy"]]  
 sen[i] <- confMat$byClass[["Sensitivity"]]  
 spe[i] <- confMat$byClass[["Specificity"]]  
 bacc[i] <- confMat$byClass[["Balanced Accuracy"]]  
 }  
  
 plot(threshold, acc,   
 type = "l",  
 xlim = c(0,1),  
 ylim = c(0,1),  
 ylab = "Acc/Sen/Spe/bal. Acc",  
 main = title)  
 lines(threshold, sen, col = "red")  
 lines(threshold, spe, col = "blue")  
 lines(threshold, bacc, col = "green")  
 abline(v = 0.5, lty = 4)  
 cat("Max. Acc = ", max(acc),   
 " at Threshold = ", threshold[which.max(acc)], "\n")  
 abline(v = thres <- threshold[which.max(bacc)], lty = 2)  
 cat("Max. bal. Acc = ", max(bacc), " at Threshold = ", thres, "\n")  
 # no information criteria  
 tmp <- summary(reference)  
 no\_info <- max(tmp) / sum(tmp)  
 abline(h = no\_info, lty = 3)  
  
 plot(1 - spe, sen,   
 type = "l",   
 main = paste("ROC", title),   
 lwd = 5)  
 abline(a = 0, b = 1, lty = 2)  
 abline(a = 1, b = -1, lty = 3)  
  
 print(pROC::roc(reference, model\_fit\_response))  
  
 plot(1 - spe, sen + spe - 1,  
 ylim = c(0,1),  
 type = "l",  
 ylab = "J = sen + spe - 1",  
 main = paste("Youden's J", title),   
 lwd = 5)  
 abline(a = 1, b = -1, lty = 2)  
 max\_youden\_j <- which.max(sen + spe - 1)  
 cat("Sen = ", sen[max\_youden\_j],   
 "; Spe = ", spe[max\_youden\_j],   
 "; threshold = ", threshold[max\_youden\_j], "\n")  
 abline(v = 1 - sen[max\_youden\_j], lty = 3)  
   
 return(thres)  
}  
  
plot\_weights\_of\_feature <- function(weights\_list, feature\_name)  
{  
 weights <- sapply(weights\_list,   
 function(x, feature\_name) x[feature\_name,],   
 feature\_name)  
 hist(weights,  
 main = paste("Weight distribution for", feature\_name),  
 #breaks = 100,  
 probability = TRUE)  
 invisible(NULL)  
}

Andere Hilfsfunktion

# create normal useable interface to glmnet  
my\_glmnet <- function(formula, data, alpha, lambda, family = "binomial")  
{  
 # use formual to create the model matrix  
 y\_data <- data[[ all.vars(formula)[1] ]]  
 x\_data <- model.matrix(formula, data)  
 # build the model  
 result <- glmnet(x = x\_data,  
 y = y\_data,  
 family = family,  
 standardize = FALSE, # not necessary  
 intercept = FALSE, # we use a design matrix  
 alpha = alpha,  
 lambda = lambda,  
 maxit = 10^6)  
 result  
}  
  
# get all trained models for a certain alpha and lambda as alist  
get\_trained\_models <- function(my\_formula, df\_data, alpha, lambda,   
 train\_index)  
{  
 lapply(train\_index,  
 function(this\_index)  
 {  
 df\_train <- df\_data[this\_index,]  
 my\_glmnet(my\_formula,  
 data = df\_train,  
 alpha = alpha,  
 lambda = lambda)  
 })  
}

# Datenvorbereitung

## Einlesen und Vorbereiten der Daten.

Einlesen und Bereinigung der Daten. Es wird sichergestellt, dass alle Daten einen R-konformen Namen haben und als Faktoren vorliegen. die Variablennamen erhalten die Namen aus dem Paper. Einige der Kategorien der Variablen werden besser lesbar gestaltet.

MRSA\_schweineherden <- read.csv(file = file.choose(),  
 header = TRUE )  
  
# all categorigal variables needs to be checked if there name is valid in R  
# E. g.: TRUE and FALSE are not; TRUE. and FALSE. are  
# All categorical Variables need to be (at least) described as factors  
for (i in seq\_len(ncol(MRSA\_schweineherden))) {  
 MRSA\_schweineherden[[i]] <- make.names(MRSA\_schweineherden[[i]])  
 MRSA\_schweineherden[[i]] <- factor(MRSA\_schweineherden[[i]])  
}  
  
# the old names in the data set are renamed, so they match the names in the paper  
new\_col\_names <- list(HerdMRSA = 'HERD\_MRSA',  
 HerdTypeNum = 'HERD\_TYPE',  
 FeedingPlacesMinGrouped = 'HERD\_SIZE',  
 PurchaseBin = 'PURCHASE',  
 AntibioticsFrom10W = 'AM\_DRUG',  
 AllInAllOut = 'ALL\_IN\_or\_OUT',  
 Cleanup = 'CLEAN\_UP',  
 Disinfection = 'DISINFECTION',  
 SlattedFloor = 'SLATTED',  
 OrganicFarm = 'ORGANIC',  
 OtherLivestockAtFarmBin = 'OTHER\_LIVESTOCK',  
 CompanionAnimalsBin = 'COMPANION',  
 IndoorHousing = 'INDOOR')  
  
# rename the column names  
column\_names <- colnames(MRSA\_schweineherden)  
for (col\_name in column\_names)  
 colnames(MRSA\_schweineherden)[column\_names == col\_name] <- new\_col\_names[[col\_name]]  
  
# replace old variable types for variable HERD\_SIZE  
levels(MRSA\_schweineherden$HERD\_SIZE)

## [1] "X..5000.pigs" "X0.499.pigs" "X1000.4999.pigs" "X500.999.pigs"

levels(MRSA\_schweineherden$HERD\_SIZE) <- c('huge', 'small', 'large', 'medium')  
  
# reorder factors  
MRSA\_schweineherden$HERD\_SIZE <- factor(MRSA\_schweineherden$HERD\_SIZE,   
 levels = c('small', 'medium',   
 'large', 'huge'),  
 ordered = TRUE)  
levels(MRSA\_schweineherden$HERD\_SIZE)

## [1] "small" "medium" "large" "huge"

levels(MRSA\_schweineherden$HERD\_TYPE)

## [1] "farrow.to.finisher" "grower.to.finisher" "weaner.to.finisher"

levels(MRSA\_schweineherden$HERD\_TYPE) <- c("farrow", "grower", "weaner")  
levels(MRSA\_schweineherden$HERD\_TYPE)

## [1] "farrow" "grower" "weaner"

# replace old variable types for variable SLATTED\_AT\_LEAST\_PARTIALLY  
levels(MRSA\_schweineherden$SLATTED)

## [1] "no..not.slatted." "yes..at.least.partially.slatted."

levels(MRSA\_schweineherden$SLATTED) <- c('no', 'yes')  
levels(MRSA\_schweineherden$SLATTED)

## [1] "no" "yes"

## Vorstellung des Datensatzes (EDA)

str(MRSA\_schweineherden)

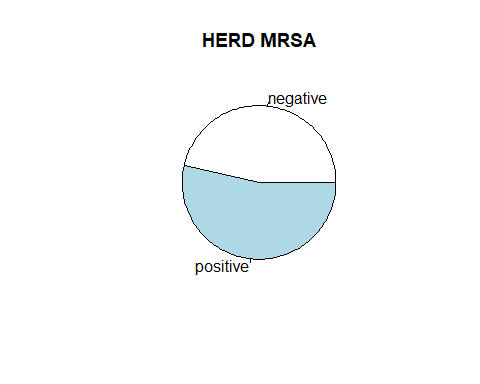
## 'data.frame': 400 obs. of 13 variables:  
## $ HERD\_MRSA : Factor w/ 2 levels "negative","positive": 2 2 1 2 1 2 2 2 2 2 ...  
## $ HERD\_TYPE : Factor w/ 3 levels "farrow","grower",..: 2 2 2 1 1 1 2 2 2 2 ...  
## $ HERD\_SIZE : Ord.factor w/ 4 levels "small"<"medium"<..: 3 3 2 1 2 2 2 2 3 3 ...  
## $ AM\_DRUG : Factor w/ 2 levels "no","yes": 2 2 1 1 1 1 2 1 2 1 ...  
## $ PURCHASE : Factor w/ 2 levels "no","yes": 2 2 2 1 1 1 2 2 2 2 ...  
## $ INDOOR : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...  
## $ OTHER\_LIVESTOCK: Factor w/ 2 levels "no","yes": 1 2 1 1 1 1 2 2 1 1 ...  
## $ COMPANION : Factor w/ 2 levels "no","yes": 1 1 1 2 2 2 1 2 1 2 ...  
## $ ALL\_IN\_or\_OUT : Factor w/ 2 levels "no","yes": 2 2 1 1 1 2 2 1 2 2 ...  
## $ CLEAN\_UP : Factor w/ 2 levels "no","yes": 2 2 2 1 2 2 2 2 2 2 ...  
## $ DISINFECTION : Factor w/ 2 levels "no","yes": 2 2 2 1 2 2 2 2 2 2 ...  
## $ SLATTED : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...  
## $ ORGANIC : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...

Bestimmung der Anzahl der Datenpunkte, Anzahl Variablen und Verteilung der Response.

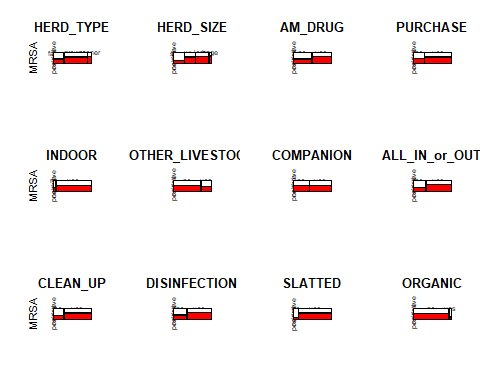
MRSA\_schweineherden$HERD\_MRSA %>% length -> number\_of\_datapoints  
MRSA\_schweineherden[1,-1] %>% length -> number\_of\_variables  
MRSA\_schweineherden$HERD\_MRSA %>% summary

## negative positive   
## 186 214

MRSA\_schweineherden$HERD\_MRSA %>% summary %>% pie(.,main = 'HERD MRSA')



Datenpunkte=400, Variablen=12.



nearZeroVar(MRSA\_schweineherden, saveMetrics = TRUE)

## freqRatio percentUnique zeroVar nzv  
## HERD\_MRSA 1.150538 0.50 FALSE FALSE  
## HERD\_TYPE 2.252252 0.75 FALSE FALSE  
## HERD\_SIZE 1.247863 1.00 FALSE FALSE  
## AM\_DRUG 1.105263 0.50 FALSE FALSE  
## PURCHASE 2.603604 0.50 FALSE FALSE  
## INDOOR 20.052632 0.50 FALSE TRUE  
## OTHER\_LIVESTOCK 2.738318 0.50 FALSE FALSE  
## COMPANION 1.409639 0.50 FALSE FALSE  
## ALL\_IN\_or\_OUT 2.149606 0.50 FALSE FALSE  
## CLEAN\_UP 2.846154 0.50 FALSE FALSE  
## DISINFECTION 1.941176 0.50 FALSE FALSE  
## SLATTED 7.695652 0.50 FALSE FALSE  
## ORGANIC 16.391304 0.50 FALSE FALSE

Die Variable INDOOR besteht fast ausschließlich aus Herden, die indoor leben, fast keine die outdoor sind. Dadurch haben wir eine near-zero-variance (nzv) für diese Variable. Wir entfernen diese Variable daher.

MRSA\_schweineherden\_reduced <- MRSA\_schweineherden  
MRSA\_schweineherden\_reduced$INDOOR <- NULL

# Set the Training Index

train\_index <- createMultiFolds(MRSA\_schweineherden\_reduced$HERD\_MRSA,  
 k = 10,  
 times = 100 )

# Hyperparamteroptimierung

## Einfacher Ansatz: Nur alle Hauptterme

### Auswertung mit repeated Cross-validation.

# lambda Sequenz für die Formel Herd\_MRSA ~ .  
alpha\_seq <- seq(0, 1, 0.01)  
lambda\_seq <- seq(0.00001, 0.5, 0.01)  
# Die Formel berücksichtigt alle Terme, aber nur als Hauptterme  
start\_time <- Sys.time()  
result\_cv\_lin <- my\_glmnet\_cv(formula = HERD\_MRSA ~ .,   
 data = MRSA\_schweineherden\_reduced,   
 train\_index,   
 alpha\_seq = alpha\_seq,  
 lambda\_seq = lambda\_seq)  
print(Sys.time() - start\_time)

## Time difference of 19.85126 mins

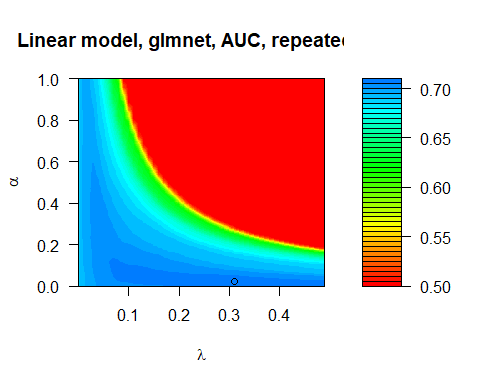
best\_auc\_lin <- max(result\_cv\_lin$ave\_auc\_test )  
best\_index <- which(result\_cv\_lin$ave\_auc\_test == best\_auc\_lin,   
 arr.ind = TRUE )  
  
#best\_auc\_lin\_se <- result\_lin$se\_auc\_test[best\_index]  
  
print(best\_index)

## row col  
## [1,] 32 3

auc\_alpha\_lin <- alpha\_seq[ best\_index[2] ]  
auc\_lambda\_lin <- lambda\_seq[ best\_index[1] ]  
  
cat('The best Test AUC=', round(best\_auc\_lin, 3),   
 ' for alpha=', auc\_alpha\_lin, 'and lambda=', auc\_lambda\_lin, '.\n')

## The best Test AUC= 0.708 for alpha= 0.02 and lambda= 0.31001 .

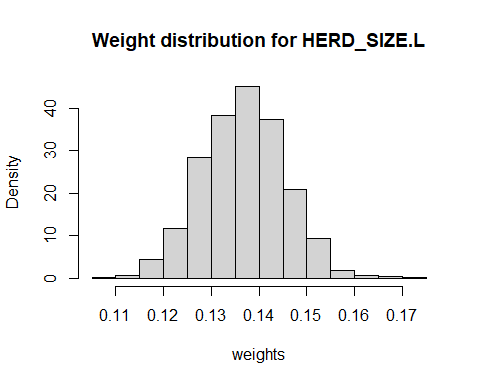
Es wurde für die Hyperparameter alpha=0.02 und lambda=0.31001 der höchste AUC=0.708 gefunden. Die Werte für alpha und lambda lassen sich in lambda1=0.0062002 und lambda2=0.3038098 umwandeln.



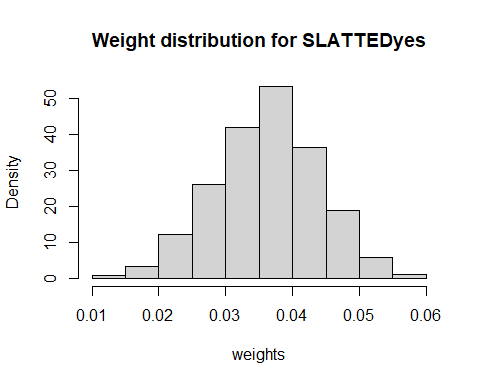
### Weights

trained\_lin\_models <- get\_trained\_models(HERD\_MRSA ~ .,  
 MRSA\_schweineherden\_reduced,  
 auc\_alpha\_lin, auc\_lambda\_lin,   
 train\_index)

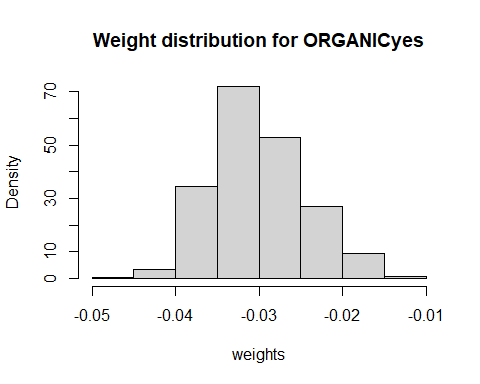
weight\_list <- lapply(trained\_lin\_models, function(x) x$beta)  
plot\_weights\_of\_feature(weight\_list, "HERD\_SIZE.L")



plot\_weights\_of\_feature(weight\_list, "SLATTEDyes")



plot\_weights\_of\_feature(weight\_list, "ORGANICyes")



## Quadratischer Ansatz: Alle Haupt- und Interaktionsterme

train\_index <- createMultiFolds(MRSA\_schweineherden\_reduced$HERD\_MRSA,  
 k = 10,  
 times = 10)

# lambda Sequenz für Formel Herd\_MRSA ~ .^2  
alpha\_seq <- seq(0,1,0.01)  
lambda\_seq <- seq(0.0001,0.3,0.01)  
  
start\_time <- Sys.time()  
# Die Formel drückte alle Haupt- und alle Interaktionsterme aus  
result\_cv\_quad <- my\_glmnet\_cv(formula = HERD\_MRSA ~ .^2,   
 data = MRSA\_schweineherden\_reduced,   
 train\_index,   
 alpha\_seq = alpha\_seq,  
 lambda\_seq = lambda\_seq)  
print(Sys.time() - start\_time)

## Time difference of 9.694276 mins

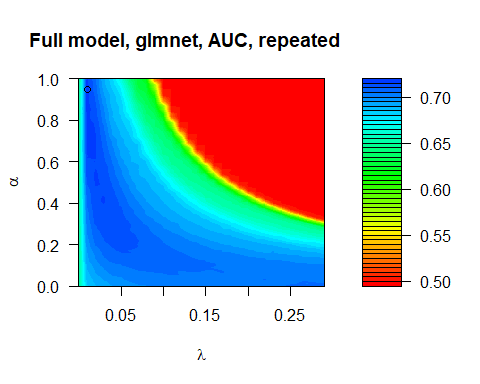
best\_auc\_quad <- max(result\_cv\_quad$ave\_auc\_test)  
best\_index <- which(result\_cv\_quad$ave\_auc\_test == best\_auc\_quad,   
 arr.ind = TRUE)  
  
print(best\_index)

## row col  
## [1,] 2 96

auc\_alpha\_quad <- alpha\_seq[ best\_index[2] ]  
auc\_lambda\_quad <- lambda\_seq[ best\_index[1] ]  
  
cat('The best Test AUC=', round(best\_auc\_quad,3),   
 ' for alpha=', auc\_alpha\_quad, 'and lambda=', auc\_lambda\_quad, '.\n')

## The best Test AUC= 0.717 for alpha= 0.95 and lambda= 0.0101 .

Es wurde für die Hyperparameter alpha=0.95 und lambda=0.0101 der höchste AUC=0.717 gefunden. Die Werte für alpha und lambda lassen sich in lambda1=0.009595 und lambda2=5.0510^{-4} umwandeln.



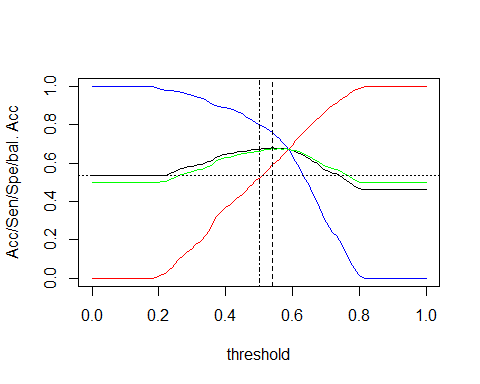
# Gegenüberstellung AUC

result <- data.frame( type\_of\_formular = c('linear', 'quadratisch'),  
 auc = round(c(best\_auc\_lin, best\_auc\_quad),3)#,  
 #se = signif(c(best\_auc\_lin\_se, best\_auc\_quad\_se),1)  
 )  
knitr::kable( result )

|  |  |
| --- | --- |
| type\_of\_formular | auc |
| linear | 0.708 |
| quadratisch | 0.717 |

# Selected Features for best model

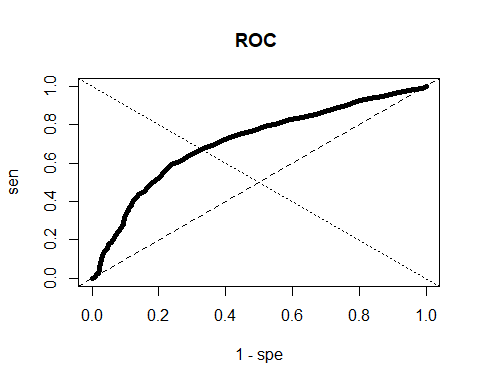
alpha\_glmnet <- auc\_alpha\_quad  
lambda\_glmnet <- auc\_lambda\_quad  
my\_formula <- HERD\_MRSA ~ .^2  
  
trained\_models <- get\_trained\_models(my\_formula,  
 MRSA\_schweineherden\_reduced,  
 alpha\_glmnet, lambda\_glmnet,  
 train\_index)  
# get predicted probabilities  
pred\_prob <- mapply(function(trained\_model, train\_data, formula)  
 {  
 df\_test <- MRSA\_schweineherden\_reduced[-train\_data,]  
 test\_design\_matrix <- model.matrix(formula, df\_test)  
 predict(trained\_model,  
 newx = test\_design\_matrix,  
 type = "response")  
 },   
 trained\_models, train\_index,  
 MoreArgs = list(formula = my\_formula))  
pred\_prob <- unlist(pred\_prob)  
# get references classes of test data  
ref\_class <- unlist(lapply(train\_index,   
 function(x) MRSA\_schweineherden\_reduced$HERD\_MRSA[-x]))  
# plot everything for test data  
plot\_func(pred\_prob, ref\_class)



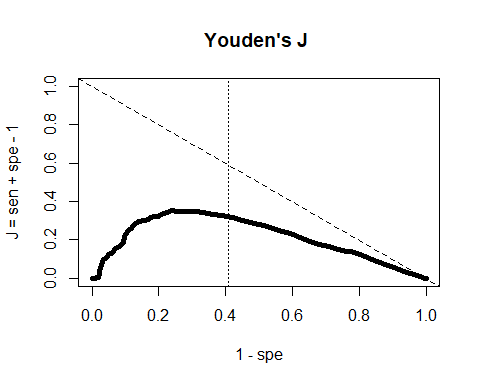
## Max. Acc = 0.683 at Threshold = 0.54   
## Max. bal. Acc = 0.6770425 at Threshold = 0.54

## Setting levels: control = negative, case = positive

## Setting direction: controls < cases



##   
## Call:  
## roc.default(response = reference, predictor = model\_fit\_response)  
##   
## Data: model\_fit\_response in 1860 controls (reference negative) < 2140 cases (reference positive).  
## Area under the curve: 0.7106



## Sen = 0.5919355 ; Spe = 0.7621495 ; threshold = 0.54

## [1] 0.54

library(roperators)

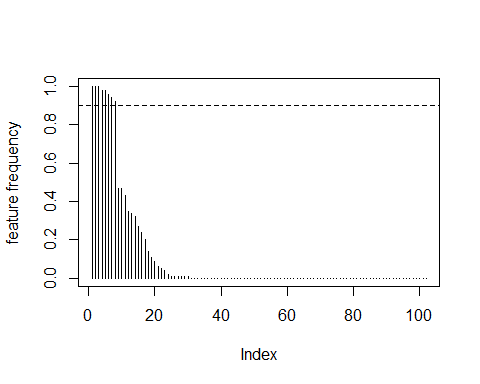
## Warning: package 'roperators' was built under R version 4.0.5

##   
## Attaching package: 'roperators'

## The following object is masked from 'package:ggplot2':  
##   
## %+%

selected\_features <- mapply(function(trained\_model, train\_data, formula)  
 {  
 df\_test <- MRSA\_schweineherden\_reduced[-train\_data,]  
 test\_design\_matrix <- model.matrix(formula, df\_test)  
 predict(trained\_model,  
 newx = test\_design\_matrix,  
 type = "nonzero")  
 },   
 trained\_models, train\_index,  
 MoreArgs = list(formula = my\_formula))  
# get selection frequeincy  
feature\_names <- colnames(model.matrix(my\_formula, MRSA\_schweineherden\_reduced))  
features <- rep(0,length(feature\_names))  
names(features) <- feature\_names  
for (this in selected\_features)  
 features[this] %+=% 1

features <- sort(features, decreasing = TRUE)  
features <- features / max(features)  
plot(features, type = "h", ylab = "feature frequency")  
abline(h = 0.9, lty = 2)



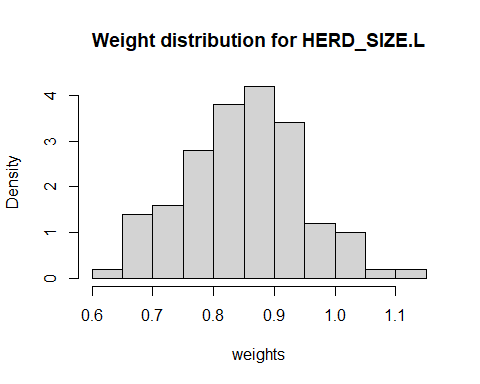
Name the most important Features, i.e. all that appear in at least 9 of 10 cases:

features[features >= 0.9]

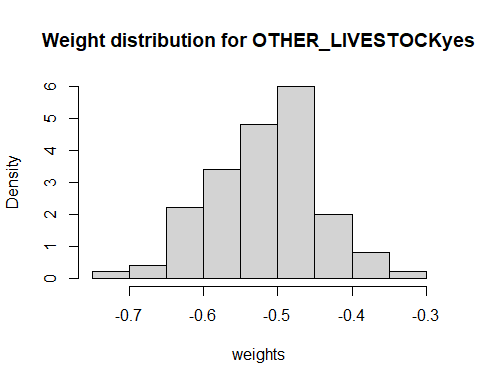
## HERD\_SIZE.L OTHER\_LIVESTOCKyes   
## 1.00 1.00   
## HERD\_TYPEgrower:HERD\_SIZE.Q HERD\_TYPEweaner:COMPANIONyes   
## 1.00 0.98   
## AM\_DRUGyes:CLEAN\_UPyes PURCHASEyes:SLATTEDyes   
## 0.98 0.96   
## AM\_DRUGyes:SLATTEDyes COMPANIONyes:ALL\_IN\_or\_OUTyes   
## 0.94 0.92

# Distribution of weights

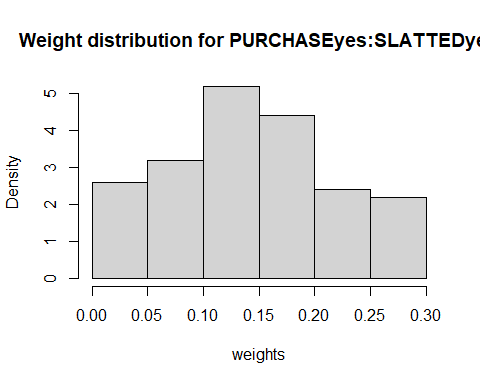
weight\_list <- lapply(trained\_models, function(x) x$beta)  
plot\_weights\_of\_feature(weight\_list, "HERD\_SIZE.L")



plot\_weights\_of\_feature(weight\_list, "OTHER\_LIVESTOCKyes")



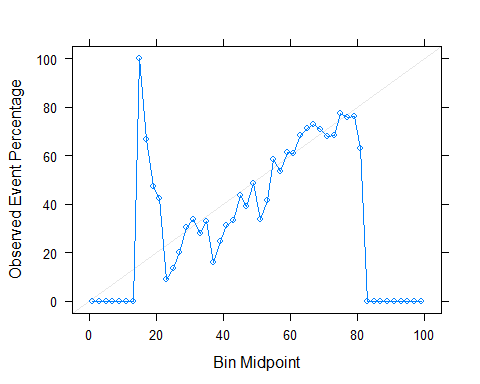
plot\_weights\_of\_feature(weight\_list, "PURCHASEyes:SLATTEDyes")



# Take a closer look – predicted distribution of risk, and its parts

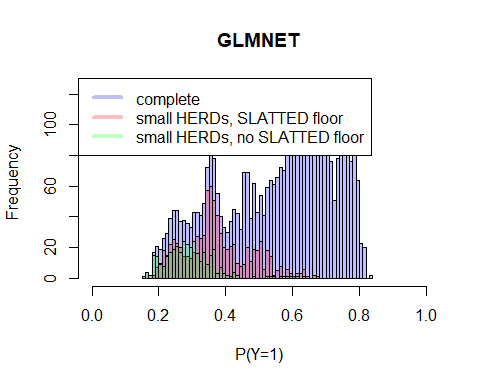
Plot the probaility calibration curve

df\_cal <- data.frame(Class = ref\_class)  
df\_cal$glmnet <- pred\_prob  
cal\_obj <- caret::calibration(Class ~ glmnet,  
 data = df\_cal,  
 cuts = 50,  
 class = "positive")  
plot(cal\_obj)



my\_subset <- function(df\_input, selection\_rule)  
{  
 selected\_col <- selection\_rule[1]  
 selected\_value <- selection\_rule[2]  
 df\_output <- NULL  
 for (i in seq\_len(nrow(df\_input)))  
 {  
 if (df\_input[i,selected\_col] == selected\_value)  
 df\_output <- rbind.data.frame(df\_output,  
 df\_input[i,])  
 }  
 if (!is.null(df\_output))  
 colnames(df\_output) <- colnames(df\_input)  
 df\_output  
}  
  
get\_risk\_subset <- function(df\_input, trained\_models,   
 train\_index, subset\_list)  
{  
 result <- mapply(function(trained\_model, train\_data, formula)  
 {  
 df\_test <- df\_input[-train\_data,]  
 for (this\_subset in subset\_list)  
 {  
 df\_test <- my\_subset(df\_test, this\_subset)  
 if (is.null(df\_test))  
 return(NULL)  
 }  
 test\_design\_matrix <- model.matrix(formula, df\_test)  
 predict(trained\_model,  
 newx = test\_design\_matrix,  
 type = "response")  
 },  
 trained\_models, train\_index,  
 MoreArgs = list(formula = my\_formula))  
 # remove empty entries in result list  
 result <- Filter(Negate(is.null), result)  
 unname(unlist(result)) # simple vector with predicted risks  
}

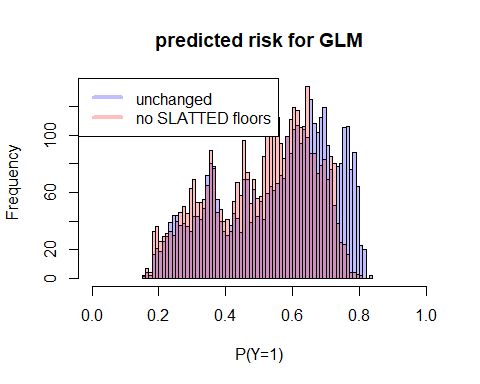
# original distribution of predicted risk  
org\_hist <- hist(pred\_prob,  
 breaks = -1\*Reduce("-", range(pred\_prob))/0.01,  
 plot = FALSE)  
  
sub\_dist\_A <- get\_risk\_subset(MRSA\_schweineherden\_reduced,  
 trained\_models, train\_index,  
 list(c("HERD\_SIZE", "small"),  
 c("SLATTED", "yes")))  
  
sub\_dist\_B <- get\_risk\_subset(MRSA\_schweineherden\_reduced,  
 trained\_models, train\_index,  
 list(c("HERD\_SIZE", "small"),  
 c("SLATTED", "no")))  
  
# distribution of stratified sample  
sub\_dist\_A\_hist <- hist(sub\_dist\_A,  
 breaks = -1\*Reduce("-", range(sub\_dist\_A))/0.01,  
 plot = FALSE)  
sub\_dist\_B\_hist <- hist(sub\_dist\_B,  
 breaks = -1\*Reduce("-", range(sub\_dist\_B))/0.01,  
 plot = FALSE)  
# plot it all  
plot(org\_hist,  
 freq = TRUE,  
 col = rgb(0,0,1,1/4),  
 xlim = c(0,1),  
 xlab = "P(Y=1)",  
 main = "GLMNET")  
plot(sub\_dist\_A\_hist,  
 freq = TRUE,  
 add = TRUE,  
 col = rgb(1,0,0,1/4))  
plot(sub\_dist\_B\_hist,  
 freq = TRUE,  
 add = TRUE,  
 col = rgb(0,1,0,1/4))  
legend("topleft",  
 legend = c("complete",  
 "small HERDs, SLATTED floor",#"sub\_dist\_A",  
 "small HERDs, no SLATTED floor"),#"sub\_dist\_B"),  
 col = c(rgb(0,0,1,1/4),  
 rgb(1,0,0,1/4),  
 rgb(0,1,0,1/4)),  
 lty = 1, lwd = 4)



# Change in predicted risk distribution

get\_mod\_pred\_risk <- function(df\_input, trained\_models,   
 train\_index, subset\_list)  
{  
 col\_name <- subset\_list[1]  
 new\_value <- subset\_list[2]  
 result <- mapply(function(trained\_model, train\_data, formula)  
 {  
 df\_test <- df\_input[-train\_data,]  
 lev <- levels(df\_test[[col\_name]])  
 df\_test[,col\_name] <- factor(rep(new\_value,   
 nrow(df\_test)),  
 levels = lev)  
 test\_design\_matrix <- model.matrix(formula, df\_test)  
 predict(trained\_model,  
 newx = test\_design\_matrix,  
 type = "response")  
 },  
 trained\_models, train\_index,  
 MoreArgs = list(formula = my\_formula))  
 # remove empty entries in result list  
 result <- Filter(Negate(is.null), result)  
 unname(unlist(result)) # simple vector with predicted risks  
}

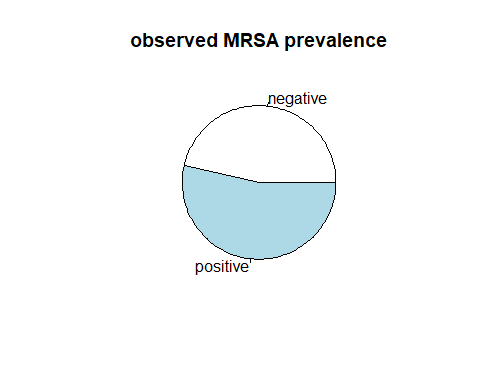
# original distribution of predicted risk  
org\_hist <- hist(pred\_prob,  
 breaks = -1\*Reduce("-", range(pred\_prob))/0.01,  
 plot = FALSE)  
  
sub\_dist\_A <- get\_mod\_pred\_risk(MRSA\_schweineherden\_reduced,  
 trained\_models, train\_index,  
 c("SLATTED", "no"))  
  
# distribution of stratified sample  
sub\_dist\_A\_hist <- hist(sub\_dist\_A,  
 breaks = -1\*Reduce("-", range(sub\_dist\_A))/0.01,  
 plot = FALSE)  
  
# plot it all  
plot(org\_hist,  
 freq = TRUE,  
 col = rgb(0,0,1,1/4),  
 xlim = c(0,1),  
 ylim = range(org\_hist$counts, sub\_dist\_A\_hist$counts),  
 xlab = "P(Y=1)",  
 main = "predicted risk for GLM")  
plot(sub\_dist\_A\_hist,  
 freq = TRUE,  
 add = TRUE,  
 col = rgb(1,0,0,1/4))  
legend("topleft",  
 legend = c("unchanged",  
 "no SLATTED floors"),#"sub\_dist\_A",  
 col = c(rgb(0,0,1,1/4),  
 rgb(1,0,0,1/4)),  
 lty = 1, lwd = 4)



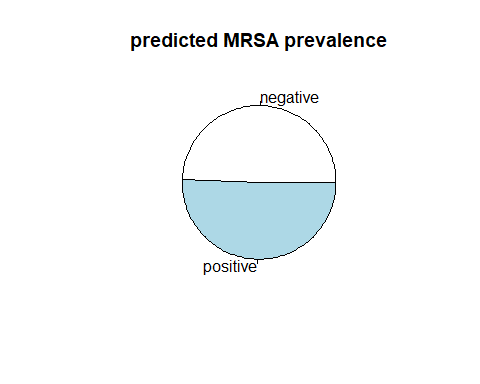
get\_mod\_pred\_prev <- function(df\_input, trained\_models,   
 train\_index, subset\_list, threshold)  
{  
 col\_name <- subset\_list[1]  
 new\_value <- subset\_list[2]  
 result <- mapply(function(trained\_model, train\_data, formula, threshold)  
 {  
 df\_test <- df\_input[-train\_data,]  
 lev <- levels(df\_test[[col\_name]])  
 df\_test[,col\_name] <- factor(rep(new\_value,   
 nrow(df\_test)),  
 levels = lev)  
 test\_design\_matrix <- model.matrix(formula, df\_test)  
 pred <- predict(trained\_model,  
 newx = test\_design\_matrix,  
 type = "response")  
 ifelse(pred > threshold, 1, 0)  
 },   
 trained\_models, train\_index,  
 MoreArgs = list(formula = my\_formula,  
 threshold = threshold))  
 # remove empty entries in result list  
 result <- Filter(Negate(is.null), result)  
 unname(unlist(result)) # simple vector with predicted risks  
}

threshold <- 0.59  
pred\_prev <- mapply(function(trained\_model, train\_data, formula)  
 {  
 test\_design\_matrix <- model.matrix(formula,   
 MRSA\_schweineherden\_reduced)  
 pred <- predict(trained\_model,  
 newx = test\_design\_matrix,  
 type = "response")  
 ifelse(pred > threshold, 1, 0)  
 },   
 trained\_models, train\_index,  
 MoreArgs = list(formula = my\_formula))  
pred\_prev <- mean(unlist(pred\_prev))  
  
mod\_pred\_prev <- get\_mod\_pred\_prev(MRSA\_schweineherden\_reduced,   
 trained\_models,  
 train\_index,  
 c("SLATTED", "no"),  
 threshold)  
mod\_pred\_prev <- mean(mod\_pred\_prev)

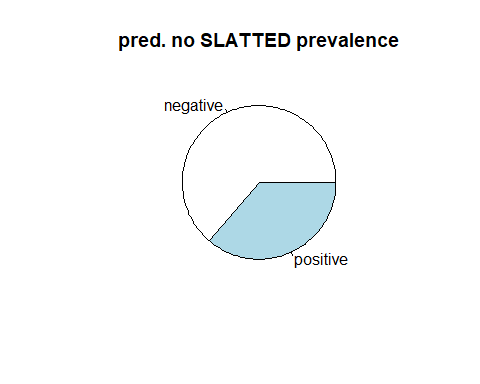
obs\_prev <- mean(MRSA\_schweineherden\_reduced$HERD\_MRSA == "positive")  
pie(c(1 - obs\_prev, obs\_prev),  
 labels = c("negative", "positive"),  
 main = 'observed MRSA prevalence')



pie(c(1 - pred\_prev, pred\_prev),   
 labels = c("negative", "positive"),  
 main = 'predicted MRSA prevalence')



pie(c(1 - mod\_pred\_prev, mod\_pred\_prev),   
 labels = c("negative", "positive"),  
 main = 'pred. no SLATTED prevalence')



stopCluster(cl)