Fallstudie

Urs Walcher

2019-07-15

Contents

Bibliotheken laden

```
## Bliotheken laden

library(caret)
library(readr)
library(dplyr)
library(corrplot)
library(caret)
library(randomForest)
```

Daten einlesen und aufbereiten

Dateien einlesen

Stuktur der Daten anzeigen

```
## 'data.frame':    576 obs. of  6 variables:
##  $ X                          : int  619 664 441 160 358 335 47 164 736 436 ...
##  $ Months.since.Last.Donation : int  2 0 1 2 1 4 2 1 5 0 ...
##  $ Number.of.Donations        : int  50 13 16 20 24 4 7 12 46 3 ...
##  $ Total.Volume.Donated..c.c..: int  12500 3250 4000 5000 6000 1000 1750 3000 11500 750 ...
##  $ Months.since.First.Donation: int  98 28 35 45 77 4 14 35 98 4 ...
##  $ Made.Donation.in.March.2007: int  1 1 1 1 0 0 1 0 1 0 ...
```

```
## [1] 576   6
```

576 Zeilen Trainingsdaten (Observations) und 6 Spalten (Variablen) eingelesen.

```
## 'data.frame':    200 obs. of  5 variables:
##  $ X                          : int  659 276 263 303 83 500 530 244 249 728 ...
##  $ Months.since.Last.Donation : int  2 21 4 11 4 3 4 14 23 14 ...
##  $ Number.of.Donations        : int  12 7 1 11 12 21 2 1 2 4 ...
##  $ Total.Volume.Donated..c.c..: int  3000 1750 250 2750 3000 5250 500 250 500 1000 ...
##  $ Months.since.First.Donation: int  52 38 4 38 34 42 4 14 87 64 ...
```

```
## [1] 200   5
```

200 Zeilen Validierungssdaten (Observations) und 5 Spalten (Variablen) eingelesen.

Daten bereinigen

Spaltennamen anpassen

```
data_tst <- data_tst %>%
  rename(
    id = "X",
    msld = "Months.since.Last.Donation",
    nod = "Number.of.Donations",
    tvd = "Total.Volume.Donated..c.c..",
    msfd = "Months.since.First.Donation",
    mdim07 = "Made.Donation.in.March.2007"
  )

str(data_tst)
```

```
## 'data.frame':    576 obs. of  6 variables:
##  $ id    : int  619 664 441 160 358 335 47 164 736 436 ...
##  $ msld  : int  2 0 1 2 1 4 2 1 5 0 ...
##  $ nod   : int  50 13 16 20 24 4 7 12 46 3 ...
##  $ tvd   : int  12500 3250 4000 5000 6000 1000 1750 3000 11500 750 ...
##  $ msfd  : int  98 28 35 45 77 4 14 35 98 4 ...
##  $ mdim07: int  1 1 1 1 0 0 1 0 1 0 ...
```

Alle Spaltennamen auf Kurzform angepasst (erster Wortbuchstabe verwendet).

Daten auf unvollständige Zeilen prüfen

N/A-Werte

```
# Auf fehlende "N/A" Werte prüfen
na_tst <- sapply(data_tst,function(x) sum(is.na(x)))
na_prd <- sapply(data_prd,function(x) sum(is.na(x)))
print(na_tst)
```

```
##     id   msld    nod    tvd   msfd mdim07
##      0      0      0      0      0      0
```

```
print(na_prd)
```

```
##                            X  Months.since.Last.Donation
##                            0                           0
##           Number.of.Donations Total.Volume.Donated..c.c..
##                            0                           0
## Months.since.First.Donation
##                            0
```

Keine“N/A” Werte vorhanden, die korrigert werden müssen.

Leerzeichen

```
# Auf fehlende " " Werte prüfen
na_tst <- sapply(data_tst,function(x) sum(x==""))
na_prd <- sapply(data_prd,function(x) sum(x==""))
print(na_tst)
```

```
##     id   msld    nod    tvd   msfd mdim07
##      0      0      0      0      0      0
```

```
print(na_prd)
```

```
##                            X  Months.since.Last.Donation
##                            0                           0
##           Number.of.Donations Total.Volume.Donated..c.c..
##                            0                           0
## Months.since.First.Donation
##                            0
```

Keine Leerzeichen vorhanden, die korrigert werden müssen.

Werte korrigieren

```
#Sollte es Nullwerte haben könnte man die Imputation anwenden (Beispiel)
if(na_tst > 0){
print("NULLWERT!!!!!!!")
preproc_df = preProcess(df, method = "bagImpute")
df <- predict(preproc_df, df)
}
```

```
## Warning in if (na_tst > 0) {: the condition has length > 1 and only the
## first element will be used
```

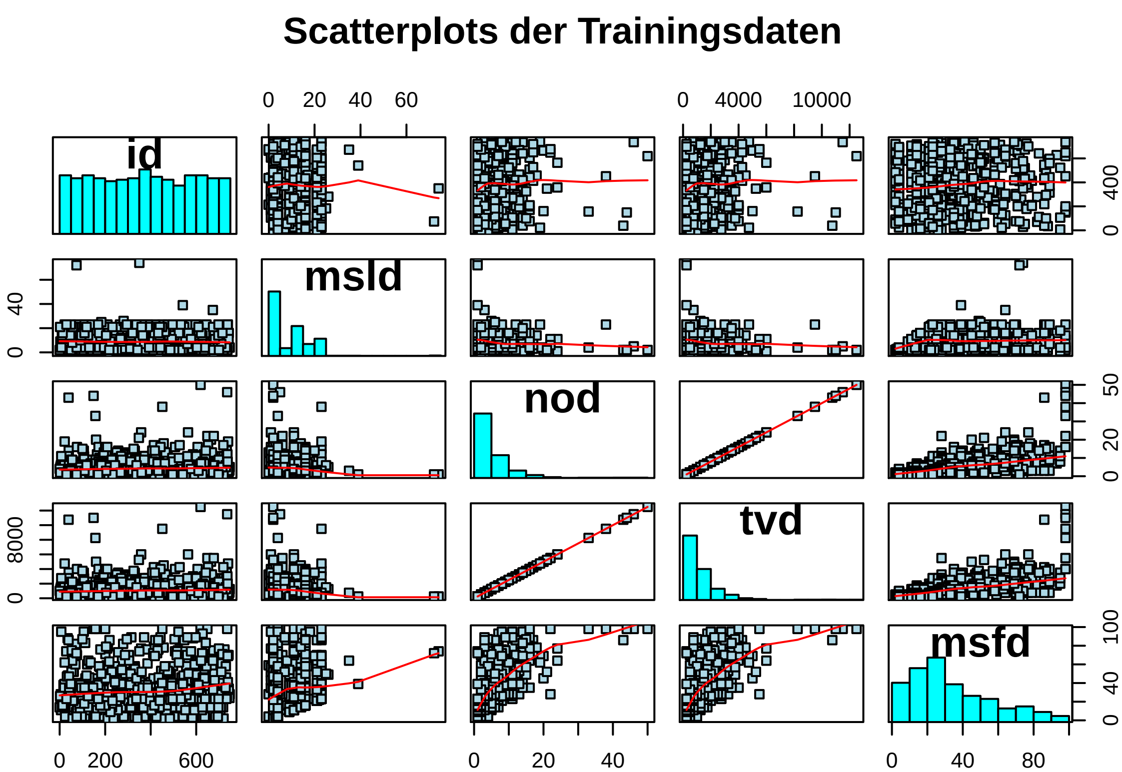
Beispiel Datenkorrektur.

Daten auf doppelte Zeilen prüfen

Nur zu Dokumentationszwecken verwendet.

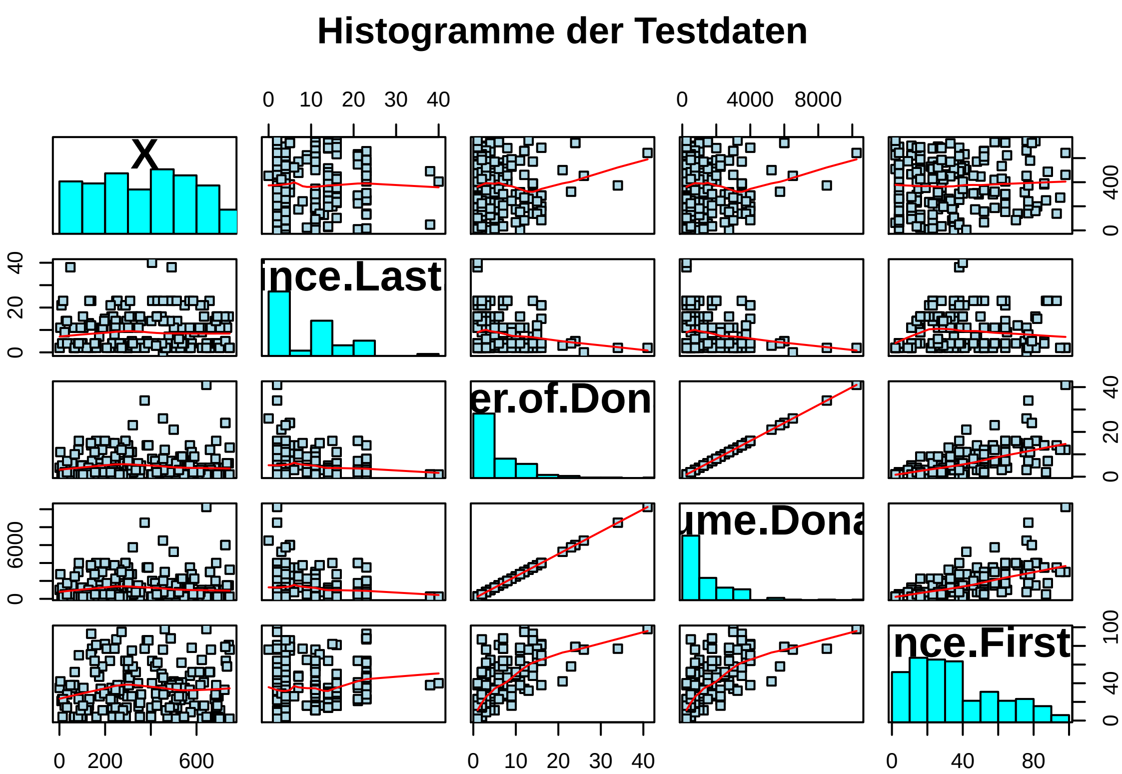
Erste Datenanalyse

Vergleich der gelieferten Trainings- und Test-Daten




```
##          id             msld             nod              tvd
##  Min.   :  0.0   Min.   : 0.000   Min.   : 1.000   Min.   :  250
##  1st Qu.:183.8   1st Qu.: 2.000   1st Qu.: 2.000   1st Qu.:  500
##  Median :375.5   Median : 7.000   Median : 4.000   Median : 1000
##  Mean   :374.0   Mean   : 9.439   Mean   : 5.427   Mean   : 1357
##  3rd Qu.:562.5   3rd Qu.:14.000   3rd Qu.: 7.000   3rd Qu.: 1750
##  Max.   :747.0   Max.   :74.000   Max.   :50.000   Max.   :12500
##       msfd           mdim07
##  Min.   : 2.00   Min.   :0.0000
##  1st Qu.:16.00   1st Qu.:0.0000
##  Median :28.00   Median :0.0000
##  Mean   :34.05   Mean   :0.2396
##  3rd Qu.:49.25   3rd Qu.:0.0000
##  Max.   :98.00   Max.   :1.0000
```

Scatterplots und "Summary" der Trainingsdaten.




```
##        X          Months.since.Last.Donation Number.of.Donations
##  Min.   :  1.0    Min.   : 0.000             Min.   : 1.000
##  1st Qu.:198.2    1st Qu.: 4.000             1st Qu.: 2.000
##  Median :377.5    Median : 7.000             Median : 4.000
##  Mean   :374.6    Mean   : 9.495             Mean   : 5.935
##  3rd Qu.:537.0    3rd Qu.:14.000             3rd Qu.: 8.000
##  Max.   :745.0    Max.   :40.000             Max.   :41.000
##  Total.Volume.Donated..c.c.. Months.since.First.Donation
##  Min.   :  250               Min.   : 2.00
##  1st Qu.:  500               1st Qu.:14.00
##  Median : 1000               Median :31.00
##  Mean   : 1484               Mean   :35.48
##  3rd Qu.: 2000               3rd Qu.:52.00
##  Max.   :10250               Max.   :98.00
```

Scatterplots und "Summary" der Validierungsdaten.

Fazit dese Vergleiches: - Trainings- und Validierungsdaten stimmen ziemlich überein. Die Daten können verwendet werden. Auch das "Summary" liefert annähernd die gleichen Werte.

Schneller, visualisierter Blick in die Daten

Daten in "train" und "validate" aufteilen

```
# Daten in Trainings- und Testdaten aufteilen

partition <- createDataPartition(data_tst[,1], times = 1, p = 0.75,list = FALSE)
train <- data_tst[partition,] # Trainings-Daten
validate  <- data_tst[-partition,] # Test-Daten
```

```r
dim(train)
```

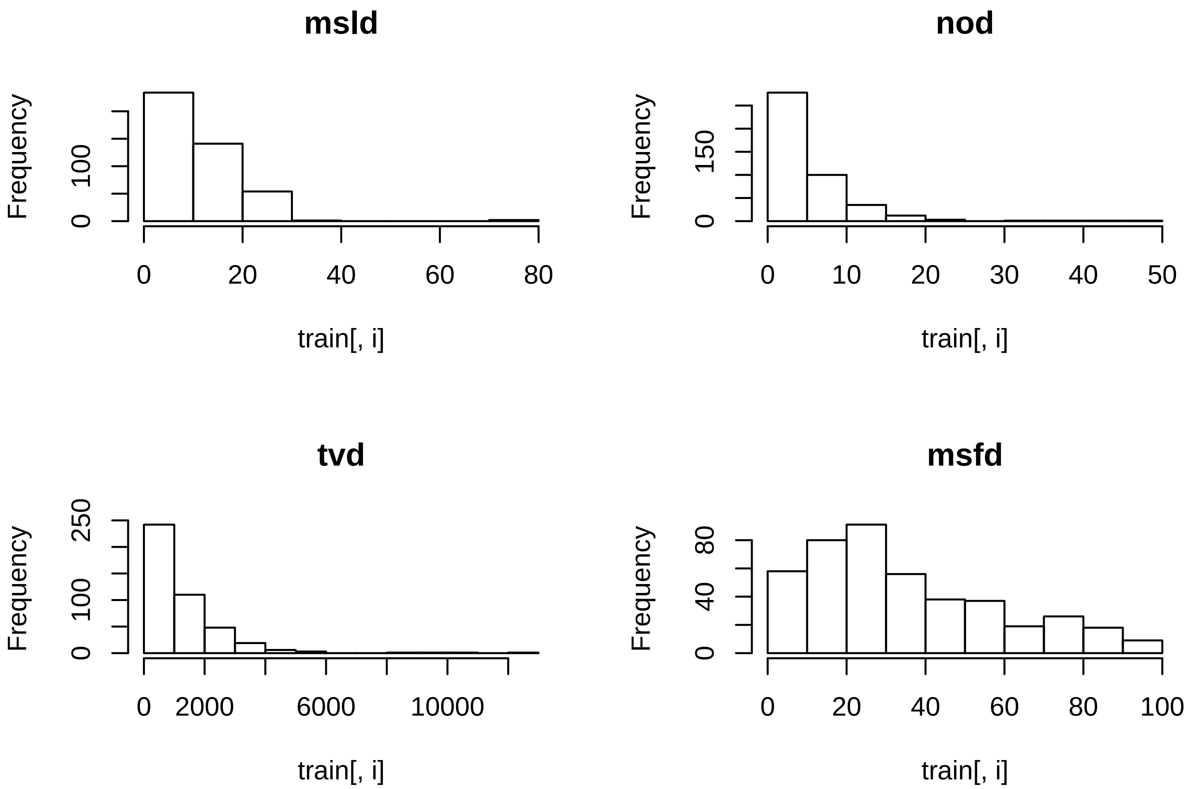
```
## [1] 432   6
```

```r
dim(validate)
```

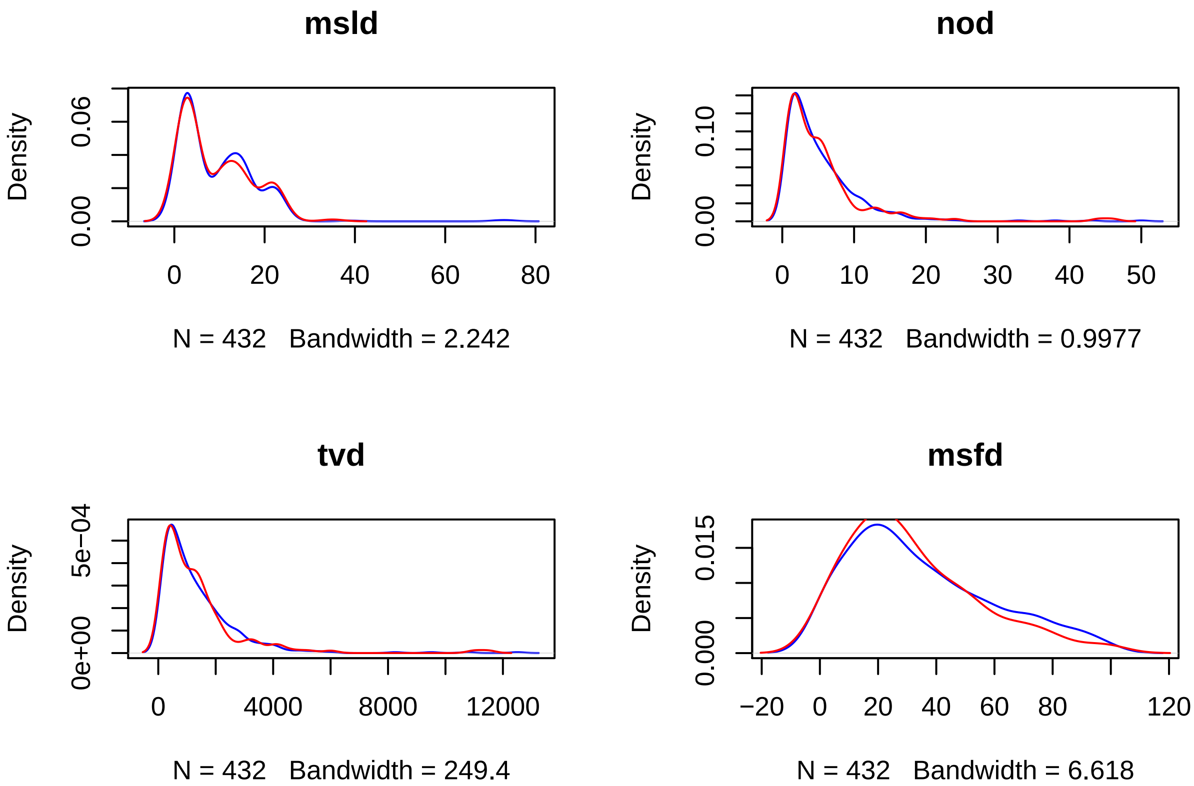
```
## [1] 144   6
```

Daten in Trainings- und Testdaten aufgeteilt.

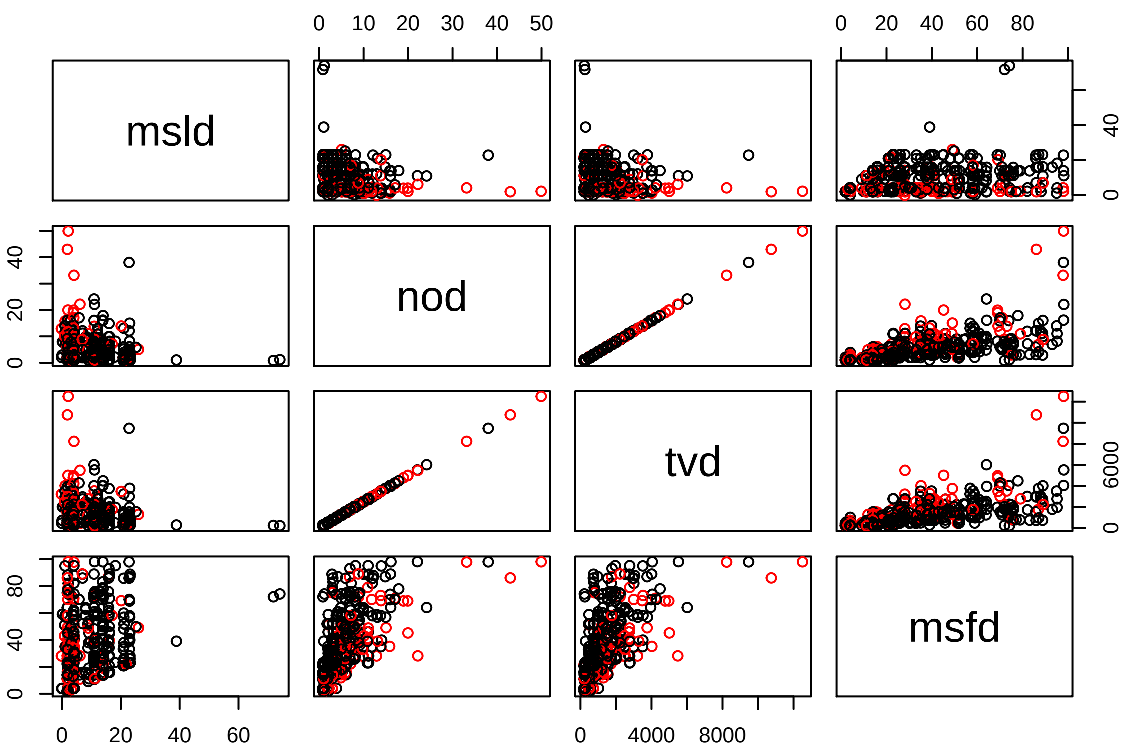
Frequenz anzeigen



Density anzeigen



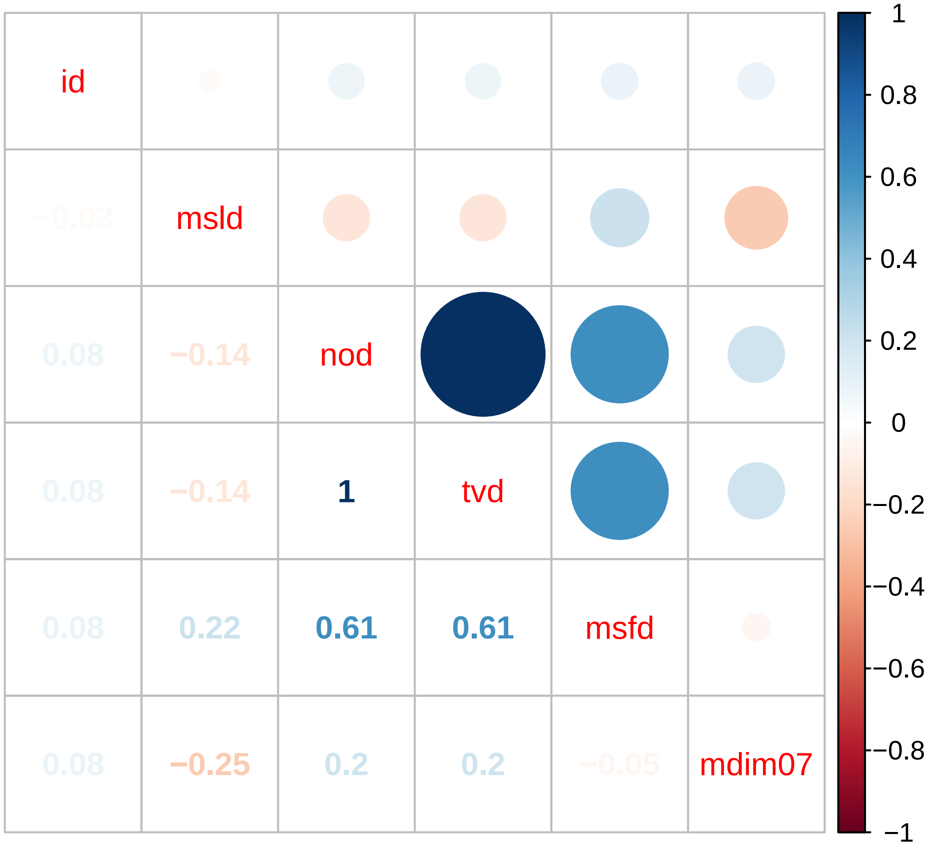
Gesamtüberblick der Daten



Korrelationen

Schnellübersicht

```
##            msld        nod        tvd      msfd
## msld  1.0000000 -0.1352728 -0.1352728 0.2164199
## nod  -0.1352728  1.0000000  1.0000000 0.6115297
## tvd  -0.1352728  1.0000000  1.0000000 0.6115297
## msfd  0.2164199  0.6115297  0.6115297 1.0000000
```




```
## [1] 0.6115297
```

Die Werte zeigen eine Korrelation zwischen msfd (Month since first donation) und der Anzahl der Spenden (nod(numbers of donation)). Da tvd & nod in Abhängigkeit zueinander stehen, kann tvd entfernt werden.

Unnötige Variabeln entfernen

```
# Variable "tvd" entfernen

useless <- c("tvd")
train <- train[,!(names(train) %in% useless)]
validate <- validate[,!(names(validate) %in% useless)]
str(train)
```

```
## 'data.frame':    432 obs. of  5 variables:
##  $ id    : int  619 664 441 160 335 164 436 499 191 638 ...
```

```
##  $ msld  : int  2 0 1 2 4 1 0 2 2 2 ...
##  $ nod   : int  50 13 16 20 4 12 3 6 15 6 ...
##  $ msfd  : int  98 28 35 45 4 35 4 15 49 15 ...
##  $ mdim07: int  1 1 1 1 0 0 0 1 1 1 ...
```

```
str(validate)
```

```
## 'data.frame':    144 obs. of  5 variables:
##  $ id    : int  358 47 736 460 285 356 40 8 482 298 ...
##  $ msld  : int  1 2 5 2 1 2 2 2 4 2 ...
##  $ nod   : int  24 7 46 10 13 5 14 6 8 12 ...
##  $ msfd  : int  77 14 98 28 47 11 48 16 21 47 ...
##  $ mdim07: int  0 1 1 1 0 1 1 1 0 1 ...
```

Variablen “Total volume donated” entfernt (überflüssig).

Variable “mdim07” in Faktor umwandeln

```
# Variable "mdim07" in Faktor umwandeln

req_labels <- train['mdim07']
rec_labels <- recode(req_labels$mdim07,'0' = "No", '1' = "Yes")
train$mdim07 <- rec_labels
train$mdim07  <-as.factor(train$mdim07)

str(train)
```

```
## 'data.frame':    432 obs. of  5 variables:
##  $ id    : int  619 664 441 160 335 164 436 499 191 638 ...
##  $ msld  : int  2 0 1 2 4 1 0 2 2 2 ...
##  $ nod   : int  50 13 16 20 4 12 3 6 15 6 ...
##  $ msfd  : int  98 28 35 45 4 35 4 15 49 15 ...
##  $ mdim07: Factor w/ 2 levels "No","Yes": 2 2 2 2 1 1 1 2 2 2 ...
```

Variable “mdim07” in Faktor umwandeln.

Machine Learning

Logistische Regression

Mit Standartwerten und mit “logloss” als Metrik

```
# Standartwerte setzen

trainControl <- trainControl(method="repeatedcv", summaryFunction=mnLogLoss, number=10, repeats=3, clas

metric <- "logLoss"

# Logistische Regressionen

set.seed(101)
fit.glm <- train(mdim07~., data=train, method="glm", metric=metric, trControl=trainControl) # GLM
```

```r
set.seed(101)
fit.lda <- train(mdim07~., data=train, method="lda", metric=metric, trControl=trainControl) # LDA

set.seed(101)
fit.glmnet <- train(mdim07~., data=train, method="glmnet", metric=metric,trControl=trainControl)  # GLM

set.seed(101)
fit.cart <- train(mdim07~., data=train, method="rpart", metric=metric,trControl=trainControl)  # CART

set.seed(101)
fit.svm <- train(mdim07~., data=train, method="svmRadial", metric=metric, trControl=trainControl)  # SV
```

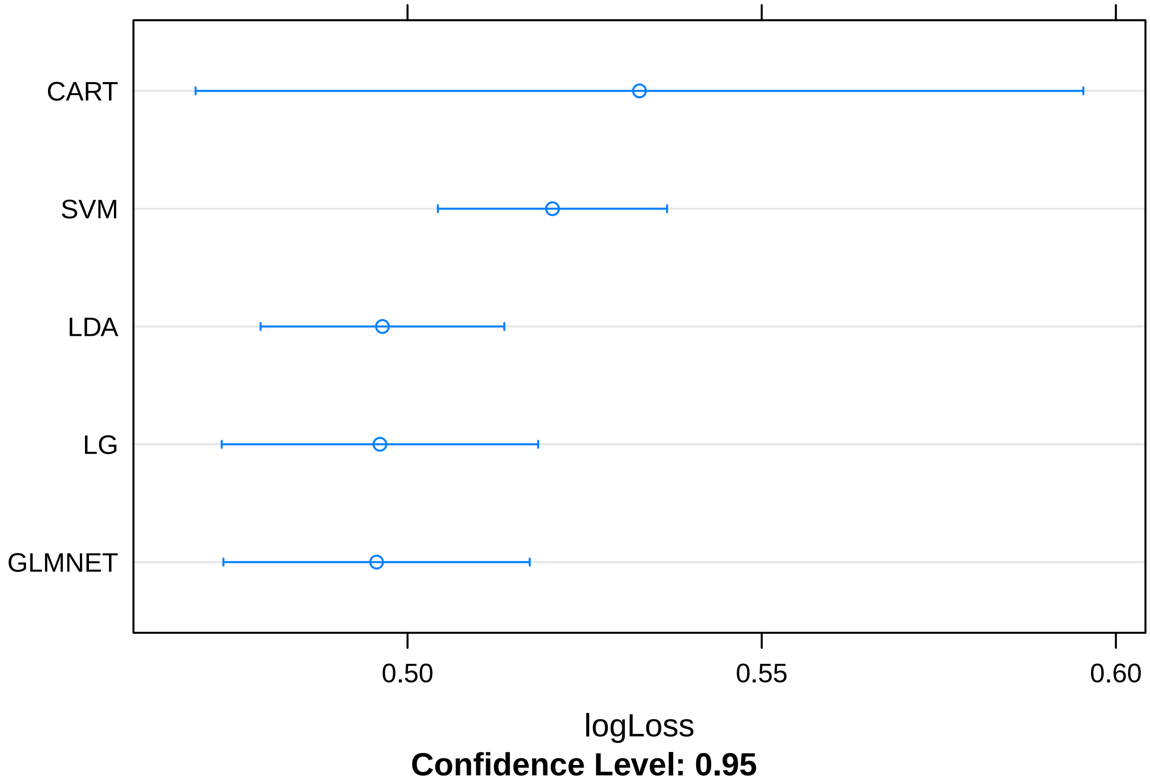
Auswertung

```r
# Auswertung

results <- resamples(list(LG=fit.glm, LDA=fit.lda, GLMNET=fit.glmnet, CART=fit.cart, SVM=fit.svm))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: LG, LDA, GLMNET, CART, SVM
## Number of resamples: 30
##
## logLoss
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LG     0.3893135 0.4463300 0.4852988 0.4961014 0.5353859 0.6055316    0
## LDA    0.4023105 0.4638187 0.4944361 0.4964558 0.5365428 0.5813167    0
## GLMNET 0.3918348 0.4488821 0.4859499 0.4956279 0.5334270 0.6020146    0
## CART   0.3650912 0.4492543 0.5056391 0.5327407 0.5498976 1.3358187    0
## SVM    0.4100113 0.4913000 0.5163180 0.5204602 0.5470081 0.6197136    0
```

```r
dotplot(results)
```



Der "logLoss" sollte möglichst tief sein. "GLMNET" bringt die beste Performance.

Optimierung mit "Box Cox" Transformation und mit "logLoss" als Metrik

```
# Standartwerte und BoxCox setzen

trainControl <- trainControl(method="repeatedcv", summaryFunction=mnLogLoss, number=10, repeats=3, clas

preProcess="BoxCox"

metric <- "logLoss"

# Logistische Regressionen

set.seed(101)
fit.glm <- train(mdim07~., data=train, method="glm", metric=metric, trControl=trainControl, preProc=pre

set.seed(101)
fit.lda <- train(mdim07~., data=train, method="lda", metric=metric, trControl=trainControl, preProc=pre

set.seed(101)
fit.glmnet <- train(mdim07~., data=train, method="glmnet", metric=metric,trControl=trainControl, prePro

set.seed(101)
fit.cart <- train(mdim07~., data=train, method="rpart", metric=metric,trControl=trainControl, preProc=p

set.seed(101)
```

```
fit.svm <- train(mdim07~., data=train, method="svmRadial", metric=metric, trControl=trainControl, prePr
```

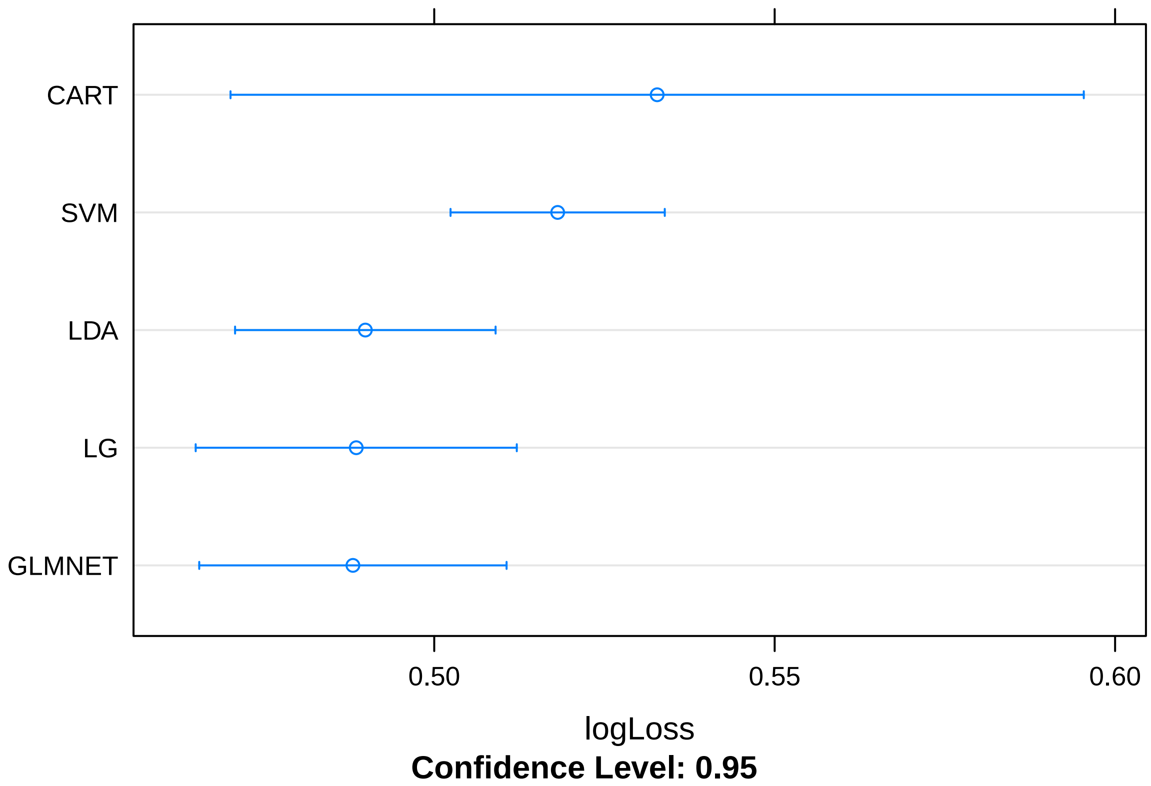
Auswertung

```
## glmnet
##
## 432 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: Box-Cox transformation (2)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 389, 389, 389, 389, 388, 390, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        logLoss
##   0.10   0.0002162577  0.4884103
##   0.10   0.0021625769  0.4880559
##   0.10   0.0216257689  0.4902963
##   0.55   0.0002162577  0.4884475
##   0.55   0.0021625769  0.4882145
##   0.55   0.0216257689  0.4939626
##   1.00   0.0002162577  0.4884714
##   1.00   0.0021625769  0.4883865
##   1.00   0.0216257689  0.5002614
##
## logLoss was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.1 and lambda
##  = 0.002162577.
```

Zeigt die "CoxBox"-Optimierung auf und welchen Wert für "alpha" und "lambda" verwendet wurden.

Auswertung

```
##
## Call:
## summary.resamples(object = results)
##
## Models: LG, LDA, GLMNET, CART, SVM
## Number of resamples: 30
##
## logLoss
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LG     0.3827768 0.4333300 0.4865001 0.4885447 0.5405587 0.6108196    0
## LDA    0.4029320 0.4436962 0.4895205 0.4898811 0.5285904 0.6017458    0
## GLMNET 0.3868571 0.4347305 0.4874971 0.4880559 0.5398231 0.6043571    0
## CART   0.3650912 0.4492543 0.5056391 0.5327407 0.5498976 1.3358187    0
## SVM    0.4171097 0.4902677 0.5109632 0.5181278 0.5476196 0.6200561    0
```



Allgemein leichte Verbesserung bei den Werten und wieder bringt "GLMNET" die beste Performance.

Random Forest, GBM & C5.0

```
trainControl <- trainControl(method="repeatedcv", summaryFunction=mnLogLoss, number=10, repeats=3, class
metric <- "logLoss"
preProcess = "BoxCox"

set.seed(101)
fit.rf <- train(mdim07~., data=train, method="rf", metric=metric, preProc=preProcess, trControl=trainCo
```

```
set.seed(101)
fit.gbm <- train(mdim07~., data=train, method="gbm", metric=metric, preProc=preProcess,
                 trControl=trainControl, verbose=FALSE)  # Gradient Boosting Machine
```

```
set.seed(101)
fit.c50 <- train(mdim07~., data=train, method="C5.0", metric=metric, preProc=preProcess,
                 trControl=trainControl)  # C5.0
```

Auswertung

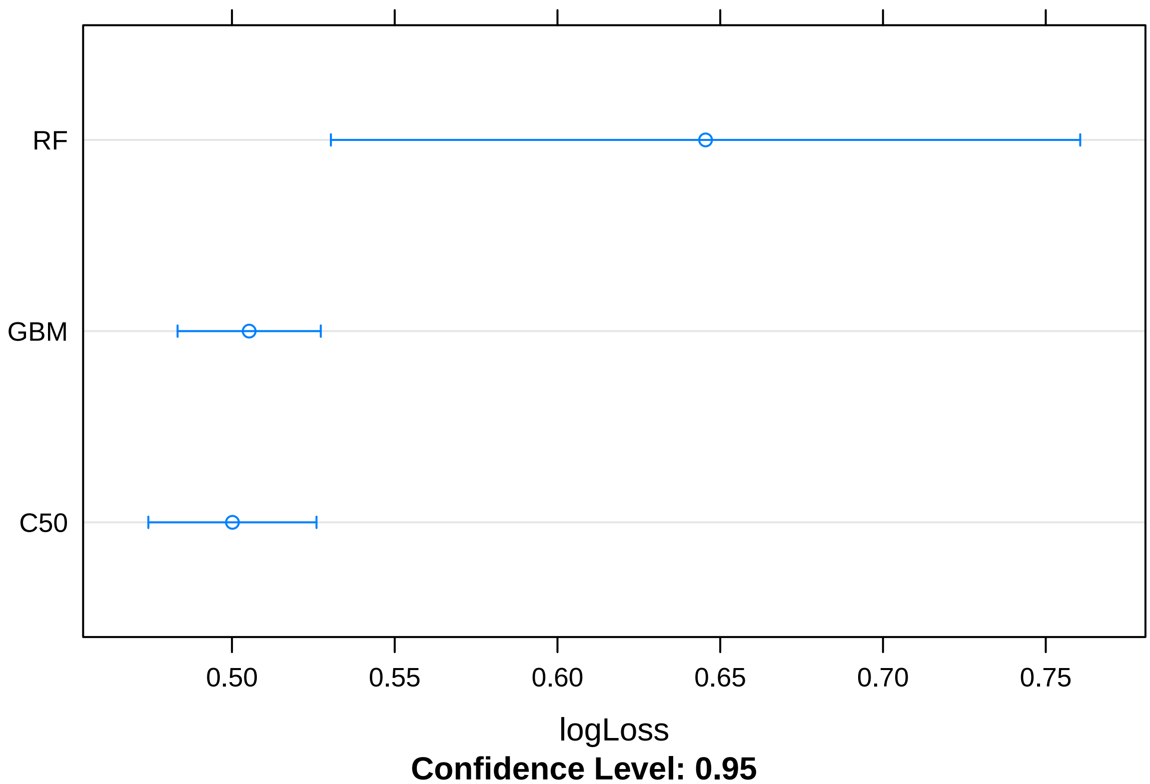
```
# Resultate

ensembleResults <- resamples(list(RF=fit.rf, GBM=fit.gbm, C50=fit.c50))
summary(ensembleResults)
```

```
##
```

```
## Call:
## summary.resamples(object = ensembleResults)
##
## Models: RF, GBM, C50
## Number of resamples: 30
##
## logLoss
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF  0.3943354 0.5074131 0.5906025 0.6454962 0.6348257 2.0291498    0
## GBM 0.4127904 0.4547596 0.5086024 0.5052926 0.5499566 0.6155020    0
## C50 0.3980659 0.4426518 0.4938751 0.5001474 0.5418238 0.6428691    0
```

```
dotplot(ensembleResults)
```



Bei diesen Methoden bringt "GBM" (Gradient Boosting Machine) die besten Werte, aber anhand der schlechteren Laufzeiten im Vergleich zu den logistischen Regressionen, werde ich nur noch "GLMNET" bevorzugen.

Validation

Validation mit dem "Validate-Set" durchführen

```
# Variable "mdim07" in Faktor umwandeln

req_labels <- validate['mdim07']
rec_labels <- recode(req_labels$mdim07,'0' = "No", '1' = "Yes")
validate$mdim07 <- rec_labels
validate$mdim07  <-as.factor(validate$mdim07)
```

```r
str(validate)
```

```
## 'data.frame':    144 obs. of  5 variables:
##  $ id    : int  358 47 736 460 285 356 40 8 482 298 ...
##  $ msld  : int  1 2 5 2 1 2 2 2 4 2 ...
##  $ nod   : int  24 7 46 10 13 5 14 6 8 12 ...
##  $ msfd  : int  77 14 98 28 47 11 48 16 21 47 ...
##  $ mdim07: Factor w/ 2 levels "No","Yes": 1 2 2 2 1 2 2 2 1 2 ...
```

```r
# GLMNET mit dem "validate-Datenset"

set.seed(101)
test.pred <- predict(fit.glmnet, newdata=validate, type  = "prob") # GLMNET
```

Auswertung

```r
# Auswertung

# logLoss kalkulieren
LogLoss <- function(actual, predicted, eps=0.00001) {
  predicted <- pmin(pmax(predicted, eps), 1-eps)
  -1/length(actual)*(sum(actual*log(predicted)+(1-actual)*log(1-predicted)))
}

# Labels wieder in "0" und "1" ändern
req_labels <- validate['mdim07']
rec_labels <- recode(req_labels$mdim07,  "No" = '0' , "Yes" = '1')
validate$mdim07 <- rec_labels

# LogLoss bestimmen
log.loss <- LogLoss(as.numeric(as.character(validate$mdim07)), test.pred$Yes)
print(log.loss)
```

```
## [1] 0.4541126
```

Der "logLoss" sollte möglichst tief sein und ich denke 0.37 ist ein guter Wert. Wir können dies also auf unsere Produktiven-Daten anwenden und die "Submission-Datei erstellen.

Test- oder Produktive-Daten vorhersagen

Vorhersage mit dem "Test-Set" durchführen

```r
# Spaltennamen anpassen

data_prd <- data_prd %>%
    rename(
    id = "X",
    msld = "Months.since.Last.Donation",
    nod = "Number.of.Donations",
    tvd = "Total.Volume.Donated..c.c..",
    msfd = "Months.since.First.Donation",
    )
```

```
str(data_prd)
```

```
## 'data.frame':    200 obs. of  5 variables:
##  $ id  : int  659 276 263 303 83 500 530 244 249 728 ...
##  $ msld: int  2 21 4 11 4 3 4 14 23 14 ...
##  $ nod : int  12 7 1 11 12 21 2 1 2 4 ...
##  $ tvd : int  3000 1750 250 2750 3000 5250 500 250 500 1000 ...
##  $ msfd: int  52 38 4 38 34 42 4 14 87 64 ...
```

```
# Vorhersage durchführen
set.seed(101)
predictions <- predict(fit.glmnet, newdata=data_prd, type  = "prob")
```

Daten preparieren und hochladen

```
# Submissions-Datei einlesen und Daten abfüllen.

submission_format <- read.csv("daten/submission_format.csv", check.names=FALSE)
submission_format <- submission_format[,-2] # Bestehende "Did Donation" entfernen
pred.df <- as.data.frame(predictions$Yes) #Vorhersagen in DataFrame umwandeln
submission_format <- cbind(submission_format, pred.df) # Vorhersage anhängen

submission_format <- submission_format %>% # Spalten umbennenen
    rename(
    ID = "submission_format",
    'Made Donation in March 2007' = "predictions$Yes",
    )

write.csv(submission_format, file="daten/submission_final.csv", row.names=FALSE ) #CSV-Datei erstellen
```

Schlussresultat anzeigen

```
# Submissions-Datei anzeigen.

head(submission_format, n = 25L)
```

```
##     ID Made Donation in March 2007
## 1  659                  0.52669045
## 2  276                  0.14424981
## 3  263                  0.19339374
## 4  303                  0.36380455
## 5   83                  0.48627171
## 6  500                  0.68768123
## 7  530                  0.39914835
## 8  244                  0.05876478
## 9  249                  0.01259396
## 10 728                  0.10295145
## 11 129                  0.16567221
## 12 534                  0.19284135
## 13 317                  0.28583210
## 14 401                  0.21137663
```

```
## 15 696                  0.35609817
## 16 192                  0.07869380
## 17 176                  0.26591809
## 18 571                  0.49124479
## 19 139                  0.09209039
## 20 423                  0.30149118
## 21 563                  0.49727847
## 22  56                  0.28745266
## 23 528                  0.46332822
## 24 101                  0.17379710
## 25 467                  0.21011898
```

Anhänge

Anhang A

KNN

```
# Vorhersage-Qualitaet: log loss Funktion, d.h unser Bewertungskriterium
# --------------------------------------
# Funktion definieren, die log loss berechnet

train.knn <- read.csv("daten/bloodtrain.csv", header = TRUE)

train.knn <- train.knn %>%
  rename(
    id = "X",
    msld = "Months.since.Last.Donation",
    nod = "Number.of.Donations",
    tvd = "Total.Volume.Donated..c.c..",
    msfd = "Months.since.First.Donation",
    mdim2007 = "Made.Donation.in.March.2007"
  )


log_loss <- function(actual, predicted, eps = 1e-15){
  actual[actual == "yes"] <- 1
  actual[actual == "no"] <- 0
  actual <- as.numeric(actual)
  # Bound probabilities (0,1) for computational purposes
  predicted[predicted < eps] <- eps
  predicted[predicted > 1 - eps] <- 1 - eps
  result=-1/length(actual)*(sum((actual*log(predicted)+(1-actual)*log(1-predicted))))
  return(result)
}

train.knn$mdim2007[train.knn$mdim2007 ==1] <- "yes"
train.knn$mdim2007[train.knn$mdim2007 ==0] <- "no"


# Train KNN algorithm
# --------------------------------------
```

```r
# Anteil fuer Traing-Daten waehlen

split_size = 0.7

# Startwert / seed waehlen --> Reproduzierbarkeit
set.seed(123)

# Initialize data frame of cross-validation log loss
# --------------------------------------
knn_cv_results <- data.frame(matrix(ncol = 6, nrow = 20))
knn_cv_results[,1] <- c(1:20)
colnames(knn_cv_results) <- c("k", "iter1", "iter2", "iter3", "iter4", "iter5")

# Perform repeated cross-validation for KNN to tune K
for (i in 1:20){
  for (j in 1:5){
    # Zufälligen Index für das Auswaheln von Subsamles definieren
    cv_idx <- sample(nrow(train.knn), nrow(train.knn)*split_size, replace = FALSE)

    # Split der Daten in Training-Set und Validation-Set, ID-Spalte weglassen
    cv_tr <- train.knn[cv_idx,-1]
    cv_val <- train.knn[-cv_idx,-1]


    # K festsetzen
    cv_grid <- expand.grid(k = c(i))

    # kNN-Modell trainieren
    knn_cv <- train(as.factor(mdim2007) ~ msfd + msld + nod + mdim2007,
                    data = cv_tr,
                    method = "knn",
                    tuneGrid = cv_grid)

    # Vorhersage machen mit Hilfe des Validierungs-Set
    pred_cv <- predict(knn_cv, cv_val, type = "prob")

    # Resultate festhalten -- i-te Zeile, (j+1). Spalte
    knn_cv_results[i,j+1] <- log_loss(cv_val$mdim2007, pred_cv$yes)
  }
}

# Durchschnittl. log loss fuer jeden Wert von K berechnen
# -----------------------------------------------------------
knn_cv_results$avg_log_loss <- rowSums(knn_cv_results[,2:6])/5

# Ansehen
knn_cv_results$avg_log_loss
```

```
##  [1] 6.8919874 3.6331202 2.7171464 2.1605216 1.7637037 1.6906097 1.5072814
##  [8] 1.0832749 1.2950310 1.0893940 1.2449975 1.2141045 0.8888012 0.7488529
## [15] 0.8751910 0.6452300 0.8845376 0.7219289 0.5365571 0.5249744
```

```r
# Anzeigen
str(knn_cv_results)
```

```
## 'data.frame':    20 obs. of  7 variables:
##  $ k          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ iter1      : num  7.21 2.59 3.82 1.63 1.72 ...
##  $ iter2      : num  7.22 2.65 3.47 2.5 1.74 ...
##  $ iter3      : num  7.41 4.61 2.28 2.46 2.48 ...
##  $ iter4      : num  5.82 5.15 2.31 1.71 1.74 ...
##  $ iter5      : num  6.81 3.16 1.71 2.49 1.14 ...
##  $ avg_log_loss: num  6.89 3.63 2.72 2.16 1.76 ...
```

Anhang B

R Code

```r
knitr::opts_chunk$set(echo = FALSE)

## Bliotheken laden

library(caret)
library(readr)
library(dplyr)
library(corrplot)
library(caret)
library(randomForest)
data_tst <- read.csv("daten/bloodtrain.csv", header = TRUE)
data_prd <- read.csv("daten/bloodtest.csv", header = TRUE)
# Struktur anzeigen
str(data_tst)
dim(data_tst)
# Tabelle Anzeigen
str(data_prd)
dim(data_prd)

data_tst <- data_tst %>%
  rename(
    id = "X",
    msld = "Months.since.Last.Donation",
    nod = "Number.of.Donations",
    tvd = "Total.Volume.Donated..c.c..",
    msfd = "Months.since.First.Donation",
    mdim07 = "Made.Donation.in.March.2007"
  )

str(data_tst)

# Auf fehlende "N/A" Werte prüfen
na_tst <- sapply(data_tst,function(x) sum(is.na(x)))
na_prd <- sapply(data_prd,function(x) sum(is.na(x)))
print(na_tst)
print(na_prd)
# Auf fehlende " " Werte prüfen
na_tst <- sapply(data_tst,function(x) sum(x==""))
na_prd <- sapply(data_prd,function(x) sum(x==""))
```

```r
print(na_tst)
print(na_prd)
#Sollte es Nullwerte haben könnte man die Imputation anwenden (Beispiel)
if(na_tst > 0){
print("NULLWERT!!!!!!!")
preproc_df = preProcess(df, method = "bagImpute")
df <- predict(preproc_df, df)
}
# Daten auf doppelte Zeilen überprüfen
data_tst[duplicated(data_tst),]
data_new <- data_tst[duplicated(data_tst)==FALSE,]
#### ueberpruefen
dim(data_new)
data_new
print(sort(data_new[,1]))


## Histogram mit der Trendline der Trainingsdaten
panel.hist <- function(x, ...)
{
    usr <- par("usr"); on.exit(par(usr))
    par(usr = c(usr[1:2], 0, 1.5) )
    h <- hist(x, plot = FALSE)
    breaks <- h$breaks; nB <- length(breaks)
    y <- h$counts; y <- y/max(y)
    rect(breaks[-nB], 0, breaks[-1], y, col = "cyan", ...)
}

pairs(data_tst[1:5], panel = panel.smooth,
      cex = 1.0, pch = 22, bg = "light blue",
      diag.panel = panel.hist, cex.labels = 2, font.labels = 2,  main="Scatterplots der Trainingsdaten")

summary(data_tst)

## Histogram mit der Trendline der Testdaten

panel.hist <- function(x, ...)
{
    usr <- par("usr"); on.exit(par(usr))
    par(usr = c(usr[1:2], 0, 1.5) )
    h <- hist(x, plot = FALSE)
    breaks <- h$breaks; nB <- length(breaks)
    y <- h$counts; y <- y/max(y)
    rect(breaks[-nB], 0, breaks[-1], y, col = "cyan", ...)
}
pairs(data_prd[1:5], panel = panel.smooth,
      cex = 1.0, pch = 22, bg = "light blue",
      diag.panel = panel.hist, cex.labels = 2, font.labels = 2, main="Histogramme der Testdaten")

summary(data_prd)

# Daten in Trainings- und Testdaten aufteilen
```

```
partition <- createDataPartition(data_tst[,1], times = 1, p = 0.75,list = FALSE)
train <- data_tst[partition,] # Trainings-Daten
validate  <- data_tst[-partition,] # Test-Daten

dim(train)
dim(validate)


### Histogramme anzeigen

par(mfrow=c(2,2))
for(i in 2:5) {
    hist(train[,i], main=names(train)[i])
}

par(mfrow=c(2,2))
for(i in 2:5) {
  dta_A <- density(train[ ,i], na.rm = TRUE)
  dta_B <- density(validate[ ,i], na.rm = TRUE)
  plot(dta_A, col = "blue", main=names(train)[i])
  lines(dta_B, col = "red")
  # plot(density(train[,i]), main=names(train)[i])
}

jittered_x <- sapply(train[,2:5], jitter)
pairs(jittered_x, names(train[,2:5]), col=(train$mdim07)+1)

  cor(train[2:5])
  M <- cor(train)
  corrplot.mixed(M)
  cor(train$nod,train$msfd)

# Variable "tvd" entfernen

useless <- c("tvd")
train <- train[,!(names(train) %in% useless)]
validate <- validate[,!(names(validate) %in% useless)]
str(train)
str(validate)

# Variable "mdim07" in Faktor umwandeln

req_labels <- train['mdim07']
rec_labels <- recode(req_labels$mdim07,'0' = "No", '1' = "Yes")
train$mdim07 <- rec_labels
train$mdim07  <-as.factor(train$mdim07)

str(train)

# Standartwerte setzen

trainControl <- trainControl(method="repeatedcv", summaryFunction=mnLogLoss, number=10, repeats=3, class
```

```
metric <- "logLoss"

# Logistische Regressionen

set.seed(101)
fit.glm <- train(mdim07~., data=train, method="glm", metric=metric, trControl=trainControl) # GLM

set.seed(101)
fit.lda <- train(mdim07~., data=train, method="lda", metric=metric, trControl=trainControl) # LDA

set.seed(101)
fit.glmnet <- train(mdim07~., data=train, method="glmnet", metric=metric,trControl=trainControl)  # GLM

set.seed(101)
fit.cart <- train(mdim07~., data=train, method="rpart", metric=metric,trControl=trainControl)  # CART

set.seed(101)
fit.svm <- train(mdim07~., data=train, method="svmRadial", metric=metric, trControl=trainControl)  # SV

# Auswertung

results <- resamples(list(LG=fit.glm, LDA=fit.lda, GLMNET=fit.glmnet, CART=fit.cart, SVM=fit.svm))
summary(results)
dotplot(results)


# Standartwerte und BoxCox setzen

trainControl <- trainControl(method="repeatedcv", summaryFunction=mnLogLoss, number=10, repeats=3, class

preProcess="BoxCox"

metric <- "logLoss"

# Logistische Regressionen

set.seed(101)
fit.glm <- train(mdim07~., data=train, method="glm", metric=metric, trControl=trainControl, preProc=preP

set.seed(101)
fit.lda <- train(mdim07~., data=train, method="lda", metric=metric, trControl=trainControl, preProc=preP

set.seed(101)
fit.glmnet <- train(mdim07~., data=train, method="glmnet", metric=metric,trControl=trainControl, preProc

set.seed(101)
fit.cart <- train(mdim07~., data=train, method="rpart", metric=metric,trControl=trainControl, preProc=pr

set.seed(101)
fit.svm <- train(mdim07~., data=train, method="svmRadial", metric=metric, trControl=trainControl, prePro

# "CoxBox" Optimierung anhand GLMNET
```

```
print(fit.glmnet)


# Auswertung

results <- resamples(list(LG=fit.glm, LDA=fit.lda, GLMNET=fit.glmnet, CART=fit.cart, SVM=fit.svm))
summary(results)
dotplot(results)


trainControl <- trainControl(method="repeatedcv", summaryFunction=mnLogLoss, number=10, repeats=3, class
metric <- "logLoss"
preProcess = "BoxCox"

set.seed(101)
fit.rf <- train(mdim07~., data=train, method="rf", metric=metric, preProc=preProcess, trControl=trainCon


set.seed(101)
fit.gbm <- train(mdim07~., data=train, method="gbm", metric=metric, preProc=preProcess,
                 trControl=trainControl, verbose=FALSE)  # Gradient Boosting Machine

set.seed(101)
fit.c50 <- train(mdim07~., data=train, method="C5.0", metric=metric, preProc=preProcess,
                 trControl=trainControl)  # C5.0

# Resultate

ensembleResults <- resamples(list(RF=fit.rf, GBM=fit.gbm, C50=fit.c50))
summary(ensembleResults)
dotplot(ensembleResults)


# Variable "mdim07" in Faktor umwandeln

req_labels <- validate['mdim07']
rec_labels <- recode(req_labels$mdim07,'0' = "No", '1' = "Yes")
validate$mdim07 <- rec_labels
validate$mdim07  <-as.factor(validate$mdim07)

str(validate)

# GLMNET mit dem "validate-Datenset"

set.seed(101)
test.pred <- predict(fit.glmnet, newdata=validate, type  = "prob") # GLMNET



# Auswertung

# logLoss kalkulieren
LogLoss <- function(actual, predicted, eps=0.00001) {
```

```r
  predicted <- pmin(pmax(predicted, eps), 1-eps)
  -1/length(actual)*(sum(actual*log(predicted)+(1-actual)*log(1-predicted)))
}

# Labels wieder in "0" und "1" ändern
req_labels <- validate['mdim07']
rec_labels <- recode(req_labels$mdim07,  "No" = '0' , "Yes" = '1')
validate$mdim07 <- rec_labels

# LogLoss bestimmen
log.loss <- LogLoss(as.numeric(as.character(validate$mdim07)), test.pred$Yes)
print(log.loss)


# Spaltennamen anpassen

data_prd <- data_prd %>%
    rename(
    id = "X",
    msld = "Months.since.Last.Donation",
    nod = "Number.of.Donations",
    tvd = "Total.Volume.Donated..c.c..",
    msfd = "Months.since.First.Donation",
    )

str(data_prd)

# Vorhersage durchführen
set.seed(101)
predictions <- predict(fit.glmnet, newdata=data_prd, type  = "prob")

# Submissions-Datei einlesen und Daten abfüllen.

submission_format <- read.csv("daten/submission_format.csv", check.names=FALSE)
submission_format <- submission_format[,-2] # Bestehende "Did Donation" entfernen
pred.df <- as.data.frame(predictions$Yes) #Vorhersagen in DataFrame umwandeln
submission_format <- cbind(submission_format, pred.df) # Vorhersage anhängen

submission_format <- submission_format %>% # Spalten umbennenen
    rename(
    ID = "submission_format",
    'Made Donation in March 2007' = "predictions$Yes",
    )

write.csv(submission_format, file="daten/submission_final.csv", row.names=FALSE ) #CSV-Datei erstellen


# Submissions-Datei anzeigen.

head(submission_format, n = 25L)


# Vorhersage-Qualitaet: log loss Funktion, d.h unser Bewertungskriterium
```

```r
# ---------------------------------------
# Funktion definieren, die log loss berechnet

train.knn <- read.csv("daten/bloodtrain.csv", header = TRUE)

train.knn <- train.knn %>%
  rename(
    id = "X",
    msld = "Months.since.Last.Donation",
    nod = "Number.of.Donations",
    tvd = "Total.Volume.Donated..c.c..",
    msfd = "Months.since.First.Donation",
    mdim2007 = "Made.Donation.in.March.2007"
  )


log_loss <- function(actual, predicted, eps = 1e-15){
  actual[actual == "yes"] <- 1
  actual[actual == "no"] <- 0
  actual <- as.numeric(actual)
  # Bound probabilities (0,1) for computational purposes
  predicted[predicted < eps] <- eps
  predicted[predicted > 1 - eps] <- 1 - eps
  result=-1/length(actual)*(sum((actual*log(predicted)+(1-actual)*log(1-predicted))))
  return(result)
}

train.knn$mdim2007[train.knn$mdim2007 ==1] <- "yes"
train.knn$mdim2007[train.knn$mdim2007 ==0] <- "no"


# Train KNN algorithm
# ---------------------------------------
# Anteil fuer Traing-Daten waehlen

split_size = 0.7

# Startwert / seed waehlen --> Reproduzierbarkeit
set.seed(123)

# Initialize data frame of cross-validation log loss
# ---------------------------------------
knn_cv_results <- data.frame(matrix(ncol = 6, nrow = 20))
knn_cv_results[,1] <- c(1:20)
colnames(knn_cv_results) <- c("k", "iter1", "iter2", "iter3", "iter4", "iter5")

# Perform repeated cross-validation for KNN to tune K
for (i in 1:20){
  for (j in 1:5){
    # Zufälligen Index für das Auswaheln von Subsamles definieren
    cv_idx <- sample(nrow(train.knn), nrow(train.knn)*split_size, replace = FALSE)

    # Split der Daten in Training-Set und Validation-Set, ID-Spalte weglassen
```

```r
    cv_tr <- train.knn[cv_idx,-1]
    cv_val <- train.knn[-cv_idx,-1]


    # K festsetzen
    cv_grid <- expand.grid(k = c(i))

    # kNN-Modell trainieren
    knn_cv <- train(as.factor(mdim2007) ~ msfd + msld + nod + mdim2007,
                    data = cv_tr,
                    method = "knn",
                    tuneGrid = cv_grid)

    # Vorhersage machen mit Hilfe des Validierungs-Set
    pred_cv <- predict(knn_cv, cv_val, type = "prob")

    # Resultate festhalten -- i-te Zeile, (j+1). Spalte
    knn_cv_results[i,j+1] <- log_loss(cv_val$mdim2007, pred_cv$yes)
  }
}

# Durchschnittl. log loss fuer jeden Wert von K berechnen
# -----------------------------------------------------------
knn_cv_results$avg_log_loss <- rowSums(knn_cv_results[,2:6])/5

# Ansehen
knn_cv_results$avg_log_loss

# Anzeigen
str(knn_cv_results)
```