Humboldt-Universität zu Berlin
School of Business and Economics
Chair of Information Systems
Applied Predictive Analytics

Data Mining Cup 2015: Basket Value Prediction

Submitted By:
Elisabeth Bommes
Sophie Burgard
Ringolf Thomschke

Submitted To:
Prof. Dr. Stefan Lessmann

Summer term 2015

Contents

1 Introduction

Prudsys AG staged the 2015 Data Mining Cup for six weeks from April 7 to May 19 2015. We participated as part of a seminar on "Applied Predictive Analytics" and out of 188 teams from a total of 153 universities from 48 countries our two submissions ranked first and sixth place. We participated in the 2015 Data Mining Cup sponsored by prudsys AG and ranked first place. For the 2015 Data Mining Cup, prudsys AG provided customer data from an online shop. From this data, the contestants had to predict the propensity of customers to redeem a coupon and the customers' basket value. After the release of the training and test data, the challenge was to create and submit predictions on the test data within six weeks. We divided the challenge into several different tasks which up to four team members were assigned to. In this report, we will describe the prediction of the basket value variable.

2 Data

The data set consisted of 6053 instances for which 33 variables were given. The distribution of the target variable basket value turned out to be highly right-skewed with a very large range of over 135000 and showed over 237 of strong extremes (by definition in the boxplot concept). Furthermore, it showed significant peaks at lower levels which, however, remained undescribed in our further analysis. Finding an appropriate algorithm to detect outliers in the basket value variable, was another important task. This will also not be covered by this report as it was assigned to other team members. The idea is to replace outliers by the customer's mean basket value or, if not available, by the overall mean. In our analysis, we decided to use a rather arbitrary upper fence of 700 for outliers which had been set by the data preparation group. based on the definition of strong extremes in the boxplot and a look at the histogram.

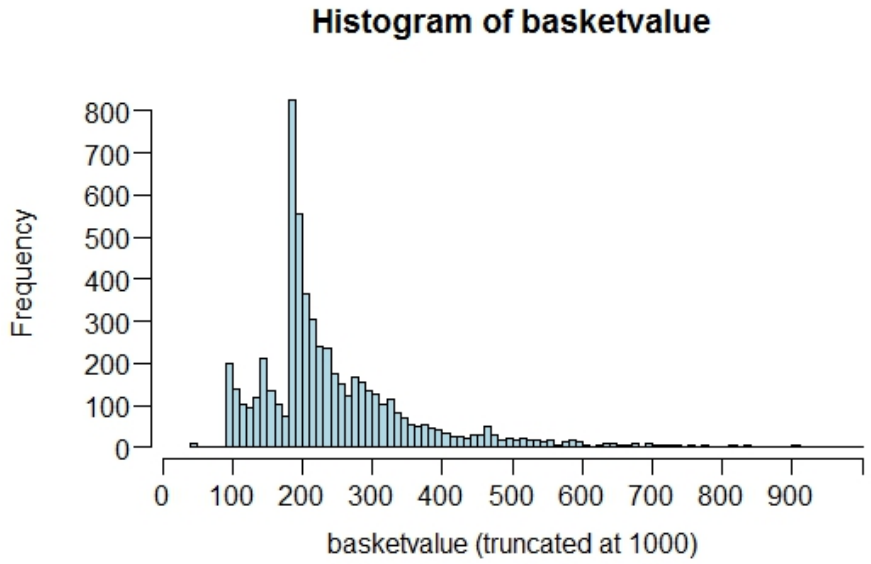

Figure 1: Distribution of the basket value variable

3 Competition Outcomes

3.1 Base Model Selection

In order to create a library of base classifiers, we used several learning methods in `R` such as random forests, support vector machines or gradient boosting machines. We estimated a total of 16 different base classifiers on the data set which, after some thoughtful variable building, consisted of 100 variables for each observation. The choice of the specific models is orientated at Delagado (2014), stating that various kinds of random forest models usually perform best, followed by some support vector machines.[1]The models were trained on randomly chosen 80 % of the given data. The remaining 20 % served as hold-out sample to test, validate and compare the base classifiers on unseen data.

The model estimation and parameter tuning was performed using the `R`-package `caret` (short for *C*lassification *a*nd *re*gression *t*raining). A five-fold cross-validation was implemented to determine the model parameters. The models were trained to minimize the following by the competition given assessment criterion on the data:

Model	AC Holdout-Sample (n=1210)	AC Test-Set (n=669)
UserID-Model	2486	1167
`cubist`	3355	1741
Ensemble	3390	1766
`parRF`	3466	1817
`RRF`	3467	1816
`RRFglobal`	3467	1816
`rf`	3468	1815
`qrf`	3472	1817
`ctree`	3475	1816
`cforest`	3485	1824
`gbm`	3469	1825
`blackboost`	3506	1830
`mlp`	3742	1971
`svmRadialCost`	3786	1995
`svmRadial`	3742	1995
`mlpWeightDecay`	3832	2020

Table 1: Evaluation of Base Classifiers (without obvious misclassifiers (AC over 100 000)

$$AC = \sum_{i=1}^{n} \frac{|basketValue_i - basketValuePrediction_i|}{\frac{1}{n}\sum_{j=1}^{n} basketValue_i}$$

The respective value of the above standing criterion for different base classifiers is listed in Table 1, "Model" refers hereby to the corresponding model name in the `R`-package `caret`.

Most of the models aim towards a relatively similiar prediction power. But the smallest prediction error was made by a more insightful model which predicted the basket value variable by taking the mean of previous basket values for known customers and predicting for unknown customers over some random forest model. We suspect the inferiority of the machine learning under a much more simpler model being caused by the drawing mechanism of the 20% hold-out sample and will investigate this issue in a later part of this report.

model	qrf	cubist	parRF	rf	RRF	blackboost
weight	0.05	0.6	0.2	0.0985	0.05	0.0015

Table 2: Models Within The Ensemble And Corresponding Weights

3.2 Ensemble Selection

To combine the various model predictions to one single value for each data point, we used the technique of ensemble selection.[2] Ensemble selection aims for a superior prediction power by determining a weighted mean of the predicted values made by different base classifiers. Idea behind that is to overcome shortcomings of various model types. For the given data set the algorithm determined weights according to Table 2. The overall prediction error for the ensemble with the given weights on the hold-out sample is 3390, therefore performing second best after the best base classifier (`cubist` 3355).

4 Post-Processing

In the following section we are going to investigate the reasons for the inferior performance of the machine learning techniques in estimating the basket value variable via a simulation study. In line with the simulation stood the focus on the post editing of the programming code, especially parallelization issues to shorten the needed calculation time.

One reason for the bad performance of any machine learning technique we suspect in the mechanism how the separation between holdout- and training-sample was made: The 20% test set was randomly chosen in the process of predicting the first of the binary coupon variables via an in the `caret`-package included function `createDataPartition` and using the same sample for all other target variables. By using this function the sampling mechanism aims to maintain the distribution of the target variable, in this case of one of the coupons. We suspect that this way of sampling leads to an insufficient distribution of the basket value variable in the hold-out sample and therefore yielding deficient prediction results.

mean	median	sd	min	max	range
56153	11033	69616	404	237538	237134

Table 3: Summary statistics for assessment criteria of the 98 ensembles on hold-out sample

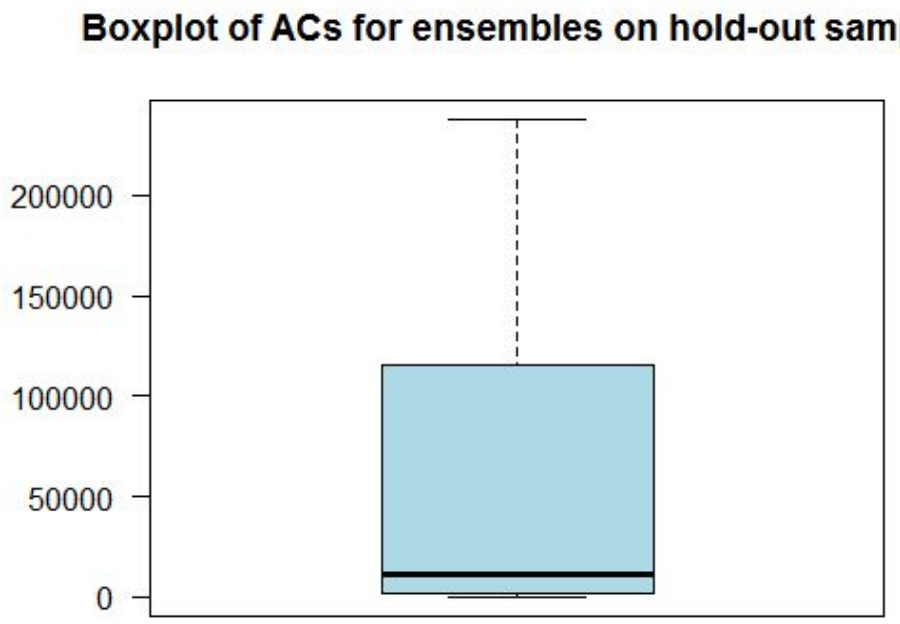

Figure 2: Boxplot of the assessment criteria for the 98 ensembles on hold-out sample

By taking a look at the boxplot (Fig. 2) and the summary statistics (Tab. 3) for the 98 ensembles' assessment criteria, we notice the very large range. Also, the median is higher than any AC from the competition. One can conclude from these results that partitioning the data with respect to the target variables distribution does not, in general, lead to lower results for the assessment criterion. Presumably, this is due to the fact that the DMC metric allocates a very high penalty to mispredicted outliers. The number of outliers in the hold-out sample can increase the AC severely as the difference between observation and prediction is squared. However, as mentioned above, the target variable shows a large range of strong extremes and, therefore, the machine learning techniques seem unable to prevent stark mispredictions.

Indeed, the simulation study produced a high number of ensembles which are inferior to those built during the competition. Yet, we were now able to choose from 98 samples of base classifier and ensembles. The minimal ACs of these sample were all lower than the ones collected during the competition (Tab. 4). This shows the

Model	Competition AC	Simulation study AC
Ensemble	3390	404
`RRF`	3467	485
`RRFglobal`	3467	700
`rf`	3468	660
`qrf`	3472	755
`ctree`	3475	670
`cforest`	3485	957
`gbm`	3469	1892
`blackboost`	3506	919
`mlp`	3742	836
`svmRadialCost`	3786	819
`svmRadial`	3742	820
`mlpWeightDecay`	3832	834

Table 4: Comparison of minimal assessment criteria collected in the competition and in the simulation study.

advantage of having a large amount of base classifiers that were trained on differently partitioned data sets. One can decrease the probability of having a highly skewed hold-out sample by simply creating more randomly partitioned data sets on which the base classifiers are trained.

After the competition was finished, prudsys AG provided target variable values which now enabled us to examine the performance of our ensembles on the validation set. By taking a look on the upper part of Figure 3, we observe that, only two ensemble perform worse than the mean prediction on the validation set. Other observations lie rather symmetrically distributed around the AC mean. Looking at the bottom part of Figure 3, we realize that the ensemble with the lowest score on the hold-out sample has an average performance on the validation set. Other ensemble with slightly higher AC on their hold-out sample show better results on the validation set. Nevertheless, the very low scores on the validation set can be found in the region up to an AC value of 3500 for the hold-out sample. Therefore, to choose an ensemble with good performance on the hold-out sample is still reasonable.

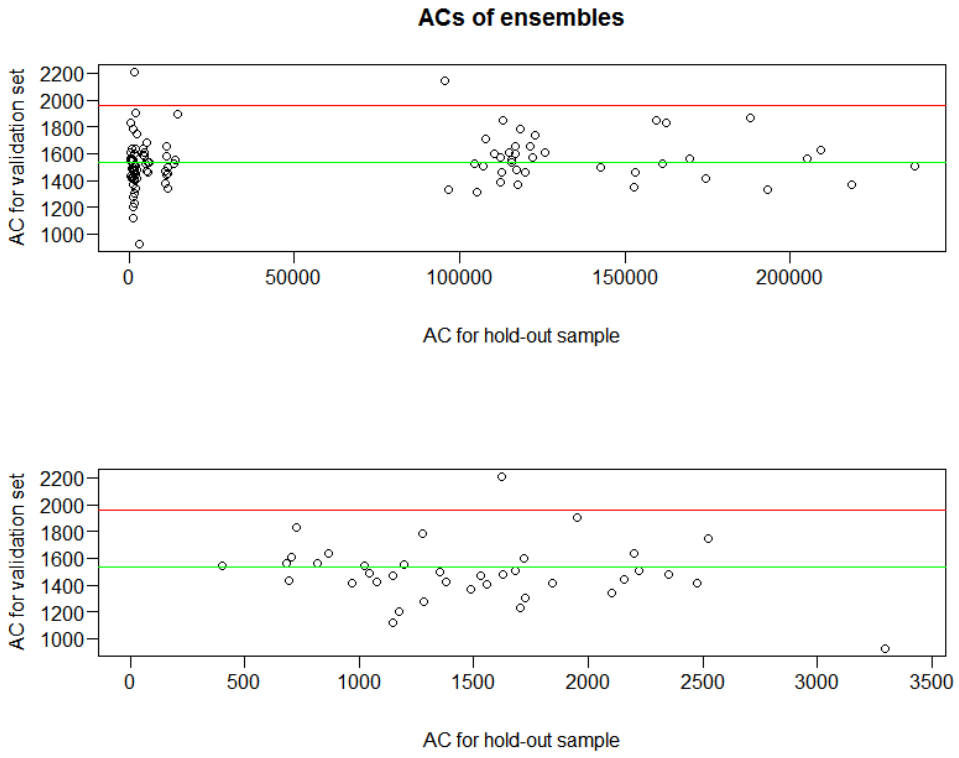

Figure 3: Scatterplot of ACs of ensembles for hold-out sample and validation set. (Top) the entire range of ACs for hold-out sample, (bottom) truncated at 3500. Red lines indicate the AC of a mean prediction, green lines indicate the average AC on the validation set.

5 Summary

By conducting a simulation study for the prediction of basket values with ensemble selection, we gained further insight in the importance of partitioning and outlyingness in the context of machine learning techniques.

Our first assumption was that the inferiority of machine learning models compared to the straight-forward customer mean prediction was caused by an undue partitioning of the data which did not take the target variable distribution into account. This assumption could not be verified most likely because of the very large range of outliers. However, we still consider this technique as the correct approach even if it cannot be substantiated by our result.

More importantly, with our simulation study we produced superior results for the assessment criteria for all base classifiers and in the ensemble selection. This was possible due to the mere amount of models we simulated on different randomly

partitioned data sets. By using differently partitioned data sets for training and testing we decreased the probability of drawing highly skewed hold-out samples which, in general, yield bad performance meassures.

A final evaluation on the validation set with observations provided after the competition allowed for further examination of the estimated ensembles.

A Appendix

R Code

- Data Preparation.R
- Model Estimation.R
- Prediction Extraction.R
- Test Prediction.R
- Ensemble Selection.R
- Results.R

```
#########################################################
################## Data Preparation #####################
#########################################################
# The original data preparation file written by
# members of the data preparation task group.
# In the context of basket value prediction,
# adjustments were made only in order to partition
# the data according to the distribution of the
# basket value variable.
# This file is called by the Model Estimation
# procedure.
#########################################################

#options(stringsAsFactors = FALSE)

pack = c("caret", "data.table", "splitstackshape",
"stats", "arules", "klaR")


lapply(pack, function(x) if (!(x %in%
installed.packages())) {
install.packages(x)
})

lapply(pack, require, character.only = TRUE)

setwd(path.data)



train <- read.table("DMC_2015_orders_train.txt",
header=TRUE, sep="|", stringsAsFactors=FALSE)
test <- read.table("DMC_2015_orders_class.txt",
header=TRUE, sep="|",stringsAsFactors=FALSE)

data.part  =
createDataPartition(y = train$basketValue,
p = 0.8, list = FALSE)
data.part  = as.vector(data.part)
```

```
data.train = train[  data.part, ]
indi       = c(1:dim(train)[1])
data.test  = train[! indi %in% data.part, ]

data.train$trainingSetIndex = 1
data.test$trainingSetIndex  = 0
test$trainingSetIndex       = -1

total <- rbind(data.train, data.test, test)

setwd(path.source)
# brand "" as "unknown"
total$brand1[total$brand1==""] <- "unknown"
total$brand2[total$brand2==""] <- "unknown"
total$brand3[total$brand3==""] <- "unknown"

# prepare time variables
# lag time since last buy

total <- data.table(total)
total[, lastBuy := c(NA, orderTime[-.N]),
by = userID]
total[, lastBasketValue := c(NA,
basketValue[-.N]), by = userID]
total <- as.data.frame(total)

# format variables as time with strptime
total$orderTime <-
strptime(total$orderTime,
format="%Y-%m-%d␣%H:%M:%S")
total$couponsReceived <- strptime(
total$couponsReceived,
format="%Y-%m-%d␣%H:%M:%S")
total$lastBuy <-
strptime(total$lastBuy,
format="%Y-%m-%d␣%H:%M:%S")
# extract information on weekday and time of day
total$orderWeekday <- weekdays(total$orderTime)
total$receivedWeekday <-
weekdays(total$couponsReceived)
```

```
total$orderDaytime <- total$orderTime$hour
total$receivedDaytime <-
total$couponsReceived$hour
total$orderedDayOfMonth <- total$orderTime$mday
# calculate the time difference between
# receiving and using
total$timeDifference <-
as.numeric(difftime(total$orderTime,
total$couponsReceived, units="mins"))

total$timeDiffLastBuy <-
as.numeric(difftime(total$orderTime,
total$lastBuy, units="days"))
total$timeDiffLastBuy <-
as.character(cut(total$timeDiffLastBuy,
breaks=c(0:21,seq(22,50,2),68),
labels=c(paste("vor",c(0:21,seq(22,49,2),68),
"Tagen",sep=""))))
total$timeDiffLastBuy[is.na(
total$timeDiffLastBuy)] <- "unknown"

# calculate consumption time
total$lastBasketConsumption <-
total$lastBasketValue/as.numeric(difftime(
total$orderTime,total$lastBuy,units="days"))

# dummy for multiple order with same coupon package
total$multiPurchases<-
as.numeric(duplicated(total$userID)=="TRUE" &
duplicated(total$couponsReceived)=="TRUE")

# time difference categories
total$timeDifferenceCategories<-
cut(total$timeDifference,breaks=c(seq(0,50,10),
seq(55,600,300),seq(800,5000,1000),
seq(6000,11000,2500)),
labels=c(paste("difference",
c(seq(0,50,10),
seq(55,600,300),
seq(800,5000,1000),
```

```
seq(6000,8500,2500)),sep="")))

# calculate several combinations of the price
# variables
total$priceRatio1 <- total$basePrice1/total$price1
total$sumPrice1 <- total$price1+total$basePrice1
total$rewardPrice1 <- total$price1 * total$reward1
total$rewardBasePrice1 <- total$basePrice1 *
total$reward1
total$priceRatio2 <- total$basePrice2/total$price2
total$sumPrice2 <- total$price2+total$basePrice2
total$rewardPrice2 <- total$price2 * total$reward2
total$rewardBasePrice2 <- total$basePrice2 *
total$reward2
total$priceRatio3 <- total$basePrice3/total$price3
total$sumPrice3 <- total$price3+total$basePrice3
total$rewardPrice3 <- total$price3 * total$reward3
total$rewardBasePrice3 <- total$basePrice3 *
total$reward3

# Create a productID
total$productID1 <-
        paste(total$price1,total$basePrice1,
        total$productGroup1, total$categoryIDs1,sep="")
total$productID1 <-
        paste("product",as.numeric(factor(
    total$productID1)))
total$productID2 <-
        paste(total$price2,total$basePrice2,
    total$productGroup2, total$categoryIDs2,sep="")
total$productID2 <-
        paste("product",as.numeric(factor(
    total$productID2)))
total$productID3 <-
        paste(total$price3,total$basePrice3,
    total$productGroup3, total$categoryIDs3,sep="")
total$productID3 <-
        paste("product",as.numeric(factor(
    total$productID3)))
```

```
total$maxPrice <-
        max(total$price1,total$price2,total$price3)
total$minPrice <-
        min(total$price1,total$price2,total$price3)
total$priceSpread <- total$maxPrice-total$minPrice
total$sumPrices <- total$price1+total$price2+
                                        total$price3
total$sumBasePrices <-
        total$basePrice1+total$basePrice2+
    total$basePrice3

# sum of premium product coupons
total$sumPremium <- total$premiumProduct1 +
total$premiumProduct2 + total$premiumProduct3

# Calculate how often each unique customer is in
# the train dataset
# total<- total[order(total$userID),]
shoppingFreq <- as.data.frame(table(
        total$userID))
colnames(shoppingFreq) <- c("userID",
        "shoppingFreq")
total<- merge(total,shoppingFreq, by="userID")


# calculate same brands, etc. for each 1,2,3
# separately
total$equalOtherBrands1 <-
        0 +
        (total$brand1==total$brand2) +
    (total$brand1==total$brand3)
total$equalOtherBrands2 <-
        0 + (total$brand2==total$brand1) +
    (total$brand2==total$brand3)
total$equalOtherBrands3 <-
        0 + (total$brand3==total$brand1) +
    (total$brand3==total$brand2)
total$equalOtherProductGroup1 <- 0 +
(total$productGroup1==total$productGroup2) +
(total$productGroup1==total$productGroup3)
```

```
total$equalOtherProductGroup2 <- 0 +
(total$productGroup2==total$productGroup1) +
(total$productGroup2==total$productGroup3)
total$equalOtherProductGroup3 <- 0 +
(total$productGroup3==total$productGroup1) +
(total$productGroup3==total$productGroup2)

############################


dmcTrain <- subset(total,
total$trainingSetIndex==1)
# calculate user specific max basket value
# (base rate if no information on user)
couponRateFrame <-
as.data.frame(tapply(dmcTrain$basketValue,
dmcTrain$userID, function(x)max(x,na.rm=TRUE)))
couponRateFrame$userID <- rownames(couponRateFrame)
rownames(couponRateFrame) <- NULL
colnames(couponRateFrame) <- c("userMaxBasketValue",
"userID")
couponRateFrame$userMaxBasketValue[
is.infinite(couponRateFrame$userMaxBasketValue)] <-
median(couponRateFrame$userMaxBasketValue,
na.rm=TRUE)
total <- merge(total,couponRateFrame,by="userID",
all.x=TRUE)
total$userMaxBasketValue[is.na(
total$userMaxBasketValue)] <-
median(couponRateFrame$userMaxBasketValue,
na.rm=TRUE)

# calculate user specific min basket value
# (base rate if no information on user)
couponRateFrame <-
as.data.frame(tapply(dmcTrain$basketValue,
dmcTrain$userID, function(x)min(x,na.rm=TRUE)))
couponRateFrame$userID <- rownames(couponRateFrame)
rownames(couponRateFrame) <- NULL
colnames(couponRateFrame) <- c("userMinBasketValue",
```

```
"userID")
couponRateFrame$userMinBasketValue[
is.infinite(couponRateFrame$userMinBasketValue)] <-
median(couponRateFrame$userMinBasketValue,
na.rm=TRUE)
total <- merge(total,couponRateFrame,by="userID",
all.x=TRUE)
total$userMinBasketValue[is.na(
total$userMinBasketValue)] <-
median(couponRateFrame$userMinBasketValue,
na.rm=TRUE)

# calculate user specific basket value variance
# (base rate if no information on user)
couponRateFrame <-
as.data.frame(tapply(dmcTrain$basketValue,
dmcTrain$userID, function(x)var(x,na.rm=TRUE)))
couponRateFrame$userID <- rownames(couponRateFrame)
rownames(couponRateFrame) <- NULL
colnames(couponRateFrame) <- c("userBasketValueVar",
"userID")
couponRateFrame$userBasketValueVar[is.na(
couponRateFrame$userBasketValueVar)] <- 0
total <- merge(total,couponRateFrame,by="userID",
all.x=TRUE)
total$userBasketValueVar[is.na(
total$userBasketValueVar)] <- 0

source("merge_over_coupons.R")

# cluster data for both datasets
source("cluster.R")
#
dmcTrain <- subset(total,total$trainingSetIndex==1 &
total$basketValue<700)
mergedTrain <- subset(merged,
merged$trainingSetIndex==1 &
total$basketValue<700)
# cluster specific median basketValue
for (cluster in grep("Cluster",
```

```
colnames(merged),value=TRUE)){
basketValueFrame <-
as.data.frame(tapply(mergedTrain$basketValue,
mergedTrain[, cluster],
function(x)median(x,na.rm=TRUE)))
basketValueFrame$shoppingFreq <-
rownames(basketValueFrame)
rownames(basketValueFrame) <- NULL
colnames(basketValueFrame) <-
c(paste("baseline",sub("Cluster","Clust",cluster),
sep=""), as.character(cluster))
basketValueFrame[is.nan(basketValueFrame[,
paste("baseline",sub("Cluster","Clust",cluster),
sep="")]),paste("baseline",
sub("Cluster","Clust",cluster),sep="")] <-
median(mergedTrain$basketValue, na.rm=TRUE)
merged <-
merge(merged,basketValueFrame,
by=as.character(cluster), all.x=TRUE)
total<- merge(total,basketValueFrame,
by=as.character(cluster),all.x=TRUE)
}

# calculate user specific coupon rate
# (base rate if no information on user)
couponRateFrame <-
as.data.frame(tapply(dmcTrain$coupon1Used,
dmcTrain$userID, function(x)mean(x,na.rm=TRUE)))
couponRateFrame$userID <- rownames(couponRateFrame)
rownames(couponRateFrame) <- NULL
colnames(couponRateFrame) <- c("userCouponRate1",
"userID")
#couponRateFrame$userCouponRate1[
is.nan(couponRateFrame$userCouponRate1)] <-
mean(dmcTrain$coupon1Used, na.rm=TRUE)
total <- merge(total,couponRateFrame,by="userID",
all.x=TRUE)
total$userCouponRate1[
is.na(total$userCouponRate1)] <-
mean(dmcTrain$coupon1Used, na.rm=TRUE)
```

```
couponRateFrame <-
as.data.frame(tapply(dmcTrain$coupon2Used,
dmcTrain$userID, function(x)mean(x,na.rm=TRUE)))
couponRateFrame$userID <- rownames(couponRateFrame)
rownames(couponRateFrame) <- NULL
colnames(couponRateFrame) <- c("userCouponRate2",
"userID")
#couponRateFrame$userCouponRate2[
is.nan(couponRateFrame$userCouponRate2)] <-
mean(dmcTrain$coupon2Used, na.rm=TRUE)
total <- merge(total,couponRateFrame,by="userID",
all.x=TRUE)
total$userCouponRate2[
is.na(total$userCouponRate2)] <-
mean(dmcTrain$coupon2Used, na.rm=TRUE)

couponRateFrame <-
as.data.frame(tapply(dmcTrain$coupon3Used,
dmcTrain$userID, function(x)mean(x,na.rm=TRUE)))
couponRateFrame$userID <- rownames(couponRateFrame)
rownames(couponRateFrame) <- NULL
colnames(couponRateFrame) <- c("userCouponRate3",
"userID")
#couponRateFrame$userCouponRate3[
is.nan(couponRateFrame$userCouponRate3)] <-
mean(dmcTrain$coupon3Used, na.rm=TRUE)
total <- merge(total,couponRateFrame,by="userID",
all.x=TRUE)
total$userCouponRate3[
is.na(total$userCouponRate3)] <-
mean(dmcTrain$coupon3Used, na.rm=TRUE)

# baselines for all merged and part total.
source("baselineByPanel.R")

# Basket Value of last buy
total$lastBasketValue[
is.na(total$lastBasketValue)] <-
total$userMedianBasketValue[
```

```
is.na(total$lastBasketValue)]
total$lastBasketValue[
is.na(total$lastBasketValue)] <-
median(dmcTrain$basketValue, na.rm=TRUE)
# total <-
total[order(total$userID,partial=total$orderTime),]
# View(total[,c("userID","orderTime","basketValue",
"lastBasketValue")])
merged$lastBasketValue <- NULL
merged <- merge(merged,total[,c("userID","orderID",
"lastBasketValue")], by=c("userID","orderID"),
all.x=TRUE)
# merged <-merged[order(merged$userID,
partial=merged$orderTime),]
# View(merged[,c("userID","orderTime","basketValue",
"lastBasketValue")])


# Factor variable reduction by weight of evidence
source("weight_of_evidence.R")

# sort by orderID
total <- total[order(total$orderID),]
merged <- merged[order(merged$orderID),]

# remove all high-level factors
total<-total[,!colnames(total) %in%
c('couponsReceived',"orderTime", "couponID")]

merged<-merged[,!colnames(merged) %in%  c("brand",
"productGroup", "categoryIDs",'couponsReceived',
"orderTime","couponID", "productID")]
merged <- merged[,!grepl("categoryIDs_",
colnames(merged))]
merged <- merged[,!grepl("brand_",
colnames(merged))]
merged <- merged[,!grepl("productGroup_",
colnames(merged))]
merged <- merged[,!grepl("productID_",
colnames(merged))]
```

```
# remove lastBuy
total$lastBuy <- NULL
merged$lastBuy <- NULL
total$lastBasketConsumption <- NULL
merged$lastBasketConsumption <- NULL


data = total

### Only keep data
rm(list = setdiff(ls(), c("data", "path",
"path.data", "path.results", "path.source",
"stempel")))

data$userID = NULL
data$lastBuy                    = NULL
data$lastBasketConsumption      = NULL
data$couponNrCouponRateTotal1 = NULL
data$couponNrCouponRateTotal2 = NULL
data$couponNrCouponRateTotal3 = NULL
data$userID                     = NULL
data$coupon1Used                = NULL
data$coupon2Used                = NULL
data$coupon3Used                = NULL
data$orderID                    = NULL

data$orderWeekday = as.factor(data$orderWeekday)
data$receivedWeekday =
as.factor(data$receivedWeekday)
data$timeDiffLastBuy =
as.factor(data$timeDiffLastBuy)

# Remove constant columns

#data[sapply(data, is.character)] <-
lapply(data[sapply(data, is.character)],
as.factor)
```

```
data = data[,sapply(data, function(v) var(v,
na.rm=TRUE)!=0)]
```

```
#########################################################
################## Model Estimation ####################
#########################################################
# This file carries out the model estimation.
# After calling Data Preparation.R, 14 different
# prediction models are trained in parallel on the
# randomly partitioned dataset and then saved to
# "/Estimated Models".
#########################################################

pack1 =
c("earth","elasticnet","leaps","kernlab","ipred",
"plyr","rpart", "kknn", "nnet","brnn", "frbs",
"RSNNS","foreach","caret","gbm","randomForest",
"RRF","party", "quantregForest", "mboost", "Cubist")

pack2 = c("earth", "elasticnet", "leaps", "kernlab",
"ipred", "plyr", "rpart","kknn", "nnet", "brnn",
"frbs", "RSNNS", "foreach", "caret", "gbm",
"randomForest", "RRF", "party", "doParallel",
"data.table","splitstackshape", "stats", "arules",
"klaR", "elasticnet")

pack = unique(c(pack1, pack2))

lapply(pack, function(x) if (!(x %in%
installed.packages())) {
  install.packages(x)
})

lapply(pack, library, character.only = TRUE)

# Set Paths
path         = "H:/Projects/Predictive␣Analytics"
path.data    = paste(path, "/DMC2015", sep = "")
path.source  = paste(path, "/Sources", sep = "")
path.results = paste(path, "/Estimated␣Models",
sep = "")

# Here: Loop
```

```
for(iter in 1:50){
print(iter)
stempel = as.character(Sys.time())
stempel = gsub(":", "", stempel, fixed = TRUE)
stempel = gsub("-", "", stempel, fixed = TRUE)
stempel = gsub("␣", "", stempel, fixed = TRUE)
# Prep data
setwd(path)
print("Prep␣Data")
source("Data␣Preparation.R")
setwd(path.results)
saveRDS(data, file = paste(stempel, "data.rds",
sep = ""))

models = c( "ctree", "blackboost", "RRFglobal",
"mlp", "mlpWeightDecay", "gbm", "qrf", "glmboost",
"cubist", "svmRadialCost", "svmRadial", "RRF",
"rf", "cforest")

# models = c("blackboost", "mlp")

# DMC metric
c.acs = function(data, lev = NULL, model = NULL){
  denom       = mean(data$obs)
  crit        = ( abs(data$obs - data$pred) /
  denom )^2
  crit        = sum(crit)
  names(crit) = "DMC"
  return(crit)
}

# Save model
model.save = function(model, data){
    L = list(model = model, data = data)
    f = list.files(pattern = ".rds")
    if(length(f) != 0){
        f = as.numeric(gsub(".rds", "", f,
        fixed = TRUE))
        i = max(f) + 1
    }else{
```

```
        i = 1
    }

    saveRDS(L, file = paste(i, ".rds", sep = ""))
}

data.train = data[data$trainingSetIndex == 1, ]
c.model    = models
m          = length(c.model)
c.fc       = trainControl(returnData = TRUE,
savePredictions = TRUE, number = 5,
summaryFunction = c.acs)

L = list()
print("Start␣Estimation")

#c.train = foreach(i = 1:m, .packages = pack)
%dopar%{for(i in 1:m){
    print(paste("Estimate␣model", c.model[i]))
    print(paste("␣␣␣␣␣Start␣time", Sys.time()))
    cl = 50
    fl = makeCluster(cl)
    registerDoParallel(fl)
    c.train.m = try(train(basketValue ~ .,
    data = data.train, method = c.model[i],
    trControl = c.fc, tuneLength = 2,
    maximize = FALSE, metric = "DMC"))
    stopCluster(fl)
    print(paste("␣␣␣␣␣End␣time", Sys.time()))
    # L[i] = c.train.m
    saveRDS(c.train.m, file = paste(stempel,
    c.model[i], ".rds", sep = ""))
}
}
```

```
#########################################################
############ Prediction Extraction ######################
#########################################################
# This file loads the pre-calculated models from
# "/Estimated Models" and extracts there predictions
# on the training set. Extracted prediction are
# saved to "/Predictions".
#########################################################

pack =
c("earth","elasticnet","leaps","kernlab","ipred",
"plyr","rpart", "kknn", "nnet","brnn", "frbs",
"RSNNS","foreach","caret","gbm","randomForest",
"RRF","party", "Cubist")

for (p in pack) {
  if (!require(p, character.only = T))
{install.packages(p);
        library(p, character.only = T)}
}
rm(p,pack)


path = "~/Desktop/DMC/Estimated␣Models"
path.output = "~/Desktop/DMC/Predictions"

setwd(path)

#Model list and unique time stamps
files= list.files(path)
tmp_stamp =
as.factor(unique(substr(files, 1, 14 )))

#use only stamps for complete model sets
for(n in 1:length(files)){
    for(i in 1:length(tmp_stamp)){
        freq =
        count(grepl(tmp_stamp[tnp_stamp=i], files))
            if (freq[2,2]==15 & n==1){
              compl =
```

```
              as.character(paste(tmp_stamp[
              tmp_stamp=i]))
            }
            if(freq[2,2]==15){
               compl2 =
               as.character(paste(tmp_stamp[
               tmp_stamp=i]))
               compl = cbind(compl, compl2)}
    }
}
stamp= as.factor(unique(compl[1,]))

models = as.factor(c("blackboost.rds",
"cforest.rds", "ctree.rds","cubist.rds","gbm.rds",
"glmboost.rds","mlp.rds","mlpWeightDecay.rds",
"qrf.rds", "rf.rds","RRF.rds","RRFglobal.rds",
"svmRadial.rds","svmRadialCost.rds"))

setwd(path)

#predictions for models
for (i in 1:nlevels(stamp)){
    ID = stamp[stamp=i]
    data = readRDS(paste(as.character(ID),
    "data.rds", sep = ""))
    train=data[data$trainingSetIndex==1,]

        for (n in 1:length(models)){
             name = paste(as.character(ID),
             as.character(models[[n]]), sep = "")
             readRDS(name)

              if(n == 1 & i==1 ){
                   col = paste(substr(name, 15,
                   nchar(name)-4), i, sep="_")
                   print(paste(Sys.time(), col,
                   sep="␣"))
                   mod = readRDS(name)
                   tmp = data.frame(predict(mod,
                   newdata=train))
```

```
                    tmpcol = as.vector(col)
                }
                else{
                    col = paste(substr(name, 15,
                    nchar(name)-4), i, sep="_")
                    print(paste(Sys.time(), col,
                    sep="␣"))
                    mod = readRDS(name)
                    tmp2 = data.frame(predict(mod,
                    newdata=train))
                    tmp  = cbind(tmp, tmp2)
                    tmpcol2 = as.vector(col)
                    tmpcol = as.vector(c(tmpcol,
                    tmpcol2))
                    colnames(tmp) = tmpcol
                }
        }
}

setwd(path.output)
write.csv2(tmp, "predictions.csv", row.names = FALSE)
```

```
#########################################################
################## Test Prediction ######################
#########################################################
# This file is used to carry out the predictions on
# test and validation set.
# For each trained model of a randomly partitioned
# data set stored in "/Estimated Models", the basket
# value in test and validation set are predicted.
# Finally, for each data set, a list is built which
# includes model predictions on training, test and
# validation set as well es the basket values from
# the DMC data.
#########################################################

pack1 =
c("earth","elasticnet","leaps","kernlab","ipred",
"plyr","rpart", "kknn", "nnet","brnn", "frbs",
"RSNNS","foreach","caret","gbm","randomForest",
"RRF","party", "quantregForest", "mboost", "Cubist")

pack2 = c("earth", "elasticnet", "leaps", "kernlab",
"ipred", "plyr", "rpart", "kknn", "nnet", "brnn",
"frbs", "RSNNS", "foreach", "caret", "gbm",
"randomForest", "RRF", "party", "doParallel",
"data.table","splitstackshape", "stats", "arules",
"klaR", "elasticnet")

pack = unique(c(pack1, pack2))

lapply(pack, function(x) if (!(x %in%
installed.packages())) {
  install.packages(x)
})

lapply(pack, library, character.only = TRUE)

models =
c("blackboost", "cforest", "ctree", "cubist", "gbm",
"glmboost", "mlp", "mlpWeightDecay","qrf", "rf",
"RRF","RRFglobal", "svmRadial","svmRadialCost")
```

```
ex = c("set.dat.train", "set.dat.test",
"set.dat.val", "models")


# Set Paths
path         = "/Volumes/bommesel.hub/Projects/
Predictive␣Analytics"
path.data    = paste(path, "/DMC2015", sep = "")
path.source  = paste(path, "/Sources", sep = "")
path.results = paste(path, "/Estimated␣Models",
sep = "")
path.pred    = paste(path, "/Predictions", sep = "")

# Basket values validation set
setwd(path.data)
bv.val = read.csv("basketvalues.csv", sep = ";",
dec = ",")
bv.val = bv.val$x


# Model list and unique time stamps
set.est   = list.files(path.results)
set.ind   =
summary(as.factor(substr(set.est, 1, 14 ))) == 15
set.compl = names(set.ind[set.ind == TRUE])

# Don't extract if we already have it
set.done  = list.files(path.pred)
set.done  = substr(set.done, 1, 14)
set.compl = set.compl[!set.compl %in% set.done]


fl = detectCores()
### Loop over completed estimations
for(i in 2:length(set.compl)){
        setwd(path.results)
        set           = set.compl[i]
        print(set)
        set.dat       =
```

```
readRDS(paste(set, "data.rds", sep = ""))
    set.dat.train =
set.dat[set.dat$trainingSetIndex ==  1, ]
    set.dat.test  =
set.dat[set.dat$trainingSetIndex ==  0, ]
    set.dat.val   =
set.dat[set.dat$trainingSetIndex == -1, ]

    cl = fl
    cl = makeCluster(cl)
    registerDoParallel(cl)

    L = foreach(j = 1:length(models),
.packages = pack, .export = ex) %dopar% {
            model.name  = models[j]
            model.sav   = paste(set, model.name,
    ".rds", sep = "")
            model       = readRDS(model.sav)
            model.train = predict(model,
    newdata = set.dat.train)
            model.test  = predict(model,
    newdata = set.dat.test)
            model.val   = predict(model,
    newdata = set.dat.val)

            list(model.train = model.train,
    model.test = model.test,
    model.val = model.val)
    }
    stopCluster(cl)

    names(L)    = models
    model.train = do.call(cbind.data.frame,
sapply(L, function(x) x[1]))
    model.test  = do.call(cbind.data.frame,
sapply(L, function(x) x[2]))
    model.val   = do.call(cbind.data.frame,
sapply(L, function(x) x[3]))

    L = list(model.train = model.train,
```

```
    model.test = model.test, model.val = model.val,
                 bv.train = set.dat.train$basketValue,
        bv.val = bv.val,
                 bv.test = set.dat.test$basketValue,
        models = models)

        setwd(path.pred)
        saveRDS(L, paste(set, ".rds", sep = ""))
}
```

```
#########################################################
################ Ensemble Selection ###############
#########################################################
# With this file, the ensemble selection for each
# random partion is realized.
# The algorithm used for ensembling is the greedy
# optimization hill-climbing technique with bagging.
# The model prediction lists are loaded from
# "/Predictions". A data frame including DMC measures
# for each ensemble built from a partition is saved
# to "/Predictions".
#########################################################

pack = unique("compiler")

lapply(pack, function(x) if (!(x %in%
installed.packages())) {
  install.packages(x)
})

lapply(pack, library, character.only = TRUE)

# Functions
AC = function(y, yhat){
  denom = mean(y)
  crit  = ( abs(y - yhat) / denom )^2
  crit  = sum(crit)
  return(crit)
}

ACs = function(y, yhat){
  denom = mean(y)
  crit  = ( abs(y - yhat) / denom )^2
  crit  = colSums(crit)
  return(crit)
}

greedOpt = function(X, Y, iter = 100L){

  # Initialize Variables
```

```
  N           = ncol(X)
  weights     = rep(0L, N)
  pred        = 0 * X
  sum.weights = 0L

  while(sum.weights < iter) {
    # No. of Models/Slots in Ensemble
    sum.weights   = sum.weights + 1L

    # "Trick": Compute Matrix with predictions
    # of each Candidate-Ensemble
    pred          = (pred + X) * (1L / sum.weights)

    # Compute Prediction error of Candidate-Ensemble
    # Here: Use our Score Function?
    errors        = ACs(Y, pred)

    # Choose Candidate-Ensemble with smallest Error
    best          = which.min(errors)

    # increase Weight of best Model in Ensemble
    # Model
    weights[best] = weights[best] + 1L

    # Correctly weight Prediction (and Use only
    # chosen ensemble) for next step
    pred          = pred[, best] * sum.weights
  }

  # Return the weight of each of M Models in
  # Ensemble Model
  return(weights / sum.weights)
}
# Compile Function
greedOptc = cmpfun(greedOpt)

# Function for bagged Ensemble Selection
BES = function(X, Y, b = 10L, p = 0.5, r = 100L){
  i = 0L
  N = nrow(X)
```

```
  M = ncol(X)
  W = matrix(rbinom(b * M, 1, p), ncol = M)

  while(i < b){
    i          = i + 1L
    ind        = which(W[i, ] == 1)
    W[i, ind] = W[i, ind] *
    greedOptc(X[, ind], Y, r)
  }

  return(colSums(W)/b)
}
BES = cmpfun(BES)

# Set Paths
path         = "/Volumes/bommesel.hub/Projects/
Predictive␣Analytics"
path.data    = paste(path, "/DMC2015", sep = "")
path.source  = paste(path, "/Sources", sep = "")
path.results = paste(path, "/Estimated␣Models",
sep = "")
path.pred    = paste(path, "/Predictions", sep = "")

# Ensemble setup (Settings for ES without Bagging)
b = 10L  # Number of Samples
p = 0.5  # Probability that a Model is selected as
# Part of Sample
r = 500L # How often should be averaged per
# Ensemble?


setwd(path.pred)

set = list.files(path.pred)
set = set[-which(set %in% c("EStrain.rds",
"EStest.rds","EStrainontest.rds"))]
set = substr(set, 1, 14 )

for(i in 1:length(set)){
        print(i)
```

```
    set.pred =
readRDS(paste(set, ".rds", sep = "")[i])

    ### Run Ensemble selection on Training set ->
### Evaluate with test and validation set
    train.pred = set.pred$model.train
    train.bv   = set.pred$bv.train

    test.pred  = set.pred$model.test
    test.bv    = set.pred$bv.test

    # Get Ensemble weights
    train.weight =
BES(train.pred, train.bv, b, p, r)
    test.weight  =
BES(test.pred,   test.bv, b, p, r)

    # Weight Predictions for validation set
    val.pred    = set.pred$model.val
    val.bv      = set.pred$bv.val
    val.n       = length(val.bv)

    # Multiply weights with model predictions

    test.es.train.pred = sweep(test.pred, MARGIN = 2,
STATS = train.weight, FUN = "*")
    test.es.train.pred = rowSums(test.es.train.pred)

    val.es.train.pred = sweep(val.pred, MARGIN = 2,
STATS = train.weight, FUN = "*")
    val.es.train.pred = rowSums(val.es.train.pred)

    val.es.test.pred  = sweep(val.pred, MARGIN = 2,
STATS = test.weight, FUN = "*")
    val.es.test.pred  = rowSums(val.es.test.pred)

    # Assess model fit

    if(i == 1){
            AC.test.train = AC(test.bv,
```

```
        test.es.train.pred)
                AC.val.train = AC(val.bv, val.es.train.pred)
                AC.val.test  = AC(val.bv, val.es.test.pred)
        }else{
                AC.test.train = c(AC.test.train,
        AC(test.bv,test.es.train.pred))
                AC.val.train = c(AC.val.train,
        AC(val.bv,val.es.train.pred))
                AC.val.test  = c(AC.val.test,  AC(val.bv,
        val.es.test.pred))
        }
}

saveRDS(AC.test.train, "EStrainontest.rds")
saveRDS(AC.val.train, "EStrain.rds")
saveRDS(AC.val.test,  "EStest.rds")
```

```
#########################################################
#################### Results ############################
#########################################################
# With this file the results of the simulation are
# evaluated. Models and ensembles are loaded from
# "/Predictions". Minimal ACs for base classifiers
# are determined as well as summary statistics for
# ensembles. Several plots for ACs are created.
#########################################################

install.packages("psych")
library(psych)

# Set Paths
path         = "H:/Projects/Predictive␣Analytics"
path.data    = paste(path, "/DMC2015", sep = "")
path.source  = paste(path, "/Sources", sep = "")
path.results = paste(path, "/Estimated␣Models",
sep = "")
path.pred    = paste(path, "/Predictions", sep = "")

# Functions
AC = function(y, yhat){
  denom = mean(y)
  crit  = ( abs(y - yhat) / denom )^2
  crit  = sum(crit)
  return(crit)
}


ACs = function(y, yhat){
  denom = mean(y)
  crit  = ( abs(y - yhat) / denom )^2
  crit  = colSums(crit)
  return(crit)
}

setwd(path.pred)

set = list.files(path.pred)
```

```
set = substr(set, 1, 14 )
set = set[-which(set %in%
c("EStrain.rds","EStest.rds"))]
for(i in 1:length(set)){
  print(i)
  set.pred = readRDS(paste(set, ".rds", sep = "")[i])

  test.pred  = set.pred$model.test
  test.bv    = set.pred$bv.test

  val.pred   = set.pred$model.val
  val.bv     = set.pred$bv.val

  for(j in 1:ncol(test.pred)){
    if(j == 1){
      AC.test = ACs(test.bv,test.pred[j])
      AC.val  = ACs(test.bv,test.pred[j])
    }
    else{
      AC.test = c(AC.test, ACs(test.bv,test.pred[j]))
      AC.val  = c(AC.val,ACs(val.bv,val.pred[j]))
    }
  }

  if(i == 1){
    AC.test.df = as.data.frame(t(AC.test))
    AC.val.df  = as.data.frame(t(AC.val))
  }
  else{
    AC.test.df = rbind(AC.test.df,AC.test)
    AC.val.df = rbind(AC.val.df,AC.val)
  }
}


AC.test.df = cbind(set = as.character(set),AC.test.df)
AC.val.df = cbind(set = as.character(set),AC.val.df)
```

```
### Find minimum over all models
subset = AC.test.df[1:98,2:15]
min.models = AC.test.df$set[which(subset ==
min(subset))]
min(subset)
min.models


### compare single models and ensembles
AC.es.train.on.val = readRDS("EStrain.rds")
AC.es.test.on.val  = readRDS("EStest.rds")
AC.es.train.on.test = readRDS("EStrainontest.rds")


boxplot(AC.es.train.on.test,
        col = "lightblue",
        main = "Boxplot␣of␣ACs␣for␣ensembles␣on
␣␣␣␣␣␣␣␣hold-out␣sample",
        las = 1)
describe(AC.es.train.on.test)

### AC of baseline prediction
AC.baseline.val  = AC(val.bv,rep(mean(val.bv),669))

par(mfrow =c(2,1))
plot(AC.es.on.test,AC.es.on.val,
     las = 1,
     main = "ACs␣of␣ensembles",
     xlab = "AC␣for␣hold-out␣sample",
     ylab = "AC␣for␣validation␣set",
     mgp  = c(3, .6, 0))
abline(h = AC.baseline.val, col = "red")
abline(h = mean(AC.es.on.val), col = "green")


plot(AC.es.on.test,AC.es.on.val,
     las = 1,
     xlab = "AC␣for␣hold-out␣sample",
     ylab = "AC␣for␣validation␣set",
     xlim = c(0,AC.baseline.test),
```

```
     mgp  = c(3, .6, 0))
abline(h = AC.baseline.val, col = "red")
abline(h = mean(AC.es.on.val), col = "green")


# par(mfrow =c(1,1))
# plot(AC.es.on.val,AC.es.test.on.val,
#      las = 1,
#      main = "ACs of ensembles",
#      xlab = "train on validation",
#      ylab = "test on validation",
#      mgp  = c(3, .6, 0))
# abline(h = AC.baseline.val,
# v = AC.baseline.val, col = "red")
# abline(h = mean(AC.es.test.on.val),
# v = mean(AC.es.on.val), col = "green")
#
# hist(AC.es.on.val,breaks="FD")
# hist(AC.es.on.val[which(AC.es.on.test <
# 3500)],breaks="FD")
```

References

[1] Fernández-Delgado, Manuel ; Cernadas, Eva ; Barro, Senén ; Amorim, Dinani: Do We Need Hundreds of Classifiers to Solve Real World Classification Problems? In: *J. Mach. Learn. Res.* 15 (2014), Januar, Nr. 1, S. 3133–3181. – ISSN 1532–4435

[2] Caruana, Rich ; Niculescu-Mizil, Alexandru: Ensemble Selection from Libraries of Models. In: *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*, 2004. – ISBN 1–58113–838–5

[3] Caruana, Rich ; Munson, Art ; Niculescu-Mizil, Alexandru: Getting the Most Out of Ensemble Selection. In: *Proceedings of the 6th International Conference on Data Mining (ICDM '06)*, 2006

[4] Niculescu-Mizil, A. et a.: Winning the KDD Cup Orange Challenge with Ensemble Selection. In: *IBM Research* (2009)