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 (DMC) for six weeks from April 7 to May 19 2015 and we participated as part of a seminar on "Applied Predictive Analytics". Our two submissions ranked first and sixth place out of 188 teams from a total of 153 universities from 48 countries. 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.

Furthermore, we extend the insight into our modeling techniques by incorporating ex post information about the values of the target variables. This knowledge is used to evaluate whether there could have been (1) problems due to the splitting of test and training sample and (2) undetected outliers and during the competition duration. We aim to answer these questions by repeatedly dividing the data set into training and hold-out samples and estimation of the respective base models and ensembles. The results are then compared by summary statistics and explorative data visualization techniques.

The written Code is available on Github (github.com/ebommes/SE-Predictive-Analytics-BasketValue) but we are not able to submit all estimation files as they need 176 GB of disk space. This data is currently stored on the servers of the CRC 649 Research Data Center (RDC) and can be provided up on request.

2 Data

The initial training data set contains 6,054 observations of 33 variables. The distribution of the target variable basket value (BV) is right-skewed as one can conclude by comparing the estimated location parameters in Table 1. Note that our naming conventions of the estimated parameters are also given in Table 1. By definition of the boxplot, strong extremes may be identified as observations lying outside of the range $[Q1 - 3\hat{\sigma}, Q3 + 3\hat{\sigma}]$. We can directly isolate 237 strong extremes by this technique. Figure 1 shows the histogram of the basket value. One can observe that orders with a basket value of around 185 are especially frequent with more or less steadily declining frequencies of higher basket values. Also, there are two (in magnitude) smaller spikes for values smaller than 180.

$\hat{\mu}$	$\hat{\sigma}$	Min.	Q1	Q2	Q3	Max.	Range
315.50	1846.07	46.36	186.30	212.60	294.10	135,800	135,828.90

$\hat{\mu}$ and $\hat{\sigma}$ refer to the estimated mean and standard deviation, respectively. Q1, Q2, Q3 are the sample's first, second (median) and third Quantile. Min. and Max. are minimum and maximum and Range is defined as $Min. - Max.$

Table 1: Summary statistics of basket value

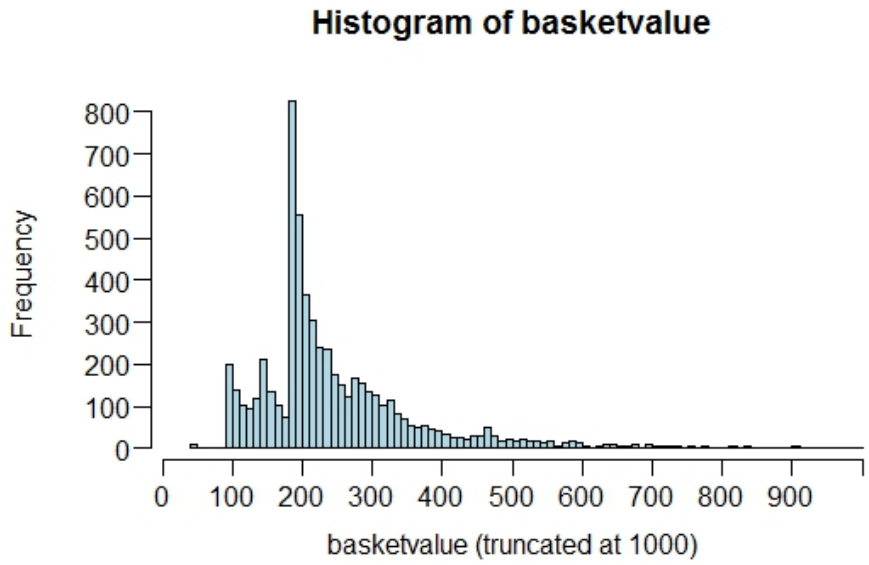

Figure 1: Distribution of the basket value variable

In order to stay consistent with the outlier detection and data preparation methods used for the initial competition submission, we do not try to incorporate the findings of Figure in the further analysis. Furthermore, we use the same procedure for outliers that is: (1) Identify outliers as $BV > 700$ and (2) if available, replace the outlier by the customer's mean basket value and otherwise by the overall mean. We also make use of the same new variables, over 100, that have been defined during the initital competition. The partitioning of the data set into training and test (hold-out) set is necessary to avoid overfitting during methods described in Section 3. Again, we stay in line with the original estimation and split our data set into 80% training and 20% test sample.

We also made sure that we can also build these variables for the provided prediction data set, also referred to as validation set. For example, we did not use any

of the coupon usage variables to predict the basket values as they have not been available for the validation set since they are also regarded as target variables in the competition. Prudsys published the actual values of this data sets' target variables (and hence also BV) after the competition ended.

3 Methods and Estimation

We make use of the `R` package `caret` by Kuhn (2015) to build a library of base classifiers. The name `caret` is short for *c*lassification *a*nd *re*gression *t*raining. The package includes several different learning methods such as random forests, support vector machines and gradient boosting machines. In the further analysis, we refer to each model by using the name given in the caret package such that e.g. `qrf` stands for "Quantile Random Forest". We forego a formal discussion of these base classifiers as it would exceed the scope of this work and the descriptions can be found as well in the `caret` manual as in many textbooks. We estimate a total of 16 different base classifiers by using five-fold crossvalidation to avoid overfitting our models. The choice of the specific models is align with Fernández-Delgado et al. (2014) who state that various kinds of random forest models usually perform best, followed by diverse support vector machines. The models are trained to minimize the following assessment criterion (AC) that has been given in the competition task:

$$AC = \sum_{i=1}^{n} \frac{|BV_i - \text{Pred}(BV_i)|}{\frac{1}{n}\sum_{j=1}^{n} BV_i} \tag{1}$$

with Pred$(\cdot)$ referring to the individual model prediction. One can observe that it is basically the Manhatten distance scaled by the factor $\frac{1}{n}\sum_{j=1}^{n} BV_i$ and thus, models optimized using AC should be more robust against outliers than e.g. models utilizing the Euclidian distance.

After estimation of the base models, we use the ensemble selection techniques as described in both Caruana et al. (2006) and Caruana and Niculescu-Mizil (2004). More specifically, we implement the bagged ensemble selection method with replacement as applied by Caruana and Niculescu-Mizil (2004). The idea behing ensemble selection is to combine the predictions of all models to a single prediction which is of superior power. The data set which is used to determine the weights of each individual model in the ensemble prediction is called hillclimbing set. To avoid overfitting, the ensemble's predictive power should be assessed according a test set with observations that have not been used to either train the individual models or create the ensemble weights.

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

Up to now, there is no difference between our approach during the competition's duration and our current discussion of the results. Our main additional contribution in this seminar paper is, that we also look at the extent to which the selection of training and test set influences the performance regarding the DMC prediction data set. We repeatedly split the data set into different training and test sets to achieve this goal. Afterwards, model estimation and ensemble selection is performed for each of these sets. This approach lets us determine how heavily the prediction performance depends on the initial choice of training and test data.

Additionally, we suspect that the performance of all machine learning methods during the competition time might have been influenced by the mechanism of how to choose training and test set. During the competition, we used the textttcaret function `createDataPartition` to split our data set which tries to maintain the distribution of the target variable. However, the basket value is only one of four target variables in the DMC competition as the set also includes three variables that indicate coupon usage. As for the competition submission, we intially used the first coupon variable to split the data. This way of sampling leads to an insufficiently described distribution of the basket value variable in the training sample and therefore yielding deficient prediction results.

We split the initial data set 98 times in different training and test set, prepare these sets according to Section 2 and subsequently perform model estimation and ensemble selection to investigate this matter. The previous estimation of the base models turned out to be rather tedious and time consuming, taking up to 24 hours to obtain estimates for all mentioned models. Hence, we improve the programming code by utilizing parallelization strategies in the `R` package `doParallel` by Analytics and Weston (2014).

4 Results

Firstly, we review the fit of the base models during the competition phase. We perform ensemble selection according to Section 3. The resulting weights are presented in Table 2. The models `cubist` and `parRF` obtain the largest weights and thus, represent together 80% of the resulting ensemble prediction.

Model	AC Holdout Set Sample (n=1210)	AC Validation Set (n=669)
UserID-Model	2,486	1,167
`cubist`	3,355	1,741
Ensemble	3,390	1,766
`parRF`	3,466	1,817
`RRF`	3,467	1,816
`RRFglobal`	3,467	1,816
`rf`	3,468	1,815
`qrf`	3,472	1,817
`ctree`	3,475	1,816
`cforest`	3,485	1,824
`gbm`	3,469	1,825
`blackboost`	3,506	1,830
`mlp`	3,742	1,971
`svmRadialCost`	3,786	1,995
`svmRadial`	3,742	1,995
`mlpWeightDecay`	3,832	2,020

Table 3: Evaluation of base classifiers (without obvious misclassifiers ($AC > 100,000$)

Table 3 reports the ACs of our models, estimated by using the training data set and evaluated by either using the initial test set or the competition's validation set. The "UserID-Model" stands for an intuitive and probably more insightful model prepared by another group: The basket value variable is predicted by taking the mean of previous basket values for known customers while basket values of unknown customers are predicted by a simple random forest model. Clearly, this model seems to have some advantages as it both has the lowest AC for the hold-out and validation set.

Other interesting findings are that the model `cubist` outperforms the ensemble selection technique while the random forest methods perform quite well and better than e.g. support vector machines as expected due to Fernández-Delgado et al. (2014).

In the next step, we investigate whether the data partitioning might have had

$\hat{\mu}$	$\hat{\sigma}$	Min.	Q1	Q2	Q3	Max.	Range
1534	187.2	924.8	1,437	1,523	1,606	2,210	1,285.2

Table 4: Summary statistics: Assessment criterion of 98 ensembles on validation sample

an influence on prediction performance. Thus, we partition the data 98 times and estimate the models and perform the ensemble selection for each data set. The resulting predictions are evaluated by using the competition's validation set that was not available during the competition duration. Summary statistics of the resulting ACs can be seen in Table 4.

Indeed, at least 75 % of the estimated ensembles perform better than the initially estimated ensemble and base models in Table 3. More specifically, a simple check shows that this is the case for 88.78 % of the ensembles. However, it is also clear that only few ensembles outperform the UserID-Model (less than 25%) and further investigation shows that this is only the case for two ensembles.

$\hat{\mu}$	$\hat{\sigma}$	Min.	Q1	Q2	Q3	Max.	Range
56,153	69,616.19	404	1,686	11,030	115,600	237,500	237,134

Table 5: Summary statistics: Assessment criterion of 98 ensembles on hold-out sample

One might argue that this ex post point of view does not improve predictions during the competition phase. Thus, we select the ensemble that minimizes the AC while being evaluated by using the hold-out set. With an AC equal to $1539,612$ for the validation set it performs worse than most of the other ensembles but it is still reasonably better than the initial ensemble. However, if evaluated during the competition phase, most ensembles show a very high AC for the hold-out set. This can be clearly seen in Table 5 and Figure 2. Of course, as the test set has 1,209 observations while the validation set only has 669 observations the AC evaluated on the test set should be naturally higher. However, an AC more than ten times greater seems quite excessive. One reason for these high ACs might be outliers that have not been previously recognized.

By taking a look on the upper part of Figure 2, 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 2, 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 3,500 for the hold-out sample. Therefore, to choose an ensemble with good performance on the hold-out sample is still reasonable.

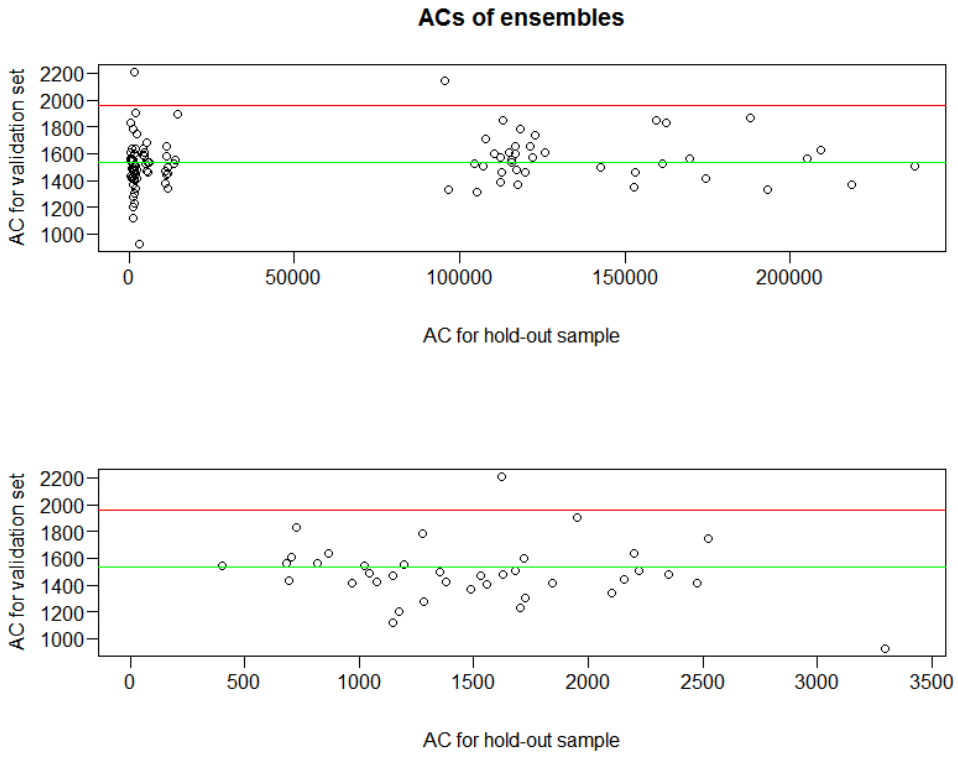

Figure 2: Scatterplot of ACs of ensembles for hold-out sample and validation set. (Top) the entire range of ACs for hold-out sample, (bottom) truncated at 3500. Red lines indicate the AC of a mean prediction, green lines indicate the average AC on the validation set.

5 Conclusion

By conducting a simulation study for the prediction of basket values with ensemble selection, we gained further insight in the importance of partitioning and detection of outliers in the context of machine learning techniques. By using differently partitioned data sets for training and testing we decreased the probability of drawing samples with different distribution than the target variable which, in general, yield bad performance measures.

Our first assumption was that the inferiority of machine learning models (during the competition phase) 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. We can sustain this theory as a large percentage of estimated ensembles perform better than the initial competition ensemble. However,

most of these ensembles still do not lead to better results if compared to the UserID-Model. Hence, we can conclude that a simple and smart approach which incorporates knowledge about the data might be superior in comparison to obtusely applied machine learnings algorithms.

We can also conclude that most ensembles perform worse on the hold-out set than on the validation set. This might be due to outliers that have not been detected in prior stages of data analysis. This emphasizes the importance of outlier handling which should be conducted with care in future competitions.

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".
#######################################################
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",
          "kknn", "nnet", "brnn", "frbs", "RSNNS", "foreach", "caret",
          "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", "glmboos
         "qrf", "rf", "RRF","RRFglobal", "svmRadial","svmRadialCost")

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


# 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 = "")

#
```

```
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])

fl = detectCores()
### Loop over completed estimations
for(i in 1: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) %d
    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
  }
  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.v
    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 = ""))
}
```

```
#########################################################
################## 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

Analytics, R. and Weston, S. (2014). *doParallel: Foreach parallel adaptor for the parallel package*. R package version 1.0.8.

Caruana, R., Munson, A., and Niculescu-Mizil, A. (2006). Getting the most out of ensemble selection. In *Proceedings of the 6th International Conference on Data Mining (ICDM '06)*.

Caruana, R. and Niculescu-Mizil, A. (2004). Ensemble selection from libraries of models. In *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res.*, 15(1):3133–3181.

Kuhn, M. (2015). *caret: Classification and Regression Training*. R package version 6.0-47.

Niculescu-Mizil, A. e. a. (2009). Winning the kdd cup orange challenge with ensemble selection. *IBM Research*.