overview result variables

Kim-Laura Speck

26 Juli, 2025

Contents

Basic structure (of results)

This document provides an overview over all result variables from models fitted to the data.

```r
# general collection of parameter values
source("utils/setParameters.R") # import parameter values

# all result files in this folder
resFolder <- paste0("results/")

##### raw results data #####
# ... with subfolders for all data generating processes
#     {"results/pwlinear", "results/nonlinear3", "results/inter"}

# there are separate result files depending on ...
# ... N = number of observations in the training data
# ... pTrash = number of "trash" predictors (without association to the outcome)
# .. reliability = amount of measurement error in the predictors (i.e., reliability)
# separation of these result files due to different data sets (raw data)
length(setParam$dgp$N) * length(setParam$dgp$pTrash) * length(setParam$dgp$reliability)
```

```
## [1] 18
```

```r
condGrid <- expand.grid(N = setParam$dgp$N,
                        pTrash = setParam$dgp$pTrash,
                        reliability = setParam$dgp$reliability)

(condN_pTrash <- paste0("N", condGrid$N,
                        "_pTrash", condGrid$pTrash,
                        "_rel", condGrid$reliability))
```

```
##  [1] "N100_pTrash10_rel0.6"  "N300_pTrash10_rel0.6"  "N1000_pTrash10_rel0.6"
##  [4] "N100_pTrash50_rel0.6"  "N300_pTrash50_rel0.6"  "N1000_pTrash50_rel0.6"
##  [7] "N100_pTrash10_rel0.8"  "N300_pTrash10_rel0.8"  "N1000_pTrash10_rel0.8"
## [10] "N100_pTrash50_rel0.8"  "N300_pTrash50_rel0.8"  "N1000_pTrash50_rel0.8"
## [13] "N100_pTrash10_rel1"    "N300_pTrash10_rel1"    "N1000_pTrash10_rel1"
## [16] "N100_pTrash50_rel1"    "N300_pTrash50_rel1"    "N1000_pTrash50_rel1"
```

```r
# thus, for all models there are 18 different data sets
# model x N x pTrash x rel = 3 * 18
modelVec = c("ENETw", "ENETwo", "GBM", "RF")

# additionally, there are different conditions withon each data set (results and raw data)
# every simulated combination of R^2 and the distribution of linear and interaction effect strength
# results are within the same results file due to identical raw data that these results are based on
#     ... reason: predictor matrices are identical; only y is different for these simulated conditions
length(setParam$dgp$Rsquared) * length(setParam$dgp$percentLinear)
```

```
## [1] 9
```

```r
setParam$dgp$condLabel
```

```
## [1] "R20.2lin_inter0.5_0.5" "R20.5lin_inter0.5_0.5" "R20.8lin_inter0.5_0.5"
## [4] "R20.2lin_inter0.8_0.2" "R20.5lin_inter0.8_0.2" "R20.8lin_inter0.8_0.2"
## [7] "R20.2lin_inter0.2_0.8" "R20.5lin_inter0.2_0.8" "R20.8lin_inter0.2_0.8"
```

```r
# this is what idxCondLabel in every results matrix refers to

##### dependent measures #####
# ... within DGP folders is the dependent measures folder of the respective DGP
#     e.g., "results/pwlinear/dependentMeasures"
# for all models there are 4 different data sets
#     {hyperParametersSample, performPerSample, performTestStats, performTrainStats}
# in these files: results of every simulated condition for the respective DGP and ML method
```

tree-based methods: GBM & RF

```r
# GBM
dataGBM <- readRDS(paste0(resFolder, "/inter/resultsModelGBM_N100_pTrash10_rel0.6.rds"))
names(dataGBM)
```

```
## [1] "performTrainStats"  "performTestStats"   "performPerSample"
## [4] "pvi"                "performCVtestStats" "interStrength"
## [7] "selectionPerSample"
```

```r
# RF
dataRF <- readRDS(paste0(resFolder, "/inter/resultsModelRF_N100_pTrash10_rel0.6.rds"))
names(dataRF)
```

```
## [1] "performTrainStats"  "performTestStats"   "performPerSample"
## [4] "pvi"                "performCVtestStats" "oobPredictions"
## [7] "oobR2"              "selectionPerSample"
```

performTrainStats

average model performance on (full) train data (M, SD, SE)

- Root Mean Squared Error (RMSE)
- explained variance (R2)
- Mean Absolute Error (MAE)

… for model training

```r
# structure of the results (essentially the same for RF)
head(dataGBM[["performTrainStats"]])
```

```
##                  M         SD          SE idxCondLabel
## RMSE     1.0835239 0.08755790 0.002767440            1
## Rsquared 0.2008915 0.07627122 0.002410703            1
## MAE      0.8589332 0.07154864 0.002261436            1
## RMSE     1.2687910 0.14763225 0.004666209            2
## Rsquared 0.3122110 0.12246661 0.003870799            2
## MAE      0.9778873 0.10362474 0.003275265            2
```

```r
dim(dataGBM[["performTrainStats"]])
```

```
## [1] 27  4
```

```r
unique(rownames(dataGBM[["performTrainStats"]]))
```

```
## [1] "RMSE"     "Rsquared" "MAE"
```

```r
length(unique(rownames(dataGBM[["performTrainStats"]]))) * length(setParam$dgp$condLabels)
```

```
## [1] 27
```

performCVtestStats

average model performance on hold out fold in cross-validation procedure (M, SD, SE)

- Root Mean Squared Error (RMSE)
- explained variance (R2)
- Mean Absolute Error (MAE)

… hold out fold data during model training (training data)

```r
# structure of the results (essentially the same for RF)
head(dataGBM[["performCVtestStats"]])
```

```
##                     M         SD          SE idxCondLabel
## RMSE_CVtest 1.0790017 0.07950181 0.002512812            1
## Rsq_CVtest  0.1445303 0.06034262 0.001907248            1
## MAE_CVtest  0.8695205 0.06705455 0.002119392            1
## RMSE_CVtest 1.2867988 0.11218902 0.003545956            2
## Rsq_CVtest  0.2272767 0.08981952 0.002838924            2
## MAE_CVtest  1.0181346 0.08456715 0.002672912           2
```

```r
dim(dataGBM[["performCVtestStats"]])
```

```
## [1] 27  4
```

```r
unique(rownames(dataGBM[["performCVtestStats"]]))
```

```
## [1] "RMSE_CVtest" "Rsq_CVtest"  "MAE_CVtest"
```

```r
length(unique(rownames(dataGBM[["performCVtestStats"]]))) * length(setParam$dgp$condLabels)
```

```
## [1] 27
```

performTestStats

average model performance on test data (M, SD, SE)

- Root Mean Squared Error (RMSE)
- explained variance (R2)
- Mean Absolute Error (MAE)

. . . for model testing

```r
# structure of the results (essentially the same for RF)
head(dataGBM[["performTestStats"]])
```

```
##                    M          SD           SE idxCondLabel
## RMSE_test 1.10345218 0.011270409 0.0003562235            1
## Rsq_test  0.02361864 0.016926214 0.0005349864            1
## MAE_test  0.89184057 0.008785307 0.0002776770            1
## RMSE_test 1.34620631 0.040657867 0.0012850723            2
## Rsq_test  0.10614406 0.040656753 0.0012850371            2
## MAE_test  1.04568056 0.027642576 0.0008736983            2
```

```r
dim(dataGBM[["performTestStats"]])
```

```
## [1] 27  4
```

```r
unique(rownames(dataGBM[["performTestStats"]]))
```

```
## [1] "RMSE_test" "Rsq_test"  "MAE_test"
```

```r
length(unique(rownames(dataGBM[["performTestStats"]]))) * length(setParam$dgp$condLabels)
```

```
## [1] 27
```

performPerSample

test and train model performance for every sample seperately

```r
# structure of the results (essentially the same for RF)
head(dataGBM[["performPerSample"]])
```

```
##      RMSE_train  Rsq_train MAE_train RMSE_test     Rsq_test  MAE_test
## [1,]   1.162375 0.22813035 0.9051808  1.101070 0.0233603743 0.8904399
## [2,]   1.091914 0.17431292 0.8769979  1.100238 0.0327222031 0.8901094
## [3,]   1.213140 0.20250021 0.9923984  1.102638 0.0010216920 0.8914031
## [4,]   1.121007 0.12426337 0.8563034  1.100246 0.0251757667 0.8901164
## [5,]   1.105688 0.09758817 0.8949247  1.100452 0.0009871436 0.8901617
## [6,]   1.212992 0.28630863 0.9389962  1.074721 0.0478410435 0.8630141
##      idxCondLabel
## [1,]            1
## [2,]            1
## [3,]            1
## [4,]            1
## [5,]            1
## [6,]            1
```

```r
dim(dataGBM[["performPerSample"]])
```

```
## [1] 9000    7
```

```r
length(setParam$dgp$condLabels) * setParam$dgp$nTrain
```

```
## [1] 9000
```

pvi

permutation variable importance from the iml package

Molnar C, Bischl B, Casalicchio G (2018). "iml: An R package for Interpretable Machine Learning." *JOSS*, *3*(26), 786. https://doi.org/10.21105/joss.00786.

- model agnostic statistic

- calculated with the FeatureImp-function
- the factor by which the model's prediction error increases when the feature is shuffled
- high values signify high importance of the predictor
- importance values of 1 signifies irrelevance of the predictor for prediction

We do not report variable importance measures, which is why we did not compute PVI and the corresponding values are NA.

```r
# structure of the results (essentially the same for RF)
# for evaluation of pvi: idxCondLabel, sample, pviRank, pviValue in the results
head(dataGBM[["pvi"]])
```

```
##                         [,1]
## R20.2lin_inter0.5_0.5   NA
## R20.5lin_inter0.5_0.5   NA
## R20.8lin_inter0.5_0.5   NA
## R20.2lin_inter0.8_0.2   NA
## R20.5lin_inter0.8_0.2   NA
## R20.8lin_inter0.8_0.2   NA
```

```r
dim(dataGBM[["pvi"]])
```

```
## [1] 9 1
```

```r
# dimensions depend on number of trash predictors
(setParam$dgp$pTrash[1] + length(setParam$dgp$linEffects)) *
  length(setParam$dgp$condLabels) * setParam$dgp$nTrain
```

```
## [1] 126000
```

```r
# pviRank = feature
# pviValue = importance
```

selectionPerSample

save cross-validated tuning parameters from each sample

```
# due to different sets of hyperparameters not the same for both tree-based methods
# GBM: {shrinkage, max_depth, min_child_weight, Nrounds, idxCondLabel}
head(dataGBM[["selectionPerSample"]])
```

```
##      shrinkage max_depth min_child_weight Nrounds idxCondLabel
## [1,]     0.001         1                5       5            1
## [2,]     0.001         1                5       5            1
## [3,]     0.001         1                5       5            1
## [4,]     0.001         1                5       5            1
## [5,]     0.001         1                5       5            1
## [6,]     0.101         1                5       5            1
```

```
dim(dataGBM[["selectionPerSample"]])
```

```
## [1] 9000    5
```

```
# RF: {mtry, splitRule, minNode, idxCondLabel}
head(dataRF[["selectionPerSample"]])
```

```
##      mtry splitRule    minNode idxCondLabel
## [1,] "2"  "variance"   "5"     "1"
## [2,] "2"  "extratrees" "5"     "1"
## [3,] "2"  "variance"   "5"     "1"
## [4,] "2"  "variance"   "5"     "1"
## [5,] "2"  "variance"   "5"     "1"
## [6,] "2"  "variance"   "5"     "1"
```

```
dim(dataRF[["selectionPerSample"]])
```

```
## [1] 9000    4
```

interStrength (only GBM)

H-statistic to quantify/evaluate predictive value of interactions

Friedman, J. H., & Popescu, B. E. (2008). Predictive learning via rule ensembles. Section 8.1

- model agnostic statistic
- only for variables with simulated main effects otherwise interaction strength is overestimated
- see also Greenwell et al. (2018) and Henninger et al. (2023)
- does feature interact with any other feature?
  - The interaction strength between two features is the proportion of the variance of the 2-dimensional partial dependence function that is not explained by the sum of the two 1-dimensional partial dependence functions.
  - intereaction strength between 0 (no interaction) and 1 (all of variation of the predicted outcome depends on a given interaction)
    * pd(ij) = interaction partial dependence of variables i and j
    * pd(i) = partial dependence of variable i
    * pd(j) = partial dependence of variable j
    * upper = sum(pd(ij) - pd(i) - pd(j))
    * lower = variance(pd(ij))
    * rho = upper / lower

  – partial dependence of the interaction relative to partial dependence of the main effects; thus, for variables without simulated effects the overall interaction strength might be as high as for variables with actually simulated interactions only because these variables do have main effects as well (see also Greenwell et al. (2018) and Henninger et al. (2023))
  – across variables comparison of overall interaction strength is meaningsless

We do not report the H-Statistic, which is why we did not compute interStrength and the corresponding values are NA.

```
head(dataGBM[["interStrength"]])
```

```
##                         [,1]
## R20.2lin_inter0.5_0.5   NA
## R20.5lin_inter0.5_0.5   NA
## R20.8lin_inter0.5_0.5   NA
## R20.2lin_inter0.8_0.2   NA
## R20.5lin_inter0.8_0.2   NA
## R20.8lin_inter0.8_0.2   NA
```

```
dim(dataGBM[["interStrength"]])
```

```
## [1] 9 1
```

oobPredictions & oobR2 (only RF)

We do not report or use the Out-of-bag predictions or their prediction error.

```
dataRF[["oobPredictions"]][1:6, 1:6]
```

```
##              [,1]        [,2]       [,3]        [,4]       [,5]        [,6]
## [1,] -0.005013359  0.35818285  0.2238570  0.28734148  0.8916145 -0.21500936
## [2,]  0.214616121  0.69044065  0.1586170  1.03065552  0.2090609 -0.03211758
## [3,]  0.408678255  0.39695138  0.4098636  0.15570533  0.2033243 -0.14859959
## [4,]  0.540611985  0.59063294 -0.4222683  0.33944760 -0.2142790 -0.28760979
## [5,] -0.133740490  0.59009687 -0.2356972 -0.05693068  0.2549343 -0.24516830
## [6,]  1.141661841 -0.02403718  0.1948533  0.24799941  0.9309819 -0.01134058
```

```
dim(dataRF[["oobPredictions"]])
```

```
## [1]  900 1000
```

```
dataRF[["oobR2"]][, 1:6]
```

```
##                              [,1]        [,2]         [,3]         [,4]
## R20.2lin_inter0.5_0.5  0.01087181  0.08043516  0.003589662 -0.007060192
## R20.5lin_inter0.5_0.5  0.07661673 -0.03202806  0.207291882  0.062823422
## R20.8lin_inter0.5_0.5  0.11464707  0.22568987  0.275921929  0.208676287
## R20.2lin_inter0.8_0.2  0.03392361  0.03069442  0.073502190 -0.016998281
## R20.5lin_inter0.8_0.2  0.08375087  0.20934937  0.119034086  0.161054509
## R20.8lin_inter0.8_0.2  0.31510384  0.34478279  0.429115145  0.338838972
## R20.2lin_inter0.2_0.8 -0.00318421  0.02160998 -0.066681743 -0.023201955
## R20.5lin_inter0.2_0.8  0.19364675 -0.05121342  0.064499112 -0.093526295
## R20.8lin_inter0.2_0.8  0.12816033  0.11942104  0.138943904  0.014644483
##                               [,5]        [,6]
## R20.2lin_inter0.5_0.5  0.008015181  0.08910513
## R20.5lin_inter0.5_0.5  0.014625058  0.30700176
## R20.8lin_inter0.5_0.5  0.215402017  0.31860593
## R20.2lin_inter0.8_0.2  0.160070742  0.10589106
## R20.5lin_inter0.8_0.2  0.248938881  0.28679569
## R20.8lin_inter0.8_0.2  0.405451688  0.40624519
## R20.2lin_inter0.2_0.8 -0.011925696 -0.03495520
## R20.5lin_inter0.2_0.8  0.045914216  0.21335860
## R20.8lin_inter0.2_0.8  0.135482362  0.18652537
```

```r
dim(dataRF[["oobR2"]])
```

```
## [1]    9 1000
```

ENET

We fitted elastic net regression with and without every possible interaction term. The results structure is exactly the same for both result sets.

```r
## ENETw
dataENETw <- readRDS(paste0(resFolder, "/inter/resultsModelENETw_N100_pTrash10_rel0.6.rds"))

## ENETwO
dataENETwo <- readRDS(paste0(resFolder, "/inter/resultsModelENETwo_N100_pTrash10_rel0.6.rds"))

names(dataENETw)
```

```
## [1] "performTrainStats"  "performTestStats"   "performPerSample"
## [4] "pvi"                "estBeta"            "estBetaFull"
## [7] "varSelection"       "selectionPerSample"
```

```r
names(dataENETwo)
```

```
## [1] "performTrainStats"  "performTestStats"   "performPerSample"
## [4] "pvi"                "estBeta"            "estBetaFull"
## [7] "varSelection"       "selectionPerSample"
```

performTrainStats

average model performance on train data (M, SD, SE)

- Root Mean Squared Error (RMSE)
- explained variance (R2)
- Mean Absolute Error (MAE)

… for model training

```r
head(dataENETw[["performTrainStats"]])
```

```
##                  M        SD          SE idxCondLabel
## RMSE     0.9638540 0.1489242 0.004707045            1
## Rsquared 0.3188610 0.2527679 0.007989228            1
## MAE      0.7660259 0.1217781 0.003849036            1
## RMSE     1.0214380 0.1679963 0.005309856            2
## Rsquared 0.5630742 0.1857922 0.005872330            2
## MAE      0.8007862 0.1335958 0.004222561            2
```

```r
dim(dataENETw[["performTrainStats"]])
```

```
## [1] 27  4
```

performTestStats

average model performance on test data (M, SD, SE)

- Root Mean Squared Error (RMSE)
- explained variance (R2)
- Mean Absolute Error (MAE)

. . . for model testing

```
head(dataENETw[["performTestStats"]])
```

```
##                    M         SD           SE idxCondLabel
## RMSE_test 1.10612651 0.02654263 0.0008389323            1
## Rsq_test  0.01962197 0.01959135 0.0006192233            1
## MAE_test  0.89231130 0.02027008 0.0006406760            1
## RMSE_test 1.29699110 0.04997829 0.0015796628            2
## Rsq_test  0.14758524 0.06118686 0.0019339318            2
## MAE_test  1.01296857 0.03377413 0.0010674981            2
```

```
dim(dataENETw[["performTestStats"]])
```

```
## [1] 27  4
```

performPerSample

test and train model performance for every sample seperately

```
head(dataENETw[["performPerSample"]])
```

```
##      RMSE_train Rsq_train MAE_train RMSE_test   Rsq_test  MAE_test idxCondLabel
## [1,]  1.0882093 0.2443076 0.8594166  1.084207 0.03048559 0.8773752            1
## [2,]  0.9544514 0.3954364 0.7666047  1.090018 0.02256270 0.8826405            1
## [3,]  1.2137252 0.0000000 0.9927100  1.102694 0.00000000 0.8914292            1
## [4,]  0.5734281 0.8152358 0.4306151  1.193142 0.02240516 0.9469955            1
## [5,]  1.1061164 0.0000000 0.8952916  1.100415 0.00000000 0.8901628            1
## [6,]  0.9785759 0.5527271 0.7527384  1.069950 0.06541126 0.8629376            1
```

```
dim(dataENETw[["performPerSample"]])
```

```
## [1] 9000    7
```

pvi

permutation variable importance from the iml package

Molnar C, Bischl B, Casalicchio G (2018). "iml: An R package for Interpretable Machine Learning." *JOSS*, *3*(26), 786. https://doi.org/10.21105/joss.00786.

- model agnostic statistic

- calculated with the FeatureImp-function
- the factor by which the model's prediction error increases when the feature is shuffled
- high values signify high importance of the predictor
- importance values of 1 signifies irrelevance of the predictor for prediction

We do not report variable importance measures, which is why we did not compute PVI and the corresponding values are NA.

```
head(dataENETw[["pvi"]])
```

```
##                          [,1]
## R20.2lin_inter0.5_0.5   NA
## R20.5lin_inter0.5_0.5   NA
## R20.8lin_inter0.5_0.5   NA
## R20.2lin_inter0.8_0.2   NA
## R20.5lin_inter0.8_0.2   NA
## R20.8lin_inter0.8_0.2   NA
```

```
dim(dataENETw[["pvi"]])
```

```
## [1] 9 1
```

estBeta

average estimated coefficients (M, SD, SE)

- for all variables in the model (including trash variables)
- additionally for all possible interactions (including trash variables)
- variables which are not selected by ENET were removed from mean calculation: coefficients that are exactly zero (i.e., not selected) are replaced by "NA"

```
head(dataENETw[["estBeta"]])
```

```
##                   M         SD           SE idxCondLabel
## Var1 0.0227737681 0.03481693 0.0011004580            1
## Var2 0.0213082007 0.03295033 0.0010414602            1
## Var3 0.0213128677 0.03256914 0.0010294120            1
## Var4 0.0230680054 0.03430125 0.0010841590            1
## Var5 0.0006496101 0.01539742 0.0004866658            1
## Var6 0.0001823027 0.01494630 0.0004724074            1
```

```
dim(dataENETw[["estBeta"]])
```

```
## [1] 945   4
```

```
length(setParam$dgp$condLabels) *
  (setParam$dgp$nModelPredictors[1] + # all interactions
     length(setParam$dgp$linEffects) + setParam$dgp$pTrash[1]) # linear predictors and trash variables
```

```
## [1] 945
```

estBetaFull

estimated coefficients

- variables which are not selected by ENET were removed: coefficients that are exactly zero (i.e., not selected) are replaced by "NA"

```
dataENETw[["estBetaFull"]][1:6, 1:6]
```

```
##                s0          s0 s0           s0 s0           s0
## Var1 0.051406900 0.003575954 NA  0.062728116 NA 0.1140403198
## Var2          NA 0.136971177 NA  0.093674186 NA 0.0009730791
## Var3 0.001473697          NA NA  0.023776425 NA 0.0311489216
## Var4 0.097028700          NA NA  0.083194649 NA 0.0309959068
## Var5          NA          NA NA -0.008819622 NA           NA
## Var6          NA          NA NA -0.041722740 NA 0.0227874497
```

```
dim(dataENETw[["estBetaFull"]])
```

```
## [1]  945 1000
```

```
# conditions x {variables, interactions} in rows
# samples in columns
length(setParam$dgp$condLabels) *
  (setParam$dgp$nModelPredictors[1] + # all interactions
     length(setParam$dgp$linEffects) + setParam$dgp$pTrash[1]) # linear predictors and trash variables
```

```
## [1] 945
```

varSelection

- How frequently are linear predictors or interactions kept in the model?
  - frequency of variable selection
  - relative frequency

```
head(dataENETw[["varSelection"]])
```

```
##      nSelection percSelection idxCondLabel
## Var1        430          43.0            1
## Var2        411          41.1            1
## Var3        423          42.3            1
## Var4        424          42.4            1
## Var5        163          16.3            1
## Var6        158          15.8            1
```

```
dim(dataENETw[["varSelection"]])
```

```
## [1] 945   3
```

```
# conditions x {variables, interactions} in rows
```

selectionPerSample

- **nLin**: how many of the linear effects are recovered?
- **nInter**: how many of the interaction effects are recovered?
- **all.T1F0**: every simulated effect selected in model?
- **nOthers**: only simulated effects selected (i.e., every other predictor is not selected!)
- final ENET tuning parameters (**alpha** & **lambda**) for every sample

```
head(dataENETw[["selectionPerSample"]])
```

```
##      nLin nInter all.T1F0 nOthers     alpha    lambda idxCondLabel
## [1,]    3      0        0       2 0.2631579 0.7819826            1
## [2,]    2      1        0      13 0.1578947 0.8630043            1
## [3,]    0      0        0       0 0.7368421 0.7336755            1
## [4,]    4      4        1      97 0.0000000 0.6370452            1
## [5,]    0      0        0       0 0.6842105 0.6485952            1
## [6,]    4      4        1      30 0.1052632 0.8984189            1
```

```
dim(dataENETw[["selectionPerSample"]])
```

```
## [1] 9000    7
```

```
# rows: every sample in every condition
```