Appendices

Jeanne van de Put

Contents

Appendices to the master thesis 'A Comparison of Tree Ensemble Methods'.

Author: Jeanne M.M.S. van de Put

Appendix A - Toy Data code

This Appendix contains an overview of the main functions used for the toy data analysis in order of usage:

- `maketoydat()`: function to make the toy data
- `extracttree()`: function to extract rules predicting $Y = 1$ from a tree object; rules eventually extracted from OTE trees are given in Appendix B.
- `nhvarimp()`: subfunction to compute relative variable importances of the ten variables of the toy dataset, as described in Section 2.3.3.1 of the thesis. This subfunction is also used in further analyses (simulation and application).

maketoydat()

```
maketoydat <- function(n, seed1, seed2) {
  set.seed(seed1)
  Xs <- rmvnorm(n, mean=rep(0, 10), sigma=diag(10))
  colnames(Xs) <- c("X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10")

  # make outcomes based on rules of four two-way interaction trees:
  Ta <- Tb <- Tc <- Td <- numeric(n)
  set.seed(seed2)
  for (i in 1:n) {
    if(Xs[i,1] > 0.5 & Xs[i,2] > qnorm(0.625)) {Ta[i] <- rbinom(1,1,0.99)}
    else {Ta[i] <- rbinom(1,1,0.01)}

    if (Xs[i,3] <= -0.5 & Xs[i,4] <= qnorm(0.375)) {Tb[i] <- rbinom(1,1,0.99)}
    else {Tb[i] <- rbinom(1,1,0.01)}

    if (Xs[i,5] > 0.5 & Xs[i,6] > qnorm(0.625)) {Tc[i] <- rbinom(1,1,0.99)}
    else {Tc[i] <- rbinom(1,1,0.01)}

    if (Xs[i,7] <= -0.5 & Xs[i,8] <= qnorm(0.375)) {Td[i] <- rbinom(1,1,0.99)}
    else {Td[i] <- rbinom(1,1,0.01)}
  }

  sumT <- c(mean(Ta), mean(Tb), mean(Tc), mean(Td))
  Y <- Ta+Tb+Tc+Td      # combine tree outcomes
  sumY <- summary(as.factor(Y)) # rule overlap indication
  Y <- 1*(Y>0) # convert all values > 0 to 1, to get 0/1 outcomes

  return(list(X=Xs, Y=Y, sumT=sumT, sumY=sumY))
}
```

extracttree()

```
extracttree <- function(tobj) {
  # counts interactions between subsequently appearing variables
  # and saves the split points

  # Required input:
  ## tobj=tree object of class random forest
```

```
  rulelist <- vector('list', length=length(which(tobj[,'prediction']==2)))

  terminalindex <- as.integer(which(tobj[,'prediction']==2))
  for (rl in 1:length(terminalindex)) {
    # The sequences of variables are found by starting from every terminal node and
    # tracing back all of its ancestors until the initial common ancestor (i.e. the very
    # first split) in the tree is found.
    # To get to the correct order of following the first node until all terminal nodes,
    # the order is later reversed after all nodes constructing a rule are found.
    nodeind <- terminalindex[rl]

    while (nodeind != 1) {  # trace all terminal nodes back to the first split point 1;
      # stop the loop if node 1 is reached

      if (any(tobj[,'left daughter']==nodeind)) {
        parent <- as.integer(which(tobj[,'left daughter']==nodeind))
        rulelist[[rl]] <- c(rulelist[[rl]], paste(tobj[parent,3],"<=",
                                                  round(tobj[parent,4], 3),sep=""))
                           # paste rule consisting of:
                           # 1) variable, 2) smaller/bigger, 3) split point
        nodeind <- parent
      }
      else if (any(tobj[,'right daughter']==nodeind)) {
        parent <- as.integer(which(tobj[,"right daughter"] == nodeind))
        rulelist[[rl]] <- c(rulelist[[rl]], paste(tobj[parent,3],">",
                                                  round(tobj[parent,4],3),sep=""))
        nodeind <- parent
      }
    }
    rulelist[[rl]] <- rev(rulelist[[rl]]) # go from down->top to top->down
    }
  return(rulelist=rulelist)
}
```

nhvarimp()

```
nhvarimp <- function(nhmod, nvar) {
  # subfunction to compute variable importances of node harvest
  ### based on the Variable Importance formula for Rule Ensembles
  ### (Friedman & Popescu, 2005: formula 35)

  vars.per.rule <- nhweights <- numeric(length(nhmod$nodes))
  nhvarindicator <- matrix(0, nrow=nvar, ncol=length(nhmod$nodes))
  for (p in 1:length(nhmod$nodes)) {
    for (l in 1:nvar) { # over all 10 variables
      nhvarindicator[l, p] <- l %in% nhmod$nodes[[p]][,1]
    }
    vars.per.rule[p] <- length(nhmod$nodes[[p]][,1])
    nhweights[p] <- attributes(nhmod$nodes[[p]])$weight
  }

  unscaled <- numeric(length(nvar))
```

```
  for (v in 1:nvar) { # compute unscaled variable importances
    unscaled[v] <- (nhvarindicator[v,] %*% nhweights)/(nhvarindicator[v,] %*%
                                                         vars.per.rule)
  }
  if (any(rowSums(nhvarindicator)==0)) { # avoid NaNs; give importance values of 0 to
    # variables that never occur in nodes
    zero.index <- which(rowSums(nhvarindicator)==0)
    unscaled[zero.index] <- 0
  }

  scaled.varimps <- (unscaled/nvar)/(max(unscaled)/nvar)
  return(scaled.varimps)
}
```

Appendix B - OTE trees

First 5 strongest trees (Tables 1-5) of the OTE ensemble with full trees fitted to the toy data (Section 2.2.3.1).

Table 1: Tree 1

Rule	Splitting	sequence					
1	7≤-0.504	8≤-0.307					
2	7≤-0.504	8>-0.307	3≤-0.467	4≤-0.392			
3	7>-0.504	5>0.534	6>0.318	5≤2.508			
4	7>-0.504	5≤0.534	4≤-0.301	3≤-0.445	6≤1.331		
5	7>-0.504	5≤0.534	4≤-0.301	3≤-0.445	6>1.331		
6	7>-0.504	5>0.534	6≤0.318	3≤-0.446	1≤-1.234		
7	7≤-0.504	8>-0.307	3≤-0.467	4>-0.392	1≤0.661	10≤-1.873	
8	7≤-0.504	8>-0.307	3≤-0.467	4>-0.392	1>0.661	9≤-1.072	
9	7≤-0.504	8>-0.307	3>-0.467	5≤0.24	4>0.734	2>1.593	
10	7≤-0.504	8>-0.307	3>-0.467	5>0.24	9>-0.808	6>0.482	
11	7>-0.504	5≤0.534	4≤-0.301	3>-0.445	2≤0.403	8≤-1.522	
12	7>-0.504	5≤0.534	4>-0.301	1≤0.465	3>1.505	2>0.727	
13	7>-0.504	5≤0.534	4>-0.301	1>0.465	3≤0.38	2>0.162	
14	7>-0.504	5≤0.534	4>-0.301	1>0.465	3>0.38	2>0.315	
15	7>-0.504	5>0.534	6≤0.318	3≤-0.446	1>-1.234	4≤-0.65	
16	7≤-0.504	8>-0.307	3≤-0.467	4>-0.392	1≤0.661	10>-1.873	7≤-1.759
17	7≤-0.504	8>-0.307	3>-0.467	5≤0.24	4>0.734	2≤1.593	3≤-0.318
18	7≤-0.504	8>-0.307	3>-0.467	5>0.24	9>-0.808	6≤0.482	2>0.344
19	7>-0.504	5≤0.534	4≤-0.301	3>-0.445	2>0.403	10≤0.675	8≤-0.983
20	7>-0.504	5≤0.534	4≤-0.301	3>-0.445	2>0.403	10>0.675	1>0.525
21	7>-0.504	5≤0.534	4>-0.301	1>0.465	3≤0.38	2≤0.162	1≤0.482
22	7>-0.504	5>0.534	6≤0.318	3≤-0.446	1>-1.234	4>-0.65	9>1.291
23	7>-0.504	5>0.534	6≤0.318	3>-0.446	2>0.329	3>0.436	1>-0.033
24	7≤-0.504	8>-0.307	3>-0.467	5>0.24	9>-0.808	6≤0.482	2≤0.344
	2≤-1.311						
25	7>-0.504	5≤0.534	4≤-0.301	3>-0.445	2>0.403	10≤0.675	8>-0.983
	2≤0.509						
26	7≤-0.504	8>-0.307	3>-0.467	5>0.24	9>-0.808	6≤0.482	2≤0.344
	2>-1.311	5≤0.271					
27	7>-0.504	5≤0.534	4>-0.301	1>0.465	3≤0.38	2≤0.162	1>0.482
	3≤-0.811	4≤0.002					
28	7>-0.504	5≤0.534	4>-0.301	1≤0.465	3≤1.505	5>-2.047	6>-1.563
	5≤0.483	9≤-0.823	9>-0.837				
29	7>-0.504	5≤0.534	4>-0.301	1>0.465	3≤0.38	2≤0.162	1>0.482
	3≤-0.811	4>0.002	3>-0.833				

Table 2: Tree 2

Rule	Splitting	sequence					
1	3≤-0.448	4≤-0.341	2>-2.674				
2	3>-0.448	5>0.497	6>0.318	10≤-1.477			
3	3>-0.448	5>0.497	6>0.318	10>-1.477			
4	3≤-0.448	4>-0.341	5≤0.566	8≤-0.219	1>0.556		
5	3≤-0.448	4>-0.341	5>0.566	2≤0.785	6>0.141		
6	3≤-0.448	4>-0.341	5>0.566	2>0.785	6>-0.029		
7	3>-0.448	5≤0.497	2>0.389	2≤1.77	1>0.451		
8	3>-0.448	5≤0.497	2>0.389	2>1.77	1>0.525		
9	3>-0.448	5>0.497	6≤0.318	8≤-0.159	7≤-0.474		
10	3≤-0.448	4>-0.341	5≤0.566	8>-0.219	5≤0.12	7>2.506	
11	3>-0.448	5≤0.497	2≤0.389	8≤-0.338	3≤0.562	8>-0.424	
12	3>-0.448	5≤0.497	2≤0.389	8≤-0.338	3>0.562	7≤-0.446	
13	3>-0.448	5≤0.497	2≤0.389	8>-0.338	8>0.681	10≤-2.466	
14	3>-0.448	5>0.497	6≤0.318	8≤-0.159	7>-0.474	1>1.256	
15	3>-0.448	5>0.497	6≤0.318	8>-0.159	5≤0.594	5>0.547	
16	3>-0.448	5>0.497	6≤0.318	8>-0.159	5>0.594	2>1.192	
17	3≤-0.448	4>-0.341	5≤0.566	8≤-0.219	1≤0.556	10>0.037	4≤-0.197
18	3≤-0.448	4>-0.341	5≤0.566	8>-0.219	5≤0.12	7≤2.506	10>2.337
19	3≤-0.448	4>-0.341	5≤0.566	8>-0.219	5>0.12	8>0.445	1>-0.579
20	3>-0.448	5≤0.497	2≤0.389	8≤-0.338	3≤0.562	8≤-0.424	7≤-0.517
21	3>-0.448	5≤0.497	2≤0.389	8≤-0.338	3>0.562	7>-0.446	10≤-1.17
22	3>-0.448	5≤0.497	2≤0.389	8>-0.338	8>0.681	10>-2.466	7>1.796
23	3>-0.448	5≤0.497	2>0.389	2≤1.77	1≤0.451	7≤-0.645	8≤-0.423
24	3>-0.448	5>0.497	6≤0.318	8≤-0.159	7>-0.474	1≤1.256	8>-0.199
25	3>-0.448	5>0.497	6≤0.318	8>-0.159	5>0.594	2≤1.192	8>1.92
26	3≤-0.448	4>-0.341	5≤0.566	8≤-0.219	1≤0.556	10≤0.037	7≤-0.289
	3>-1.727						
27	3>-0.448	5≤0.497	2≤0.389	8≤-0.338	3≤0.562	8≤-0.424	7>-0.517
	2≤-2.116						
28	3>-0.448	5≤0.497	2≤0.389	8≤-0.338	3>0.562	7>-0.446	10>-1.17
	6≤-1.537						
29	3>-0.448	5≤0.497	2>0.389	2≤1.77	1≤0.451	7≤-0.645	8>-0.423
	10>0.924						
30	3>-0.448	5>0.497	6≤0.318	8>-0.159	5>0.594	2≤1.192	8≤1.92
	7≤-1.861						
31	3≤-0.448	4>-0.341	5≤0.566	8>-0.219	5≤0.12	7≤2.506	10≤2.337
	9≤-1.344	2>0.229					
32	3≤-0.448	4>-0.341	5>0.566	2≤0.785	6≤0.141	3>-1.204	4>-0.148
	8≤0.197	9≤0.491					
33	3>-0.448	5≤0.497	2≤0.389	8≤-0.338	3>0.562	7>-0.446	10>-1.17
	6>-1.537	4≤-0.876					
34	3>-0.448	5≤0.497	2≤0.389	8>-0.338	8>0.681	10>-2.466	7≤1.796
	6≤-0.392	6>-0.785					
35	3>-0.448	5≤0.497	2≤0.389	8>-0.338	8>0.681	10>-2.466	7≤1.796
	6>-0.392	3>2.002					
36	3>-0.448	5>0.497	6≤0.318	8>-0.159	5>0.594	2≤1.192	8≤1.92
	7>-1.861	2>0.329	1>0.802				

Table 3: Tree 3

Rule	Splitting	sequence					
1	5≤0.846	4≤-0.362	3≤-0.463				
2	5>0.846	1>0.496	6>-1.056				
3	5≤0.846	4>-0.362	8≤-0.366	7≤-0.493			
4	5>0.846	1≤0.496	2≤0.825	6>0.326			
5	5>0.846	1≤0.496	2>0.825	6>0.058			
6	5≤0.846	4≤-0.362	3>-0.463	8≤-0.306	7≤-0.548		
7	5≤0.846	4≤-0.362	3>-0.463	8>-0.306	10≤-2.405		
8	5≤0.846	4>-0.362	8≤-0.366	7>-0.493	8>-0.394		
9	5>0.846	1>0.496	6≤-1.056	6≤-1.496	10≤0.871		
10	5≤0.846	4≤-0.362	3>-0.463	8≤-0.306	7>-0.548	8≤-1.707	
11	5≤0.846	4>-0.362	8>-0.366	1≤0.343	5>0.494	6>0.346	
12	5≤0.846	4>-0.362	8>-0.366	1>0.343	2>0.36	1≤0.511	
13	5≤0.846	4>-0.362	8>-0.366	1>0.343	2>0.36	1>0.511	
14	5≤0.846	4≤-0.362	3>-0.463	8≤-0.306	7>-0.548	8>-1.707	5>0.634
15	5≤0.846	4≤-0.362	3>-0.463	8>-0.306	10>-2.405	1≤0.905	5>0.734
16	5≤0.846	4>-0.362	8>-0.366	1>0.343	2≤0.36	4>0.834	7>0.405
17	5>0.846	1≤0.496	2≤0.825	6≤0.326	9>-0.858	8>-0.11	6≤-1.989
18	5≤0.846	4≤-0.362	3>-0.463	8>-0.306	10>-2.405	1≤0.905	5≤0.734
	2>1.395						
19	5≤0.846	4≤-0.362	3>-0.463	8>-0.306	10>-2.405	1>0.905	8≤0.837
	7>0.941						
20	5≤0.846	4≤-0.362	3>-0.463	8>-0.306	10>-2.405	1>0.905	8>0.837
	2>-0.383						
21	5≤0.846	4>-0.362	8≤-0.366	7>-0.493	8≤-0.394	1≤0.871	2≤1.308
	6≤-1.625						
22	5≤0.846	4>-0.362	8≤-0.366	7>-0.493	8≤-0.394	1≤0.871	2>1.308
	9≤-0.478						
23	5≤0.846	4>-0.362	8≤-0.366	7>-0.493	8≤-0.394	1>0.871	7≤0.784
	2>0.317						
24	5≤0.846	4>-0.362	8≤-0.366	7>-0.493	8≤-0.394	1>0.871	7>0.784
	2>-0.328						
25	5≤0.846	4>-0.362	8>-0.366	1≤0.343	5≤0.494	7≤1.883	6≤-0.432
	6>-0.442						
26	5≤0.846	4>-0.362	8>-0.366	1>0.343	2≤0.36	4>0.834	7≤0.405
	8≤-0.277						
27	5>0.846	1≤0.496	2≤0.825	6≤0.326	9>-0.858	8≤-0.11	7≤0.11
	9≤-0.519						
28	5>0.846	1≤0.496	2≤0.825	6≤0.326	9>-0.858	8≤-0.11	7≤0.11
	9>-0.519						
29	5>0.846	1≤0.496	2≤0.825	6≤0.326	9>-0.858	8>-0.11	6>-1.989
	9≤-0.694						
30	5≤0.846	4≤-0.362	3>-0.463	8≤-0.306	7>-0.548	8>-1.707	5≤0.634
	3≤-0.063	5≤-1.29					
31	5≤0.846	4≤-0.362	3>-0.463	8≤-0.306	7>-0.548	8>-1.707	5≤0.634
	3>-0.063	5>0.426					
32	5≤0.846	4≤-0.362	3>-0.463	8>-0.306	10>-2.405	1>0.905	8≤0.837
	7≤0.941	2>0.46					
33	5≤0.846	4≤-0.362	3>-0.463	8>-0.306	10>-2.405	1≤0.905	5≤0.734
	2≤1.395	7≤-1.355	7>-1.684				
34	5≤0.846	4>-0.362	8≤-0.366	7>-0.493	8≤-0.394	1≤0.871	2≤1.308
	6>-1.625	5>0.567	6>0.165				

Table 4: Tree 4

Rule	Splitting	sequence					
1	6>0.365	5>0.48	6≤2.226				
2	6≤0.365	4≤-0.302	3≤-0.432	6≤-0.139			
3	6≤0.365	4>-0.302	8≤-0.339	7≤-0.508			
4	6>0.365	5≤0.48	3≤-0.497	4≤-0.438			
5	6>0.365	5>0.48	6>2.226	10>-1.43			
6	6≤0.365	4≤-0.302	3≤-0.432	6>-0.139	5>-1.24		
7	6≤0.365	4≤-0.302	3>-0.432	1≤0.945	8≤-1.883		
8	6≤0.365	4≤-0.302	3>-0.432	1>0.945	2>0.213		
9	6≤0.365	4>-0.302	8≤-0.339	7>-0.508	7>1.928		
10	6≤0.365	4>-0.302	8>-0.339	1≤0.54	7>2.506		
11	6≤0.365	4>-0.302	8>-0.339	1>0.54	2>0.346		
12	6>0.365	5≤0.48	3>-0.497	7≤-0.856	8≤0.216		
13	6>0.365	5≤0.48	3≤-0.497	4>-0.438	9≤-1.014	8≤0.793	
14	6>0.365	5≤0.48	3>-0.497	7≤-0.856	8>0.216	6≤0.647	
15	6≤0.365	4≤-0.302	3>-0.432	1≤0.945	8>-1.883	4>-0.781	10≤-0.768
16	6>0.365	5≤0.48	3≤-0.497	4>-0.438	9>-1.014	8≤-0.363	9>0.548
17	6>0.365	5≤0.48	3>-0.497	7>-0.856	1>0.402	2≤0.346	4>1.234
18	6>0.365	5≤0.48	3>-0.497	7>-0.856	1>0.402	2>0.346	5>-0.996
19	6≤0.365	4≤-0.302	3>-0.432	1≤0.945	8>-1.883	4≤-0.781	9>1.258
	7>0.798						
20	6≤0.365	4≤-0.302	3>-0.432	1≤0.945	8>-1.883	4>-0.781	10>-0.768
	4≤-0.737						
21	6≤0.365	4>-0.302	8≤-0.339	7>-0.508	7≤1.928	2>0.476	3>-0.579
	7≤-0.176						
22	6>0.365	5≤0.48	3≤-0.497	4>-0.438	9>-1.014	8≤-0.363	9≤0.548
	10≤0.068						
23	6>0.365	5≤0.48	3>-0.497	7>-0.856	1>0.402	2≤0.346	4≤1.234
	2≤-2.104						
24	6≤0.365	4>-0.302	8≤-0.339	7>-0.508	7≤1.928	2>0.476	3>-0.579
	7>-0.176	1>0.444					
25	6≤0.365	4>-0.302	8>-0.339	1≤0.54	7≤2.506	7>-1.884	2≤-0.368
	6≤-0.41	6>-0.531					
26	6>0.365	5≤0.48	3>-0.497	7>-0.856	1≤0.402	10≤1.522	6>0.469
	7>1.799	9>0.91					
27	6≤0.365	4≤-0.302	3>-0.432	1≤0.945	8>-1.883	4>-0.781	10>-0.768
	4>-0.737	7>-0.634	4>-0.318				
28	6≤0.365	4>-0.302	8>-0.339	1≤0.54	7≤2.506	7>-1.884	2≤-0.368
	6≤-0.41	6≤-0.531	2>-0.465				
29	6≤0.365	4>-0.302	8>-0.339	1≤0.54	7≤2.506	7>-1.884	2≤-0.368
	6≤-0.41	6≤-0.531	2≤-0.465	7≤-1.239			

Table 5: Tree 5

Rule	Splitting	sequence					
1	1>0.525	2>0.303					
2	1≤0.525	7≤-0.476	8≤-0.32	7≤-0.507			
3	1≤0.525	7>-0.476	3≤-0.443	4≤-0.341			
4	1≤0.525	7≤-0.476	8>-0.32	4≤-0.31	3≤-0.44		
5	1≤0.525	7>-0.476	3≤-0.443	4>-0.341	8≤-2.418		
6	1>0.525	2≤0.303	6≤0.381	8>0.074	3≤-1.493		
7	1>0.525	2≤0.303	6>0.381	5≤0.078	6>1.279		
8	1≤0.525	7>-0.476	3>-0.443	5≤0.664	2>1.588	2≤1.672	
9	1≤0.525	7>-0.476	3>-0.443	5>0.664	6>0.342	5≤2.508	
10	1>0.525	2≤0.303	6≤0.381	8≤0.074	3≤0.231	6>0.068	
11	1>0.525	2≤0.303	6≤0.381	8≤0.074	3>0.231	7≤-0.615	
12	1>0.525	2≤0.303	6>0.381	5>0.078	7>-2.237	10≤0.616	
13	1>0.525	2≤0.303	6>0.381	5>0.078	7>-2.237	10>0.616	
14	1≤0.525	7≤-0.476	8>-0.32	4≤-0.31	3>-0.44	5≤0.215	10≤-2.445
15	1≤0.525	7≤-0.476	8>-0.32	4≤-0.31	3>-0.44	5>0.215	6>-0.559
16	1≤0.525	7≤-0.476	8>-0.32	4>-0.31	5≤0.494	7≤-1.947	7>-1.97
17	1≤0.525	7≤-0.476	8>-0.32	4>-0.31	5>0.494	8≤0.879	6>0.343
18	1≤0.525	7>-0.476	3≤-0.443	4>-0.341	8>-2.418	6>0.286	9≤-1.38
19	1≤0.525	7>-0.476	3>-0.443	5≤0.664	2≤1.588	6>1.966	9>0.184
20	1≤0.525	7>-0.476	3≤-0.443	4>-0.341	8>-2.418	6>0.286	9>-1.38
	5>0.438						
21	1>0.525	2≤0.303	6≤0.381	8≤0.074	3≤0.231	6≤0.068	4≤-0.436
	1>0.885						
22	1>0.525	2≤0.303	6≤0.381	8≤0.074	3≤0.231	6≤0.068	4>-0.436
	9>1.056						
23	1>0.525	2≤0.303	6≤0.381	8>0.074	3>-1.493	1≤2.13	8>1.439
	10>0.675						
24	1≤0.525	7>-0.476	3≤-0.443	4>-0.341	8>-2.418	6>0.286	9>-1.38
	5≤0.438	6≤0.402					
25	1≤0.525	7>-0.476	3>-0.443	5≤0.664	2≤1.588	6≤1.966	5>-1.627
	9≤-1.091	3>1.424					
26	1≤0.525	7>-0.476	3>-0.443	5≤0.664	2≤1.588	6≤1.966	5>-1.627
	9≤-1.091	3≤1.424	4>0.869				
27	1≤0.525	7>-0.476	3>-0.443	5≤0.664	2≤1.588	6≤1.966	5>-1.627
	9>-1.091	8>1.516	5>0.088				

Appendix C - Main Codes Large Simulation

Functions for the large simulation performed for Section 3 to assess predictive and recovery performance of random forests, OTE, node harvest, rule ensembles with RuleFit, and logistic regression in various design settings. The following two functions are displayed:

- `makedatasets()`: Function to make datasets used in the simulation, depending on the design factors and the covariance matrix `Sigma1` (as described in subsection 3.1.2). Returns a list with 100 training sets and 100 test sets for one design cell.
- `SimLarge()`: The function performing the simulation. Requires lists of training and test sets, a grid `rfGrid` with candidate values for choosing the optimal subset size S for random forests and a formula `form` for fitting a logistic regression model (the terms and weights of `form` depend on the design factors). Note that rule ensembles with PRE is not included in this function; a separate simulation with PRE was performed on a PC cluster with a similarly structured function.
- `do.sim()`: The wrapper function that executes the simulation via the `SimLarge()` function. Also constructs a formula `form` for fitting logistic regression models that depend on the design factors (included in `numcombos`).

makedatasets()

```
makedatasets <- function() {
  # Function to automatically create datasets with varying design
  # factors (which are specified in the function).
  # Uses the subfunction `maketraintest()` to make data drawn from
  # underlying logistic regression or tree model, depending on
  # the design factors specified.

  # Input arguments:
  ### i = random seed used
  ### n.train = training set size
  ### n.test = test set size
  ### create.args = arguments to be passed on to data creation function
  ###               (Sigma: covar matrix for underlying distribution variables,
  ###               thrs: thresholds to produce signals, betas
  ###               betas: betas/weights for terms in model)

  # design factors with actual values used:
  matCombos <- as.matrix(expand.grid(c(250, 500, 1000, 5000),
                                     c(0.5, 1),
                                     c(0.1, 0.5)))

  # design factors represented as factors:
  dummyargs <- expand.grid(1:length(unique(matCombos[,1])),
                           1:length(unique(matCombos[,2])),
                           1:length(unique(matCombos[,3])))

  Sigma1 <- matrix(c(1,.5,.0,.0,.0,.0,.0,.0,.0,.0,
                     .5,1,.0,.0,.0,.0,.0,.0,.0,.0,
                     .0,.0,1,.0,.0,.0,.0,.0,.0,.0,
                     .0,.0,.0,1,.0,.0,.0,.0,.5,.0,
                     .0,.0,.0,.0,1,.0,.0,.0,.0,.5,
                     .0,.0,.0,.0,.0,1,.0,.0,.0,.0,
                     .0,.0,.0,.0,.0,.0,1,.5,.0,.0,
```

```
                    .0,.0,.0,.0,.0,.5,1,.0,.0,
                    .0,.0,.0,.5,.0,.0,.0,.0,1,.0,
                    .0,.0,.0,.0,.5,.0,.0,.0,.0,1), nrow=10, byrow=T)

for (i in 1:nrow(matCombos)) {

  datset <- trainsets <- testsets <- vector('list', length=100)
  # random seeds run from 1 to 1600 by specifying seed=((i-1)*100+j)

  # Designs using data drawn from underlying logistic regression model:

  # Nagelkerke's R2=0.5 and prop(Y=1)=0.1
  if (matCombos[i, 2] == 0.5 & matCombos[i, 3] == 0.1) {
    for (j in 1:100) {
      argus <- list(Sigma=Sigma1, thrs=c(-1.95, 1.95),
                    betas = c(4,4,4,4,-4,1), method=2)
      datset[[j]] <- maketraintest(seed=((i-1)*100+j), n.train=matCombos[i, 1],
                                   n.test=250,create.args=argus)
      trainsets[[j]] <- datset[[j]]$trainset
      testsets[[j]] <- datset[[j]]$testset
    }
  }
  # Nagelkerke's R2=0.5 and prop(Y=1)=0.5
  else if (matCombos[i, 2] == 0.5 & matCombos[i, 3] == 0.5) {
    for (j in 1:100) {
      argus <- list(Sigma=Sigma1, thrs=c(-0.9, 0.9),
                    betas = c(3,3,3,3,-3,1), method=2)
      datset[[j]] <- maketraintest(seed=((i-1)*100+j),
                                   n.train=matCombos[i, 1], n.test=250, create.args=argus)
      trainsets[[j]] <- datset[[j]]$trainset
      testsets[[j]] <- datset[[j]]$testset
    }
  }
  # Designs using data drawn from underlying tree model:
  # Nagelkerke's R2=1 and prop(Y=1)=0.1
  else if (matCombos[i, 2] == 1.0 & matCombos[i, 3] == 0.1) {
    for (j in 1:100) {
      argus <- list(Sigma=Sigma1, thrs=c(-1.35, 1.35), method=1)
      datset[[j]] <- maketraintest(seed=((i-1)*100+j), n.train=matCombos[i, 1],
                                   n.test=250, create.args=argus)
      trainsets[[j]] <- datset[[j]]$trainset
      testsets[[j]] <- datset[[j]]$testset
    }
  }
  # Nagelkerke's R2=1 and prop(Y=1)=0.5
  else if (matCombos[i, 2] == 1.0 & matCombos[i, 3] == 0.5) {
    for (j in 1:100) {
      argus <- list(Sigma=Sigma1, thrs=c(-0.5, 0.5), method=1)
      datset[[j]] <- maketraintest(seed=((i-1)*100+j), n.train=matCombos[i, 1],
                                   n.test=250, create.args=argus)
      trainsets[[j]] <- datset[[j]]$trainset
      testsets[[j]] <- datset[[j]]$testset
    }
```

```
    }
    save(trainsets, testsets,
         file=paste("N",dummyargs[i, 1],"er",dummyargs[i, 2],"prop",
                    dummyargs[i, 3],"simdat.Rdata",sep='')) # save dataset object
  }
}
```

SimLarge()

```
SimLarge <- function(trainset, testset, form, rfGrid) {
  # Function to run simulations with. Automatically creates datasets;
  # trains RF, OTE and NH models;
  # extracts variable importance measures from trained models
  # and assessed model performances on test sets.

  # Input arguments:
  ### trainsets = list of training sets, each element is a matrix or
  ###             a data frame with a different training set
  ### testsets = list of test sets, each element is a matrix or
  ###             a data frame with a different test set

  # Returns:
  ### A list with lists of recovery/model performances and other information gathered:
  ### RFlist = a list with:
  ###         Optmtry = a vector with chosen optimal subset size for every
  ###                   replicate in a design cell
  ###         Perfresults = a matrix with model and global recovery performances
  ### OTElist = a list with:
  ###         Ensize = a vector with final ensemble sizes of the fitted OTE models
  ###         Perfresults = a matrix with model and global recovery performances
  ### NHlist = a list wtih:
  ###         Ensize = a vector with final ensemble sizes of the fitted NH models
  ###         Perfresults = a matrix with model and global recovery performances
  ###         nh.interact = a vector with specific recovery performance rates (for interactions)
  ###         nh.lowsplit = a list with split points (saved conditionally
  ###                       on interaction recovery) where threshold value is a lower bound
  ###         nh.upsplit = a list with split points (saved conditionally
  ###                       on interaction recovery) where threshols value is an upper bound
  ### OTElist = a list with:
  ###         Ensize = a vector with final ensemble sizes of the fitted RuleFit models
  ###         Perfresults = a matrix with model and global recovery performances
  ### Logislist = a list with:
  ###         Nagel = a vector with Nagelkerke's R^2 values computed from fitted models
  ###         Perfresults = a matrix with model and global recovery performances


  Nagel <- optmtry <- numeric(length(trainset))

  RFmodelperfs <- OTEmodelperfs <- NHmodelperfs <- Rulemodelperfs <-
    Logismodelperfs <- matrix(0, nrow=length(trainset), ncol=5)
  colnames(RFmodelperfs) <- colnames(OTEmodelperfs) <- colnames(NHmodelperfs) <-
    colnames(Rulemodelperfs) <- colnames(Logismodelperfs) <-
```

```
  c("Acc", "Brier", "AUC", "PressQ", "Detect")

OTEensizes <- NHensizes <- Ruleensizes <- numeric(length=length(trainset))
NHinteract <- numeric(length=length(trainset))

NHlowsplit <- NHupsplit <- vector('list', length=length(trainset))

# Train the pre model:
# (note: random seed is included as argument 'seed' in the pre function)
for (tr in 1:length(trainset)) {
  print(tr)

  set.seed(tr*3)
  caretrf <- train(x=trainset[[tr]][,-1], y=(trainset[[tr]][,1]),
                   method='rf', nodesize=5, tuneGrid=rfGrid)
  optmtry[tr] <- as.numeric(caretrf$bestTune)

  set.seed(tr*tr)
  rfmodel <- randomForest(trainset[[tr]][,-1], trainset[[tr]][,1],
                          nodesize=5, mtry=optmtry[tr])
  set.seed(tr*tr+1)
  otemodel <- ot2(XTraining=trainset[[tr]][,-1], YTraining=trainset[[tr]][,1], ns=5,
                  nf=optmtry, t.initial=500, p=0.1)
  set.seed(tr*tr+tr*2)
  nhmodel <- nodeHarvest(trainset[[tr]][,-1], as.numeric(trainset[[tr]][,1])-1,
                         nodesize=5, maxint=2, nodes=1500)

  set.seed(tr*tr+2)
  rulemodel <- rulefit(trainset[[tr]][,-1], ifelse(trainset[[tr]][,1]==0, -1, 1),
                       max.rules=1500, tree.size=3, rfmode="class", model.type="rules")
  #browser()
  glmmodel <- glm(formula=form, family = binomial, data=trainset[[tr]])

  #### Manipulation check:
  # compute & save Nagelkerke's R^2 from the logistic regression model:
  Nagel[tr] <- round((1-exp((glmmodel$dev-glmmodel$null)/nrow(trainset[[tr]])))/
                       (1-exp(-glmmodel$null/nrow(trainset[[tr]]))), digits=3)

  #### save ensemble sizes ########
  OTEensizes[tr] <- otemodel[[2]]
  NHensizes[tr] <- length(nhmodel$nodes)
  Ruleensizes[tr] <- as.numeric(rulemodel[[1]][[13]])

  #### performance measures #######

  # compute predicted probabilities:
  prob.rf <- predict(rfmodel, newdata=testset[[tr]][,-1], type='prob')
  prob.ote <- Predict.OTProb(otemodel, XTesting=testset[[tr]][,-1],
                             YTesting=testset[[tr]][,1])
  prob.nh <- predict(nhmodel, newdata=testset[[tr]][,-1])
  logodd.rule <- rfpred(testset[[tr]][,-1])
  prob.rule <- 1/(1+exp(-logodd.rule))
  prob.glm <- predict(glmmodel, newdata=testset[[tr]][,-1], type='response')
```

```
# compute Accuracy rate:
RFmodelperfs[tr, "Acc"] <- mean(round(prob.rf[,2])==testset[[tr]]$Y)
OTEmodelperfs[tr, "Acc"] <- mean(round(prob.ote$Estimated.Probabilities)
                                 ==testset[[tr]]$Y)
NHmodelperfs[tr, "Acc"] <- mean(round(prob.nh)==testset[[tr]]$Y)
Rulemodelperfs[tr, "Acc"] <- mean(round(prob.rule)==testset[[tr]]$Y)
Logismodelperfs[tr, "Acc"] <- mean(round(prob.glm)==testset[[tr]]$Y)

# compute Brier score:
numY <- as.numeric(testset[[tr]]$Y)-1
RFmodelperfs[tr, "Brier"] <- mean((numY - prob.rf[,2])^2)
OTEmodelperfs[tr, "Brier"] <- mean((numY - prob.ote$Estimated.Probabilities)^2)
NHmodelperfs[tr, "Brier"] <- mean((numY - prob.nh)^2)
Rulemodelperfs[tr, "Brier"] <- mean((numY - prob.rule)^2)
Logismodelperfs[tr, "Brier"] <- mean((numY - prob.glm)^2)

# compute AUC value:
RFmodelperfs[tr, "AUC"] <- AUCcomp(pred.obj = prob.rf[,2], ytest=testset[[tr]]$Y)
OTEmodelperfs[tr, "AUC"] <- AUCcomp(pred.obj = prob.ote, ytest=testset[[tr]]$Y)
NHmodelperfs[tr, "AUC"] <- AUCcomp(pred.obj = prob.nh, ytest=testset[[tr]]$Y)
Rulemodelperfs[tr, "AUC"] <- AUCcomp(pred.obj = prob.rule, ytest=testset[[tr]]$Y)
Logismodelperfs[tr, "AUC"] <- AUCcomp(pred.obj = prob.glm, ytest=testset[[tr]]$Y)

# compute Press' Q value:
RFmodelperfs[tr, "PressQ"] <- (250 - sum(round(prob.rf[,2])==
                                         testset[[tr]]$Y)*2)^2/(250*(2-1))
OTEmodelperfs[tr, "PressQ"] <- (250 - sum(round(prob.ote$Estimated.Probabilities)==
                                          testset[[tr]]$Y)*2)^2/(250*(2-1))
NHmodelperfs[tr, "PressQ"] <- (250 - sum(round(prob.nh)==
                                         testset[[tr]]$Y)*2)^2/(250*(2-1))
Rulemodelperfs[tr, "PressQ"] <- (250 - sum(round(prob.rule)==
                                           testset[[tr]]$Y)*2)^2/(250*(2-1))
Logismodelperfs[tr, "PressQ"] <- (250 - sum(round(prob.glm)==
                                            testset[[tr]]$Y)*2)^2/(250*(2-1))

############ Recovery performance measures
# Extract variable importance values & detection counts
imprf.info <- importancedetection(rfmodel)
impote.info <- importancedetection(otemodel)
impnh.info <- importancedetection(nhmodel)
imprule.info <- importancedetection(rulemodel)

# Save detection counts:
RFmodelperfs[tr, "Detect"] <- imprf.info$detect
OTEmodelperfs[tr, "Detect"] <- impote.info$detect
NHmodelperfs[tr, "Detect"] <- impnh.info$detect
Rulemodelperfs[tr, "Detect"] <- imprule.info$detect
Logismodelperfs[tr, "Detect"] <- sum(as.numeric(round(summary(glmmodel)$coef[,4],
                                                      digits=4))[c(2:5)]<= 0.1)
                                 # take alpha=0.1 instead of 0.05
                                 # NB: for logistic regression this is
                                 # SPECIFIC interaction recovery performance,
                                 # NOT global recovery performance
```

```
    # Extract specific measures (for node harvest only)
    ## 1) regarding recovery of correct 2-way interactions
    ## 2) regarding split point recovery
    allnode.info <- extractnodeinfo(nhmod = nhmodel)
    NHinteract[[tr]] <- allnode.info$trueint
    NHlowsplit[[tr]] <- allnode.info$lows
    NHupsplit[[tr]] <- allnode.info$ups
  }

  ##### save all results per method:
  RFlist <- list(Optmtry=optmtry, Perfresults=RFmodelperfs)
  OTElist <- list(Ensize=OTEensizes, Perfresults=OTEmodelperfs)
  NHlist <- list(Ensize=NHensizes, Perfresults=NHmodelperfs, nh.interact=NHinteract,
                 nh.lowsplit=NHlowsplit, nh.upsplit=NHupsplit)
  Rulelist <- list(Ensize=Ruleensizes, Perfresults=Rulemodelperfs)
  Logislist <- list(Nagel=Nagel, Perfresults=Logismodelperfs)

  ################### return the required values:
  return(list(RF=RFlist, OTE=OTElist, NH=NHlist, Rule=Rulelist, Logis=Logislist))
}
```

do.sim()

```
do.sim <- function() {
  numcombos <- as.matrix(expand.grid(1:4, 1:2, 1:2))
  colnames(numcombos) <- c("N", "er", "prop")

  # candidate RF subset size values for caret to choose from
  rfGrid <- expand.grid(mtry=c(1, floor(sqrt(10)), 10/2, 8, 10))

  # make threshold values for interaction terms for fitting logistic regression model:
  thrs <- matrix(c(-1.95, 1.95, -0.9, 0.9, -1.35, 1.35, -0.5, 0.5),
                 ncol=2, nrow=4, byrow=TRUE)

  for (com in 1:nrow(numcombos)) {
    print(com)
    # Load the data:
    load(paste("N",numcombos[com, "N"],"er",numcombos[com, "er"],"prop",
               numcombos[com, "prop"],"simdat.Rdata",sep=''))
    # Make formula for fitting logistic regression model:
    if (numcombos[com, 'er']==1) {
      # formula for data based on glm model (with anti-term)
      if (numcombos[com, 'prop']==1) {
        form <- Y~I(X1>thrs[1, 2]&X2>thrs[1, 2]) + I(X9<=thrs[1, 1]&X4<=thrs[1, 1])+
          I(X5>thrs[1, 2]&X10>thrs[1, 2])+I(X7<=thrs[1, 1]&X8<=thrs[1, 1])+
          #anti-term:
          I(X1<=thrs[1, 2]&X2<=thrs[1, 2]&X9>thrs[1, 1]&X4>thrs[1, 1]&
              X5<=thrs[1, 2]&X10<=thrs[1, 2]&X7>thrs[1, 1]&X8>thrs[1, 1])
      } else {
        form <- Y~I(X1>thrs[2, 2]&X2>thrs[2, 2]) + I(X9<=thrs[2, 1]&X4<=thrs[2, 1])+
          I(X5>thrs[2, 2]&X10>thrs[2, 2])+I(X7<=thrs[2, 1]&X8<=thrs[2, 1])+
```

```
          # anti-term:
          I(X1<=thrs[2, 2]&X2<=thrs[2, 2]&X9>thrs[2, 1]&X4>thrs[2, 1]&
              X5<=thrs[2, 2]&X10<=thrs[2, 2]&X7>thrs[2, 1]&X8>thrs[2, 1])
      }
    } else {
      # formula for tree-based data
      if (numcombos[com, 'prop']==1) {
        form <- Y~I(X1>thrs[3, 2]&X2>thrs[3, 2]) + I(X9<=thrs[3, 1]&X4<=thrs[3, 1])+
          I(X5>thrs[3, 2]&X10>thrs[3, 2]) + I(X7<=thrs[3, 1]&X8<=thrs[3, 1])
      } else {
        form <- Y~I(X1>thrs[4, 2]&X2>thrs[4, 2]) + I(X9<=thrs[4, 1]&X4<=thrs[4, 1])+
          I(X5>thrs[4, 2]&X10>thrs[4, 2])+I(X7<=thrs[4, 1]&X8<=thrs[4, 1])
      }
    }
    # Do the simulations:
    largesimobj <- SimLarge(trainset=trainsets, testset=testsets,
                            rfGrid=rfGrid, form=form)
    # Save results object:
    save(largesimobj, file=paste("N",numcombos[com, 1],"er",
                                 numcombos[com, 2],
                                 "prop",numcombos[com, 3],
                                 "results_largesim.Rdata",sep=''))
  }
}
```

Appendix D - Simulation Results

Secondary simulation results, including:

- Manipulation check: average Nagelkerke's R^2 values computed from logistic regression models fitted to the simulation training sets.
- Most frequently chosen random forest optimal subset size (through training) per design cell.
- Rounded average ensemble sizes of final models (OTE, node harvest, RuleFit, and PRE)
- Accuracy rates (all methods)
- Brier scores (all methods)
- AUC values (all methods)

Table 6: Manipulation check: Average Nagelkerke's R^2 value per design cell, computed from fitted logistic regression models from the simulation.

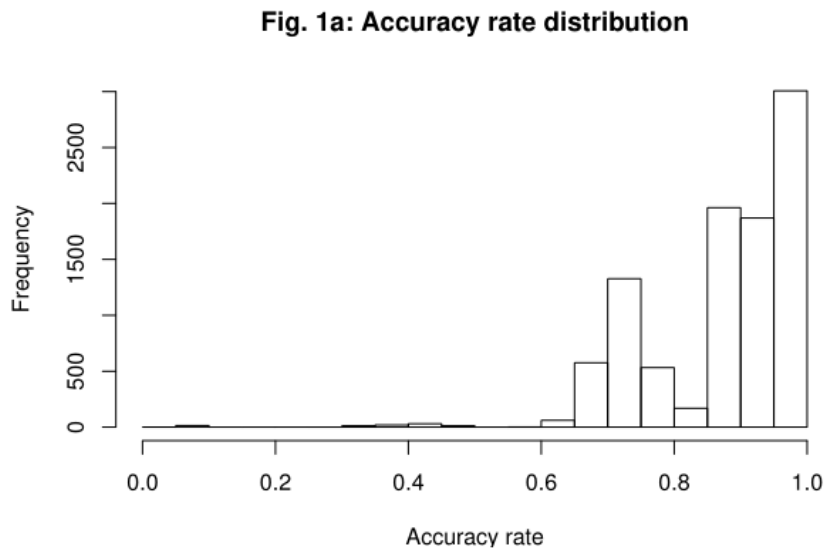
	$P(Y = 1) = .1$		$P(Y = 1) = .5$	
n_{train}	$1 - error = .5$	$1 - error = 1.0$	$1 - error = .5$	$1 - error = 1.0$
250	.521	1.000	.523	1.000
500	.513	1.000	.514	1.000
1000	.500	1.000	.512	1.000
5000	.503	1.000	.510	1.000

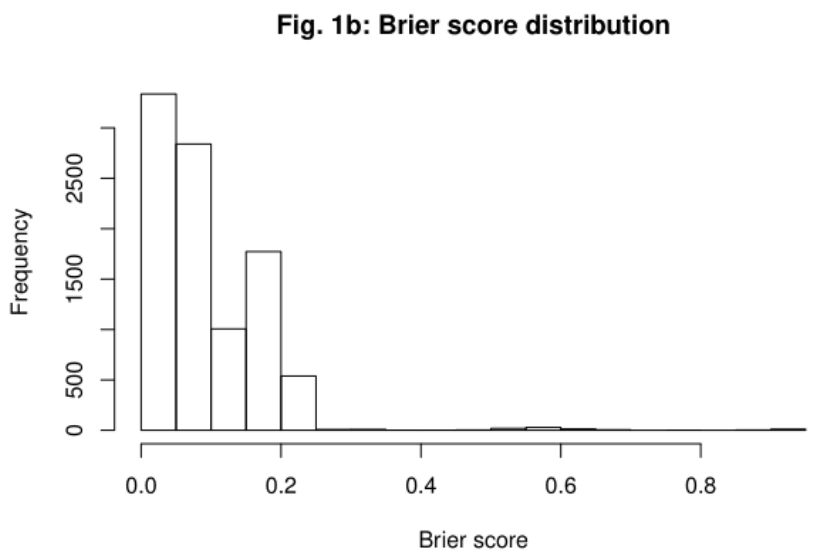
Table 7: Most frequently chosen optimal subset size S by 'caret' for random forests per design cell.

	$P(Y = 1) = .1$		$P(Y = 1) = .5$	
n_{train}	$1 - error = .5$	$1 - error = 1.0$	$1 - error = .5$	$1 - error = 1.0$
250	1	3 and 5	1 and 3	5
500	3	8	3	5
1000	3	8	1	5
5000	3	5	1	5

Table 8: Rounded averaged ensemble sizes per design cell from the methods where ensemble sizes are reduced.

		n_{train}				
$P(Y = 1)$	$1 - error$	Method	250	500	1000	5000
.1	.5	OTE	25	26	27	31
		Node Harvest	33	40	43	41
		Rule Ensembles (RuleFit)	18	19	26	37
		Rule Ensembles (PRE)	20	31	39	65
	1.0	OTE	26	27	27	27
		Node Harvest	36	39	38	39
		Rule Ensembles (RuleFit)	80	108	74	43
		Rule Ensembles (PRE)	23	31	33	48
.5	.5	OTE	26	29	30	33
		Node Harvest	57	40	42	51
		Rule Ensembles (RuleFit)	21	29	33	89
		Rule Ensembles (PRE)	26	40	66	148
	1.0	OTE	26	27	27	29
		Node Harvest	35	41	45	54
		Rule Ensembles (RuleFit)	56	43	42	45
		Rule Ensembles (PRE)	29	32	38	63





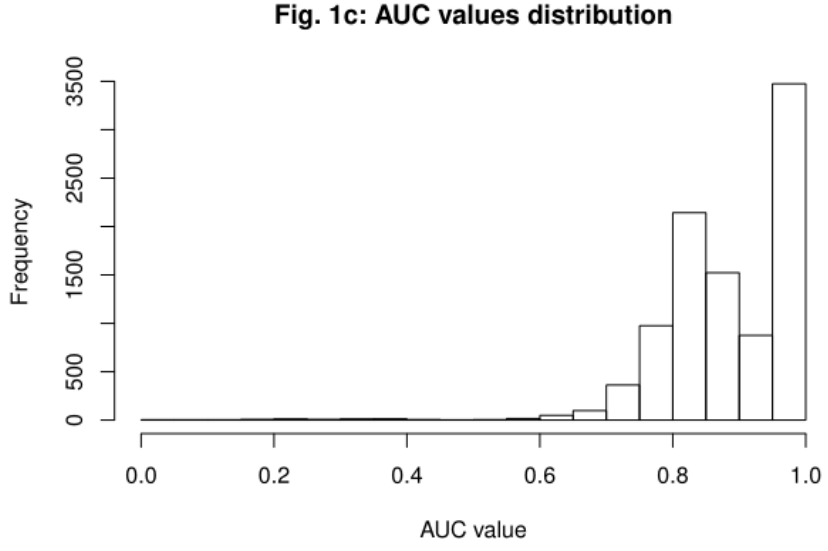


Figure 1: Histograms of all model performances (accuracy rates (1a), Brier scores (1b), AUC values (1c)) of all methods together, including outliers.

Table 9: Cell-averaged accuracy rate values per method (with corresponding standard deviations) after outlier removal, rounded to 3 digits.

$P(Y=1)$	$1-error$	Method	n_{train}			
			250	500	1000	5000
.1	.5	Random Forests	.890(0.019)	.895(0.017)	.897(0.019)	.899(0.021)
		OTE	.890(0.020)	.889(0.018)	.897(0.018)	.898(0.022)
		Node Harvest	.883(0.020)	.886(0.020)	.886(0.022)	.887(0.022)
		Rule Ensembles (RuleFit)	.889(0.019)	.895(0.017)	.899(0.019)	.899(0.021)
		Rule Ensembles (PRE)	.875(0.022)	.886(0.019)	.897(0.019)	.898(0.021)
		Logistic Regression	.897(0.019)	.897(0.019)	.900(0.019)	.901(0.021)
	1.0	Random Forests	.937(0.025)	.966(0.018)	.978(0.015)	.961(0.018)
		OTE	.947(0.024)	.972(0.017)	.981(0.015)	.961(0.018)
		Node Harvest	.934(0.021)	.940(0.020)	.925(0.021)	.910(0.019)
		Rule Ensembles (RuleFit)	.948(0.026)	.978(0.017)	.985(0.016)	.989(0.018)
		Rule Ensembles (PRE)	.933(0.021)	.974(0.017)	.984(0.016)	.989(0.018)
		Logistic Regression	.993(0.014)	.989(0.017)	.990(0.016)	.990(0.018)
.5	.5	Random Forests	.709(0.030)	.721(0.029)	.734(0.028)	.732(0.029)
		OTE	.698(0.032)	.717(0.030)	.730(0.028)	.725(0.027)
		Node Harvest	.679(0.040)	.705(0.031)	.727(0.029)	.730(0.026)
		Rule Ensembles (RuleFit)	.704(0.032)	.722(0.031)	.736(0.029)	.730(0.028)
		Rule Ensembles (PRE)	.702(0.035)	.722(0.029)	.733(0.029)	.726(0.025)
		Logistic Regression	.732(0.028)	.728(0.028)	.734(0.029)	.731(0.028)
	1.0	Random Forests	.908(0.045)	.940(0.047)	.958(0.053)	.950(0.067)
		OTE	.915(0.044)	.941(0.047)	.959(0.053)	.950(0.067)
		Node Harvest	.854(0.051)	.888(0.060)	.917(0.058)	.945(0.066)
		Rule Ensembles (RuleFit)	.947(0.046)	.963(0.048)	.967(0.55)	.951(0.067)
		Rule Ensembles (PRE)	.943(0.048)	.961(0.048)	.967(0.053)	.951(0.067)
		Logistic Regression	.973(0.049)	.972(0.049)	.972(0.054)	.952(0.067)

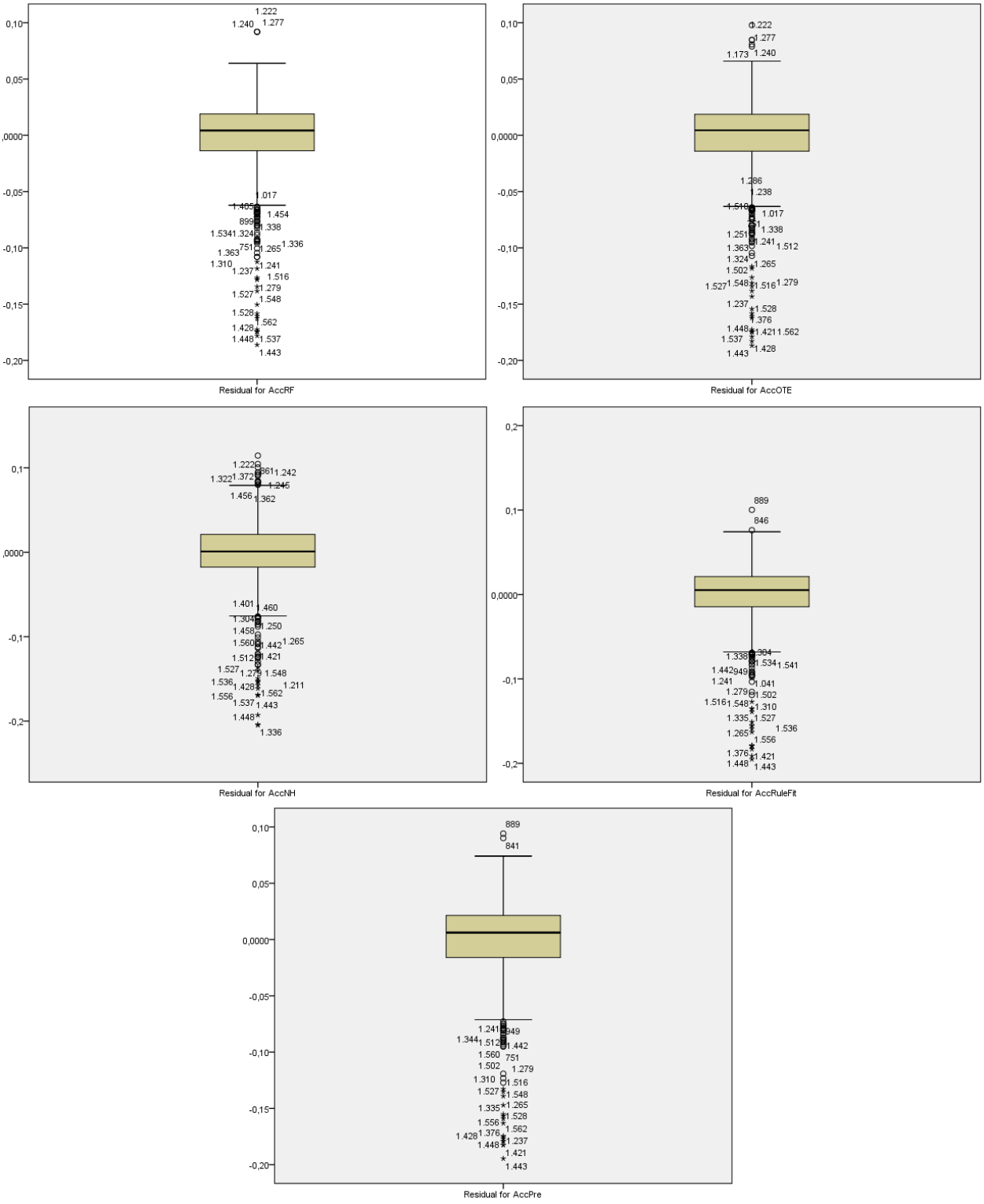


Figure 2: Boxplots of the residuals of the fitted accuracy rates per method (RF=random forests, OTE=optimal trees ensembles, NH=node harvest, RuleFit=rule ensembles with RuleFit, PRE=rule ensembles with PRE); all distributions significantly deviate from a normal distribution (all Shapiro-Wilk p values $\approx .00$).

Table 10: Cell-averaged Brier score values per method (with corresponding standard deviations) after outlier removal, rounded to 3 digits.

$P(Y=1)$	$1-error$	Method	n_{train}			
			250	500	1000	5000
.1	.5	Random Forests	.080(0.013)	.075(0.011)	.072(0.011)	.066(0.012)
		OTE	.086(0.015)	.083(0.013)	.074(0.012)	.067(0.012)
		Node Harvest	.092(0.014)	.089(0.014)	.087(0.015)	.083(0.014)
		Rule Ensembles (RuleFit)	.086(0.015)	.077(0.013)	.071(0.012)	.065(0.012)
		Rule Ensembles (PRE)	.101(0.016)	.086(0.014)	.073(0.013)	.066(0.012)
		Logistic Regression	.064(0.011)	.063(0.010)	.062(0.010)	.061(0.011)
	1.0	Random Forests	.048(0.015)	.029(0.013)	.019(0.012)	.012(0.016)
		OTE	.040(0.016)	.024(0.014)	.018(0.013)	.012(0.017)
		Node Harvest	.049(0.015)	.043(0.012)	.045(0.011)	.045(0.008)
		Rule Ensembles (RuleFit)	.043(0.019)	.020(0.016)	.014(0.015)	.011(0.016)
		Rule Ensembles (PRE)	.050(0.015)	.023(0.015)	.015(0.015)	.011(0.018)
		Logistic Regression	.007(0.014)	.011(0.017)	.010(0.016)	.010(0.018)
.5	.5	Random Forests	.193(0.012)	.181(0.011)	.170(0.011)	.161(0.011)
		OTE	.197(0.014)	.185(0.012)	.171(0.013)	.165(0.013)
		Node Harvest	.212(0.010)	.206(0.007)	.201(0.006)	.201(0.005)
		Rule Ensembles (RuleFit)	.195(0.016)	.180(0.014)	.168(0.012)	.162(0.012)
		Rule Ensembles (PRE)	.198(0.018)	.183(0.016)	.171(0.014)	.162(0.012)
		Logistic Regression	.153(0.012)	.152(0.012)	.148(0.011)	.149(0.011)
	1.0	Random Forests	.096(0.030)	.066(0.033)	.077(0.044)	.075(0.063)
		OTE	.085(0.032)	.062(0.037)	.074(0.047)	.076(0.065)
		Node Harvest	.153(0.014)	.150(0.014)	.162(0.017)	.156(0.011)
		Rule Ensembles (RuleFit)	.062(0.046)	.041(0.047)	.067(0.054)	.076(0.067)
		Rule Ensembles (PRE)	.063(0.046)	.042(0.047)	.066(0.053)	.076(0.067)
		Logistic Regression	.040(0.049)	.033(0.049)	.062(0.054)	.075(0.067)

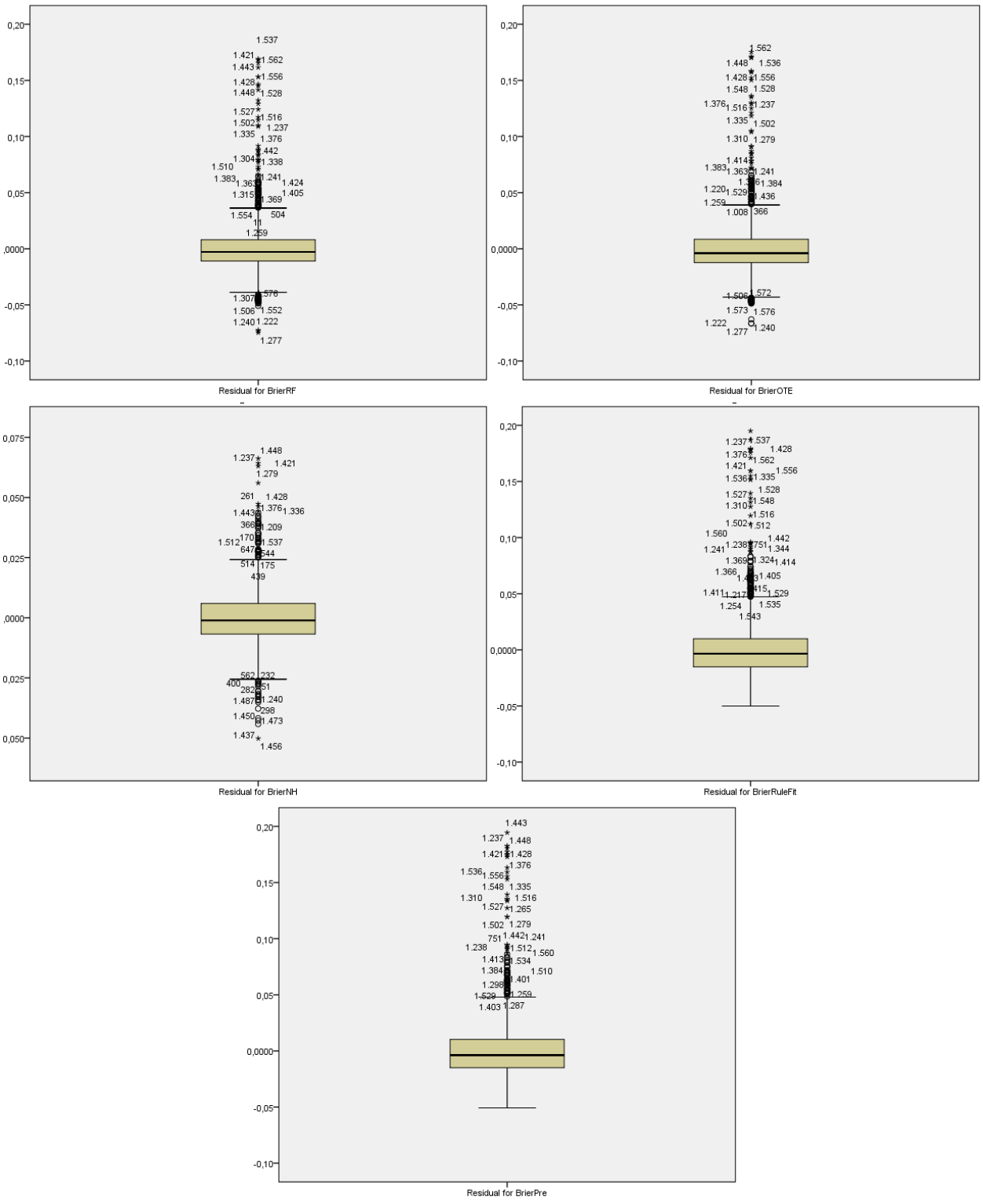


Figure 3: Boxplots of the residuals of the fitted Brier scores per method; all distributions significantly deviate from a normal distribution (all Shapiro-Wilk p values $\approx .00$).

Table 11: Cell-averaged AUC values per design cell per method (with corresponding standard deviations) after outlier removal, rounded to 3 digits.

$P(Y=1)$	$1-error$	Method	n_{train}			
			250	500	1000	5000
.1	.5	Random Forests	.833(0.050)	.850(0.041)	.862(0.040)	.870(0.039)
		OTE	.807(0.057)	.829(0.043)	.849(0.044)	.865(0.043)
		Node Harvest	.727(0.062)	.764(0.063)	.796(0.050)	.847(0.041)
		Rule Ensembles (RuleFit)	.781(0.061)	.834(0.048)	.858(0.043)	.872(0.037)
		Rule Ensembles (PRE)	.707(0.065)	.805(0.051)	.851(0.042)	.872(0.039)
		Logistic Regression	.861(0.038)	.863(0.041)	.863(0.037)	.871(0.032)
	1.0	Random Forests	.948(0.042)	.960(0.057)	.976(0.047)	.973(0.060)
		OTE	.941(0.056)	.957(0.061)	.973(0.053)	.970(0.065)
		Node Harvest	.913(0.077)	.928(0.077)	.952(0.058)	.969(0.067)
		Rule Ensembles (RuleFit)	.930(0.057)	.957(0.062)	.973(0.051)	.970(0.067)
		Rule Ensembles (PRE)	.936(0.056)	.959(0.063)	.972(0.053)	.972(0.058)
		Logistic Regression	.982(0.066)	.963(0.096)	.984(0.058)	.979(0.080)
.5	.5	Random Forests	.783(0.030)	.809(0.026)	.831(0.025)	.837(0.025)
		OTE	.768(0.032)	.795(0.027)	.823(0.026)	.832(0.025)
		Node Harvest	.748(0.041)	.784(0.029)	.819(0.029)	.836(0.024)
		Rule Ensembles (RuleFit)	.779(0.033)	.809(0.028)	.831(0.025)	.837(0.023)
		Rule Ensembles (PRE)	.774(0.035)	.805(0.029)	.829(0.027)	.837(0.024)
		Logistic Regression	.837(0.022)	.839(0.021)	.843(0.020)	.840(0.020)
	1.0	Random Forests	.962(0.039)	.974(0.039)	.978(0.043)	.958(0.063)
		OTE	.964(0.038)	.973(0.040)	.977(0.044)	.955(0.066)
		Node Harvest	.928(0.042)	.948(0.043)	.967(0.043)	.958(0.061)
		Rule Ensembles (RuleFit)	.972(0.040)	.977(0.039)	.979(0.043)	.959(0.063)
		Rule Ensembles (PRE)	.967(0.042)	.975(0.038)	.979(0.042)	.959(0.063)
		Logistic Regression	.977(0.057)	.982(0.049)	.978(0.066)	.960(0.063)

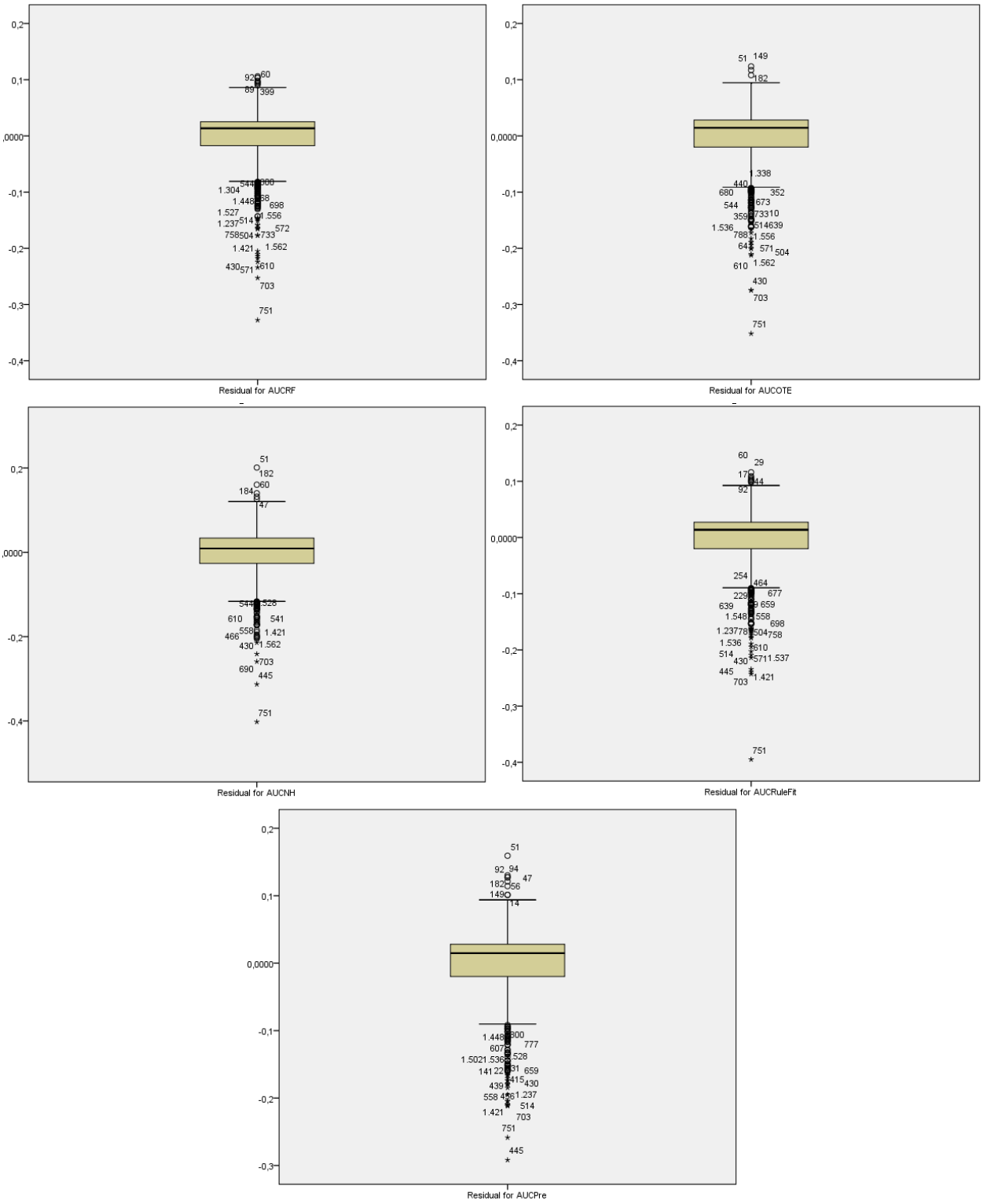


Figure 4: Boxplots of the residuals of the fitted AUC values per method; all distributions significantly deviate from a normal distribution (all Shapiro-Wilk p values $\approx .00$).

Appendix E - MRI application codes

Contains main scripts for the application study on the MRI data (in order of usage):

- `cvreps()`: function to perform repeated k-fold cross-validation with random forests, OTE, node harvest and rule ensembles through RuleFit.
- `ModelSpec()`: function to compute sensitivity and specificity of a model at every produced cutpoint (code kindly provided by E. Dusseldorp).

cvreps()

```
cvreps <- function(kfold, reps, dat, method, yvar, optmtry=NULL, prop=NULL) {
  # Function to do repeated k-fold cross-validation on a dataset with methods
  # Random forests, Optimal Tree Ensembles (OTE), Rule Ensembles (via RuleFit)
  # and Node Harvest.

  # Inputs:
  ## kfold = number of folds for CV
  ## reps = number of repeats of CV procedure
  ## dat = dataset
  ## method: specify 'rf' for random forest, 'ote' for OTE, 're' for RuleFit/rule
  ##          ensembles or 'nh' for NodeHarvest.
  ## yvar = specification of position outcome in dataset
  ## optmtry: optimized mtry value. Specify for Random Forests or OTE
  ##          (set to NULL for other methods)
  ## prop: proportion of OTE trees that will form initial OTE enseble
  ##    (default = 0.2; set to NULL for other methods)

  N <- nrow(dat)

  # distribution of accuracy values
  accs <- AUCs <- Briers <- Pressq <- sensi <- speci <- numeric(reps)
  predprobs <- cutoffs <- vector('list', length=reps)

  if(method=='rf') {

    for(i in 1:reps) {
      set.seed(i)
      nums <- sample(c(1:N), replace=FALSE)
      folds <- vector('list', length=kfold)

      for (k in 1:kfold) {
        folds[[k]] <- nums[((k-1)*round(N/kfold)+1):(k*round(N/kfold))]
      }
      rest <- nums[(k*round(N/kfold)+1):length(nums)]
      for (j in 1:length(rest)) {
        folds[[j]] <- c(folds[[j]], rest[j])
      }

      predictedvals <- vector('list', length=kfold)

      unorderedpred <- NULL
      for(k in 1:kfold) {
```

```
      cat('rep is ',i,' fold is',k,"\n")
      train <- dat[-(folds[[k]]), ]
      test <- dat[folds[[k]], ]

      set.seed(k + i)
      rfmodel <- randomForest(x=train[,-yvar], y=train[,yvar], mtry=optmtry,
                              nodesize=5)

      predictedvals[[k]] <- predict(rfmodel, newdat=test, type = 'prob')
      unorderedpred <- c(unorderedpred, predictedvals[[k]][,2])
    }

    ypred <- unorderedpred[order(as.numeric(names(unorderedpred)))]
    predprobs[[i]] <- ypred
    accs[i] <- mean(round(ypred) == dat[,yvar])
    AUCs[i] <- AUCcomp(ypred, ytest=dat[,yvar])
    Briers[i] <- mean(((as.numeric(dat[,yvar])-1) - ypred)^2)
    Pressq[i] <- (N-sum(round(ypred)==dat[,yvar])*2)^2 / N
    cutoffs[[i]] <- Modelspec(phat=ypred, group=(as.numeric(dat[,yvar])-1))
  }
  return(list(opthyper=optmtry, predprobs=predprobs,
              Acc=accs, AUC=AUCs, Brier=Briers, Pressq=Pressq, Cutoff=cutoffs))
}

if(method=='ote') {
  for(i in 1:reps) {
    print(i)
    set.seed(i)
    nums <- sample(c(1:N), replace=FALSE)
    folds <- vector('list', length=kfold)

    for (k in 1:kfold) {
      folds[[k]] <- nums[((k-1)*round(N/kfold)+1):(k*round(N/kfold))]
    }
    rest <- nums[(k*round(N/kfold)+1):length(nums)]
    for (j in 1:length(rest)) {
      folds[[j]] <- c(folds[[j]], rest[j])
    }

    unordpredote <- NULL
    unordprobote <- NULL

    for (k in 1:length(folds)) {
      cat('rep is ',i,' fold is',k,"\n")
      train <- defdat.fram[-(folds[[k]]), ]
      test <- defdat.fram[folds[[k]], ]

      set.seed(k + i)
      otemod <- ot2(XTraining=train[,-yvar], YTraining=train[,yvar], t.initial=500,
                    nf=optmtry, ns=5, p=prop)

      unordpredote <- c(unordpredote,
                        Predict.OTClass(otemod, XTesting=test[,-yvar],
```

```
                                        YTesting=test$label)$Predicted.Class.Labels)
      unordprobote <- c(unordprobote,
                        Predict.OTProb(otemod, XTesting=test[,-yvar],
                                       YTesting=(as.numeric(test$label)-1))$
                          Estimated.Probabilities)
    }
    Ypred.ote <- unordpredote[order(as.numeric(names(unordpredote)))]
    Ypred.ote <- as.factor(as.numeric(Ypred.ote)-1)
    Yprob.ote <- unordprobote[order(as.numeric(names(unordprobote)))]
    predprobs[[i]] <- Yprob.ote
    accs[i] <- mean(Ypred.ote==dat[,yvar])
    AUCs[i] <- AUCcomp(Yprob.ote, ytest=dat[,yvar])
    Briers[i] <- mean(((as.numeric(dat[,yvar])-1) - Yprob.ote)^2)
    Pressq[i] <- (N-sum(Ypred.ote==dat[,yvar])*2)^2 / N
    cutoffs[[i]] <- Modelspec(phat=Yprob.ote, group=(as.numeric(dat[,yvar])-1))
  }
  return(list(predprobs=predprobs, Acc=accs, AUC=AUCs, Brier=Briers,
              Pressq=Pressq, Cutoff=cutoffs))
}

if (method == 'nh') {
  for(i in 1:reps) {
    set.seed(i)
    nums <- sample(c(1:N), replace=FALSE)
    folds <- vector('list', length=kfold)

    for (k in 1:kfold) {
      folds[[k]] <- nums[((k-1)*round(N/kfold)+1):(k*round(N/kfold))]
    }
    rest <- nums[(k*round(N/kfold)+1):length(nums)]
    for (j in 1:length(rest)) {
      folds[[j]] <- c(folds[[j]], rest[j])
    }

    prednh <- numeric(N)

    for (m in 1:length(folds)) { # use m as iteration variable here because
      # k is already used in subfunction extractnodeinfo
      # browser()
      cat('rep is ',i,', fold is',m,"\n")
      Xtrain <- dat[-(folds[[m]]), -yvar]
      Ytrain <- as.numeric(dat[-(folds[[m]]),yvar])-1

      Xtest <- dat[folds[[m]], -yvar]

      set.seed(m + i)
      nh1 <- nodeHarvest(X=Xtrain, Y=Ytrain, nodes=1000, maxinter=2, nodesize=5)
      set.seed((m+1)^2 + i^2)
      nhmod <- nodeHarvest(X=Xtrain, Y=Ytrain, nodes=1000, maxinter=2,
                           nodesize=5, addto=nh1)
      prednh[folds[[m]]] <- predict(nhmod, newdata=Xtest)
    }
    predprobs[[i]] <- prednh
```

```
    accs[i] <- mean(round(prednh) == dat[,yvar])
    AUCs[i] <- AUCcomp(prednh, ytest=dat[,yvar])
    Briers[i] <- mean(((as.numeric(dat[,yvar])-1) - prednh)^2)
    Pressq[i] <- (N-sum(round(prednh)==dat[,yvar])*2)^2 / N
    cutoffs[[i]] <- Modelspec(phat=prednh, group=(as.numeric(dat[,yvar])-1))
  }
  return(list(predprobs=predprobs, Acc=accs, AUC=AUCs, Brier=Briers, Pressq=Pressq,
              Cutoff=cutoffs))
}


if(method=='re') {
  for(i in 1:reps) {
    set.seed(i)
    nums <- sample(c(1:N), replace=FALSE)
    folds <- vector('list', length=kfold)

    for (k in 1:kfold) {
      folds[[k]] <- nums[((k-1)*round(N/kfold)+1):(k*round(N/kfold))]
    }
    rest <- nums[(k*round(N/kfold)+1):length(nums)]
    for (j in 1:length(rest)) {
      folds[[j]] <- c(folds[[j]], rest[j])
    }

    ytrue <- ifelse(dat[,yvar]==0, -1, 1) # convert to -1/1
    predictedvals <- vector('list', length=kfold)
    orderedpred <- numeric(N)

    for(k in 1:kfold) {
      cat('rep is ',i,' fold is',k,"\n")

      xtrain <- dat[-(folds[[k]]), -yvar]
      ytrain <- ytrue[-(folds[[k]])]
      xtest <- dat[folds[[k]], -yvar]

      set.seed(k + i)
      rulemodel <- rulefit(xtrain, ytrain, tree.size=4, max.rules=2000,
                           rfmode="class", model.type="rules", cat.vars=c(2,3))
      logoddsvals <- rfpred(xtest)
      predictedvals <- 1/(1+exp(-logoddsvals))
      orderedpred[folds[[k]]] <- predictedvals
    }
    ypred <- orderedpred
    predprobs[[i]] <- ypred
    accs[i] <- mean(round(ypred) == dat[,yvar])
    AUCs[i] <- AUCcomp(ypred, ytest=dat[,yvar])
    Briers[i] <- mean(((as.numeric(dat[,yvar])-1) - ypred)^2)
    Pressq[i] <- (N-sum(round(ypred)==dat[,yvar])*2)^2 / N
    cutoffs[[i]] <- Modelspec(phat=ypred, group=(as.numeric(dat[,yvar])-1))
  }
  return(list(predprobs=predprobs, Acc=accs, AUC=AUCs,
              Brier=Briers, Pressq=Pressq, Cutoff=cutoffs))
```

```
  }

}
```

Modelspec()

```
Modelspec<-function(phat,group,select=TRUE){
  # (code from Elise Dusseldorp)
  #
  #computes modelfit for several cut off values from predicted values
  #Be aware: group: 1=case, 0=control
  cutpoints<-sort(unique( phat[!is.na(phat)] ))
  summary(as.factor(phat))
  #various cut-off values possible;
  restab<-as.data.frame(matrix(0,nrow=(length(cutpoints)-1),ncol=7))
  for (i in 1:(length(cutpoints)-1) ){
    pred<-ifelse(phat> cutpoints[i],1,0)
    tab<-CrossTable(-pred,-group); #case has to be lin left column,
    # control in right column (as customary)
    res<-round(lrpos(tab$t[1,1],tab$t[1,2],tab$t[2,1],tab$t[2,2]),dig=2)
    restab[i,1]<-sum(tab$t);
    restab[i,2:3]<-res[2:1]*100;
    restab[i,4:6]<-res[3:5]
    restab[i,7]<-cutpoints[i]
  }
  if(select==TRUE){
    vec<-numeric(nrow(restab))
    for(i in 2:(length(vec))){
      vec[i]<-ifelse(restab[i,2]==restab[i-1,2],0,1)
    }
    vec[1]<-1
    restab<-restab[vec==1,]
  }
  names(restab)<-c("N","Spec","Sens","LR+","CIlower","CIupper","cutpoint")
  return(restab)
}
```