Replication scripts - Parameter tuning

1 Introduction

All analyses were performed in `R` (version 4.1.0, R Core Team, 2021) We fitted (penalized) logistic regression models using `R` package `glmnet` (version 4.1.2, Friedman et al., 2010); generalized additive models (GAMs) with smoothing splines suing pacakge `mgcv` (version 1.8.35, Wood, 2017); conditional inference trees using package `partykit` (Hothorn et al., 2006; version 1.2.13, Hothorn & Zeileis, 2015); gradient boosted tree ensembles using package `gbm` (version 2.1.8, Greenwell et al., 2020); random forests using package `ranger` (version 0.13.1, Wright & Ziegler, 2017); prediction rule ensembles using package `pre` (version 1.0.1, Fokkema, 2020); k nearest neighbours using package `class` (version 7.3.19, Venables & Ripley, 2002).

We tuned the model-fitting parameters for all models using resampling and cross validation (CV) on the training dataset. For tuning the parameters of random forests, boosted tree ensembles and prediction rule ensembles, we used package `caret` (version 6.0.89, Kuhn, 2021). For tuning the parameters of conditional inference trees and k nearest neighbours, we wrote custom code; we tuned the penalized regression models using function `cv.glmnet` from package `glmnet`.

We did not tune the parameters of the GAMs with smoothing splines, because we expected the defaults to work well out of the box. We simply employed the defaults of thin-plate regression splines and generalized cross validation method to fit the smoothing splines. This provides built-in regularization of the wigglyness, without the need for choosing an optimal value for penalty or tuning parameters. For smaller samples, restricted maximum likelihood (REML) would be preferred. For the analyses using the items as predictors, we set the number of basis functions of the thin-plate regression splines to four, instead of the default of eight, because the item responses have only five possible values; increasing the number of basis functions would yield an unidentified model. Furthermore, because of the larger number of predictors, and the benefit of penalization observed in the logistic regression, we also applied a penalty to the linear part of each smoothing spline function, so that some items can be completely eliminated from the model.

The remainder of this document is structured as follows: Section 2 provides the code used in data preparation. Section 3 provides the code and output of the cross validation of model parameters (subsection 3.1 for (penalized) logistic regressions; subsection 3.2 for random forests; subsection 3.3 for boosted tree ensembles; subsection 3.4 for prediction rule ensembles; subsection 3.5 for conditional inference trees; subsection 3.6 for k nearest neighbours). Section 4 provides version info about R and all packages used. Section 5 provides the references.

2 Data preparation

The data can be downloaded as a .csv file (contained in a .zip file) from https://openpsychometrics.org/_rawdata/, or more specifically: http://openpsychometrics.org/_rawdata/RIASEC_data12Dec2018.zip.

```r
data <- read.delim("data.csv", header = TRUE)
```

Items are scored 1-5, thus 0s are assumed to be missing values:

```r
# sapply(data[ , 1:48], table)
data[ , 1:48][sapply(data[ , 1:48], function(x) x == 0)] <- NA
data <- data[complete.cases(data[ , 1:48]), ]
# sapply(data[ , 1:48], table)
```

We select participants who completed a university degree only:

```r
data <- data[data$education >= 3, ]
```

The variable `major` contains the answer to the question: "If you attended a university, what was your major (e.g. psychology, English, civil engineering)?" We code it as a binary factor, indiciating whether the respondent did take psychology as a major, or not. The variable contains several typos, which we take into account when constructing the binary factor:

```r
# table(data$major)
psych_ids <- rowSums(sapply(c("psych", "psyhcology", "psycotherapy", "couns",
                              "behavior", "behaviour", "neuro"),
                            function(x) grepl(x, data$major, ignore.case = TRUE)))
anim_ids <- grepl("anim", data$major, ignore.case = TRUE) ## exclude animal psych
data$major <- factor(ifelse(psych_ids > 0, "psychology", "other"))
data$major[anim_ids > 0 & psych_ids > 0] <- "other"
```

We create identifiers to separate the dataset into 75% training and 25% test observations:

```r
set.seed(42)
test_ids <- sample(1:nrow(data), ceiling(nrow(data)/4))
train_ids <- which(!1:nrow(data) %in% test_ids)
train_y <- as.numeric(data$major)[train_ids] - 1
test_y <- as.numeric(data$major)[test_ids] - 1
```

Finally, we compute RIASEC scale scores by summing the item responses:

```r
data$Real <- rowSums(data[ , paste0("R", 1:8)])
data$Inve <- rowSums(data[ , paste0("I", 1:8)])
data$Arti <- rowSums(data[ , paste0("A", 1:8)])
data$Soci <- rowSums(data[ , paste0("S", 1:8)])
data$Ente <- rowSums(data[ , paste0("E", 1:8)])
data$Conv <- rowSums(data[ , paste0("C", 1:8)])
```

3 Cross validation of parameters

3.1 (Penalized) Logistic Regression

```
## Adapt plotting function:
library("glmnet")
plot.cv.glmnet <- function (x, sign.lambda = 1, cex = .7, main = "") {
  cvobj = x
  xlab = expression(Log(lambda))
  if (sign.lambda < 0)
    xlab = paste("-", xlab, sep = "")
  plot.args = list(x = sign.lambda * log(cvobj$lambda), y = cvobj$cvm,
                   ylim = range(cvobj$cvup, cvobj$cvlo), xlab = xlab, ylab = cvobj$name,
                   type = "n", cex = cex, cex.lab = cex, cex.main = cex, cex.axis = cex,
                   main = main)
  do.call("plot", plot.args)
  glmnet:::error.bars(sign.lambda * log(cvobj$lambda), cvobj$cvup, cvobj$cvlo,
                      width = 0.01, col = "darkgrey", cex = cex)
  points(sign.lambda * log(cvobj$lambda), cvobj$cvm, pch = 20,
         col = "red", cex = cex)
  axis(side = 3, at = sign.lambda * log(cvobj$lambda), labels = paste(cvobj$nz),
       tick = FALSE, line = 0, cex.axis = cex)
  abline(v = sign.lambda * log(cvobj$lambda.min), lty = 3)
  abline(v = sign.lambda * log(cvobj$lambda.1se), lty = 3)
  invisible()
}
```

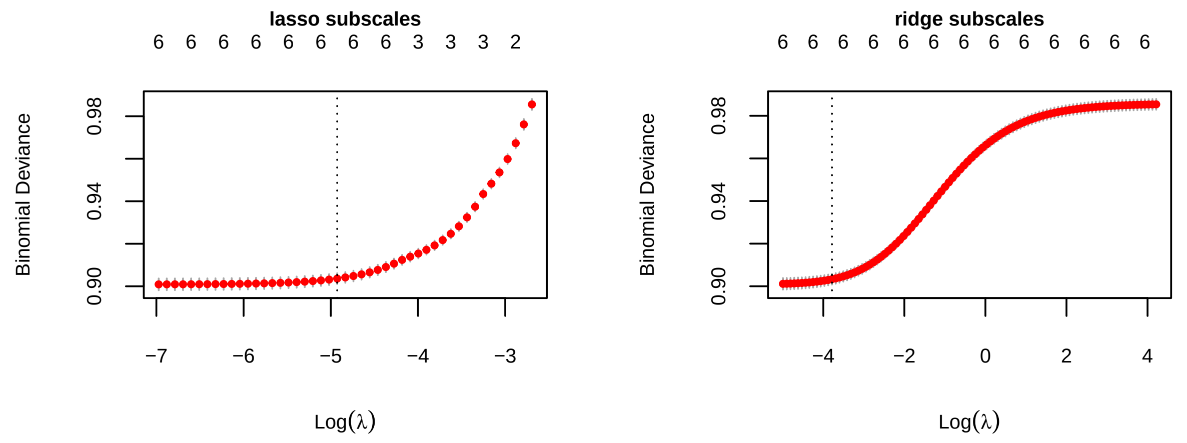
```
library("glmnet")
```

```
## Lasso scale scores
varnames <- c("Real", "Inve", "Arti", "Soci", "Ente", "Conv")
X <- as.matrix(data[train_ids, varnames])
set.seed(42)
l1 <- cv.glmnet(X, train_y, alpha = 1, family = "binomial")
lambda_l1 <- l1$lambda
## cv.glmnet() does not include lambda=0 by default, so need to include manually
set.seed(42)
l1 <- cv.glmnet(X, train_y, alpha = 1, family = "binomial",
                   lambda = c(lambda_l1, 0))

## Ridge scale scores
set.seed(42)
l2 <- cv.glmnet(X, train_y, alpha = 0, family = "binomial")
lambda_l2 <- l2$lambda
```

```r
set.seed(42)
l2 <- cv.glmnet(X, train_y, alpha = 0, lambda = c(lambda_l2, 0), family = "binomial")
```

```r
## Plot and print results
par(mfrow = c(1, 2))
plot(l1, cex = .7, main = "lasso subscales")
plot(l2, cex = .7, main = "ridge subscales")
```



```r
l1$lambda.min
```

```
## [1] 0
```

```r
l1$lambda.1se
```

```
## [1] 0.007250111
```

```r
l2$lambda.min
```

```
## [1] 0
```

```r
l2$lambda.1se
```

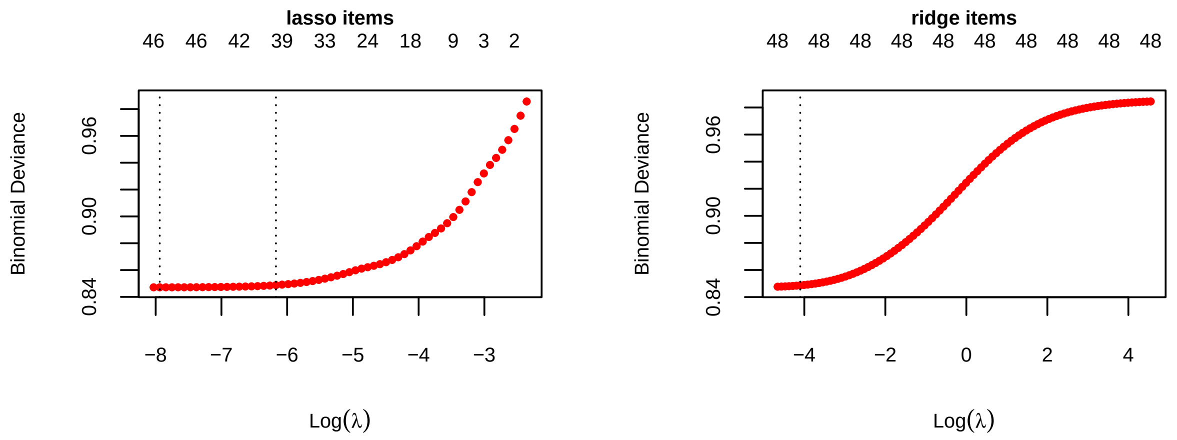
```
## [1] 0.02266179
```

```r
## Items
varnames <- paste0(rep(c("R", "I", "A", "S", "E", "C"), each = 8), 1:8)
X <- as.matrix(data[train_ids, varnames])

## Lasso
set.seed(42)
l1 <- cv.glmnet(X, train_y, alpha = 1, family = "binomial")
lambda_l1 <- l1$lambda
set.seed(42)
l1 <- cv.glmnet(X, train_y, alpha = 1, family = "binomial",
                lambda = c(lambda_l1, 0))

## Ridge
set.seed(42)
l2 <- cv.glmnet(X, train_y, alpha = 0, family = "binomial")
lambda_l2 <- l2$lambda
set.seed(42)
l2 <- cv.glmnet(X, train_y, alpha = 0, lambda = c(lambda_l2, 0), family = "binomial")
```

```
## Plot and print results
par(mfrow = c(1, 2))
plot(l1, cex = .7, main = "lasso items")
plot(l2, cex = .7, main = "ridge items")
```



```
l1$lambda.min
```

```
## [1] 0.0003568404
```

```
l1$lambda.1se
```

```
## [1] 0.002090022
```

```
l2$lambda.min
```

```
## [1] 0
```

```
l2$lambda.1se
```

```
## [1] 0.01656306
```

```
l1$cvm[which(l1$lambda.min == l1$lambda)]
```

```
## [1] 0.8471471
```

```
l2$cvm[which(l2$lambda.min == l2$lambda)]
```

```
## [1] 0.8472255
```

3.2 Random Forest

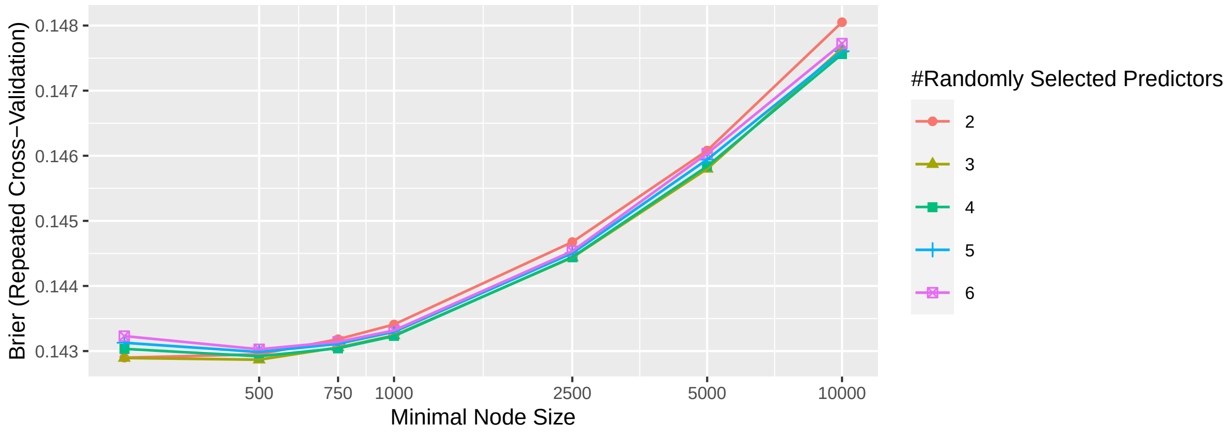
Parameters for gradient boosting, random forests and prediction rule ensembles were tuned using package `caret`.

```
## Load library, set up custom functions
library("caret")
library("ggplot2")
BigSummary <- function (data, lev = NULL, model = NULL) {
  brscore <- try(mean((data[, lev[2]] - ifelse(data$obs == lev[2], 1, 0)) ^ 2),
                 silent = TRUE)
  rocObject <- try(pROC::roc(ifelse(data$obs == lev[2], 1, 0), data[, lev[2]],
                             direction = "<", quiet = TRUE), silent = TRUE)
  if (inherits(brscore, "try-error")) brscore <- NA
  rocAUC <- if (inherits(rocObject, "try-error")) {
    NA
  } else {
    rocObject$auc
  }
  tmp <- unlist(e1071::classAgreement(table(data$obs,
                                           data$pred)))[c("diag", "kappa")]
  out <- c(Acc = tmp[[1]],
           Kappa = tmp[[2]],
           AUCROC = rocAUC,
           Brier = brscore)
  out
}
fitControl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 1,
                           ## Estimate class probabilities:
                           classProbs = TRUE,
                           ## Evaluate performance using
                           ## the following function:
                           summaryFunction = BigSummary,
                           verboseIter = TRUE)
```

```
## Subscales
rfGrid <-  expand.grid(mtry = c(2:6),
                       min.node.size = c(10000, 5000, 2500, 1000, 750, 500),
                       splitrule = "gini")
```

```r
set.seed(825)
rfFit <- train(major ~ Real + Inve + Arti + Soci + Ente + Conv,
               data = data[train_ids, ], method = "ranger", trControl = fitControl,
               tuneGrid = rfGrid, metric = "Brier", maximize = FALSE)
```

```r
## Print and plot results
ggplot(rfFit) + scale_x_continuous(trans="log") + theme_gray(base_size=9) +
  scale_x_continuous(trans = "log", breaks = c(10000, 5000, 2500, 1000, 750, 500))
```

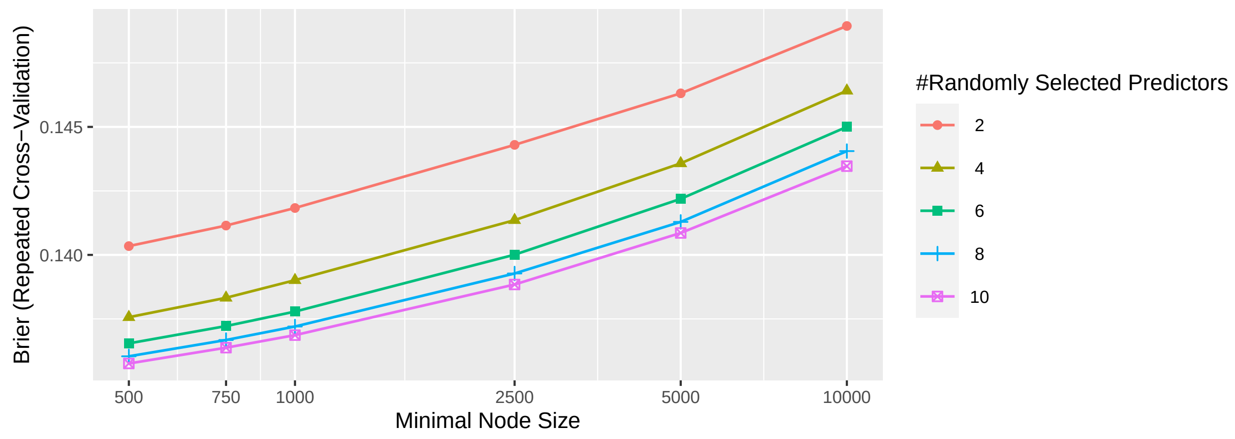


```r
rfFit$bestTune
```

```
##   mtry splitrule min.node.size
## 9    3      gini           500
```

```r
## Items
rfGrid_i <-  expand.grid(mtry = 2*(1:5),
                         min.node.size = c(10000, 5000, 2500, 1000, 750, 500),
                         splitrule = "gini")
x <- data[train_ids, paste0(rep(c("R", "I", "A", "S", "E", "C"), each = 8), 1:8)]
y <- data$major[train_ids]
set.seed(825)
rfFit_i <- train(x = x, y = y, method = "ranger", trControl = fitControl,
                 tuneGrid = rfGrid_i, metric = "Brier", maximize = FALSE)
```

```r
## Print and plot results
ggplot(rfFit_i) + scale_x_continuous(trans="log") + theme_gray(base_size=9) +
  scale_x_continuous(trans = "log", breaks = c(10000, 5000, 2500, 1000, 750, 500))
```



```
rfFit_i$bestTune
```

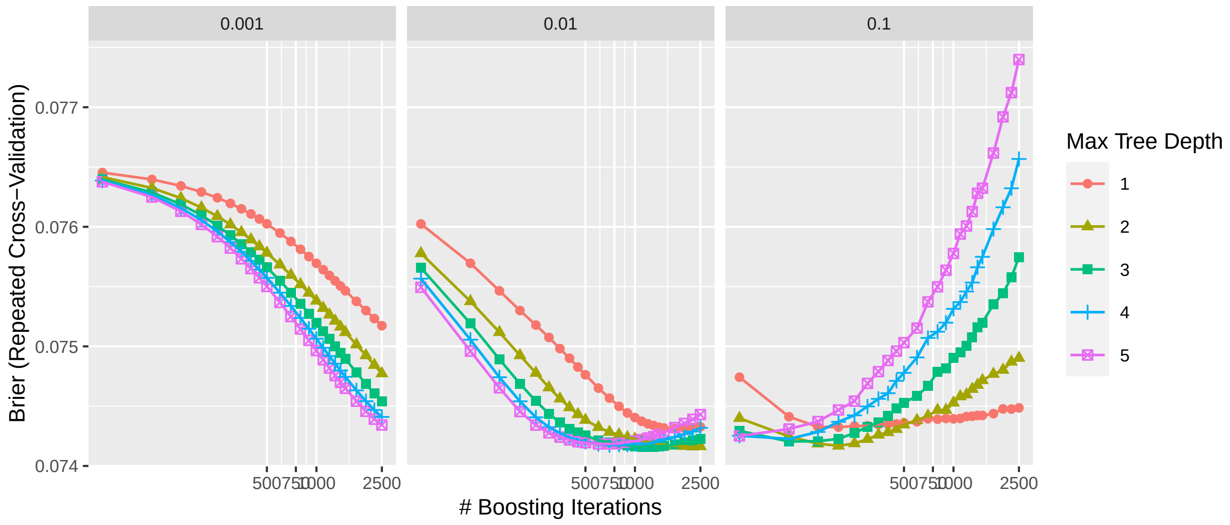
```
##    mtry splitrule min.node.size
## 25   10      gini           500
```

3.3 Gradient Tree Boosting

Gradient boosting is one of the top prediction approaches. Many forecasting competitions have been won by using gradient boosting. To obtain good performance with boosting, careful tuning of the model-fitting parameters is necessary. The most important parameters are tree size, the number of trees in the ensemble and the shrinkage or learning rate. Tree size (or interaction depth) determines the highest degree of interactions that can be captured by the tree in the ensembles. The learning rate (or shrinkage) parameter reflects the weight that is attributed to the predictions of each previous tree, when fitting the current tree.

```
## Subscales
gbmGrid <-  expand.grid(interaction.depth = 1:5,
                        n.trees = c((1:10)*50, 500+(1:10)*100, 1500+(1:4)*250),
                        shrinkage = c(0.001, 0.01, 0.1),
                        n.minobsinnode = 20)
set.seed(825)
gbmFit <- train(major ~ Real + Inve + Arti + Soci + Ente + Conv,
                data = data[train_ids, ],
                 method = "gbm",
                 trControl = fitControl,
                 tuneGrid = gbmGrid,
                 metric = "Brier",
                 maximize = FALSE)
```

```
## Print and plot results
ggplot(gbmFit, size = 2) + theme_gray(base_size=9) +
  scale_x_continuous(trans = "log", breaks = c(10000, 5000, 2500, 1000, 750, 500))
```

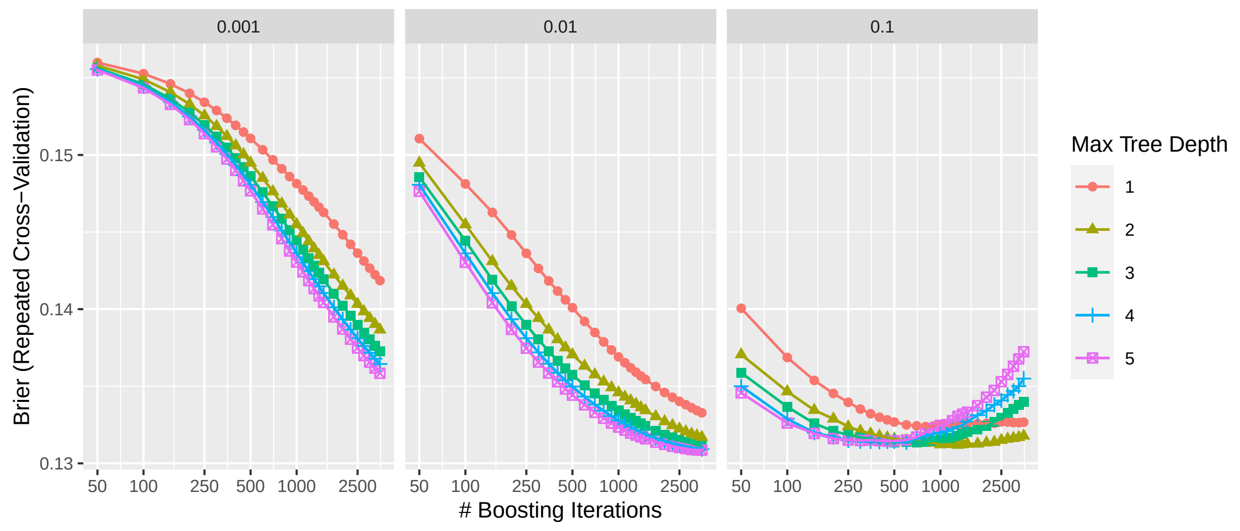


```
gbmFit$bestTune
```

```
##     n.trees interaction.depth shrinkage n.minobsinnode
## 184    1100                 3      0.01             20
```

```
## Items
x <- data[train_ids, paste0(rep(c("R", "I", "A", "S", "E", "C"), each = 8), 1:8)]
y <- data$major[train_ids]
gbmGrid_i <-  expand.grid(interaction.depth = 1:5,
                        n.trees = c((1:10)*50, 500+(1:10)*100, 1500+(1:8)*250),
                        shrinkage = c(0.001, 0.01, 0.1),
                        n.minobsinnode = 20)
set.seed(825)
gbmFit_i <- train(x = x, y = y,
                method = "gbm",
                trControl = fitControl,
                tuneGrid = gbmGrid_i,
                metric = "Brier",
                maximize = FALSE)
```

```
## Print and plot results
ggplot(gbmFit_i) + theme_gray(base_size=9) +
  scale_x_continuous(trans = "log", breaks = c(50, 100, 250, 500, 1000, 2500))
```



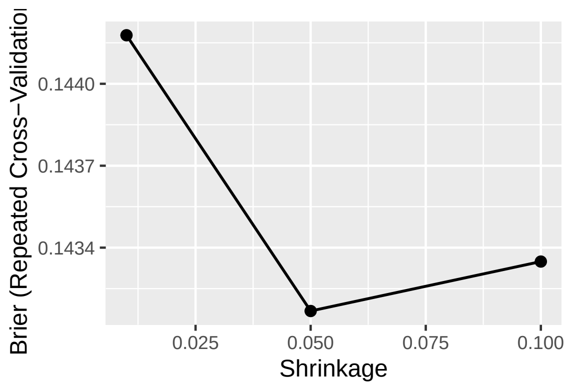
```
gbmFit_i$bestTune
```

```
##     n.trees interaction.depth shrinkage n.minobsinnode
## 280    3500                 5      0.01             20
```

3.4 Prediction Rule Ensembling

```
## Subscales
preGrid <- getModelInfo("pre")[[1]]$grid(
  learnrate = c(.01, .05, .1))
set.seed(825)
preFit <- train(major ~ Real + Inve + Arti + Soci + Ente + Conv,
                data = data[train_ids, ],
                method = "pre",
                trControl = fitControl,
                tuneGrid = preGrid,
                metric = "Brier",
                maximize = FALSE)
```

```
## Print and plot results
ggplot(preFit) + theme_gray(base_size=9)
```

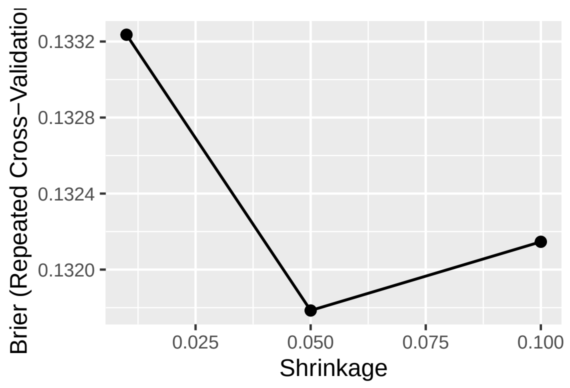


```r
preFit$bestTune
```

```
##   sampfrac maxdepth learnrate mtry use.grad penalty.par.val
## 2      0.5        3      0.05  Inf     TRUE      lambda.1se
```

```r
## items
varnames <- paste0(rep(c("R", "I", "A", "S", "E", "C"), each = 8), 1:8)
pr_form <- formula(paste("major ~", paste(varnames, collapse = "+")))
set.seed(825)
preFit_i <- train(pr_form,
                  data = data[train_ids, ],
                  method = "pre",
                  trControl = fitControl,
                  tuneGrid = preGrid,
                  metric = "Brier",
                  maximize = FALSE)
```

```r
## Print and plot results
ggplot(preFit_i) + theme_gray(base_size=9)
```



```r
preFit_i$bestTune
```

```
##   sampfrac maxdepth learnrate mtry use.grad penalty.par.val
## 2      0.5        3      0.05  Inf     TRUE      lambda.1se
```

3.5 Conditional Inference Tree

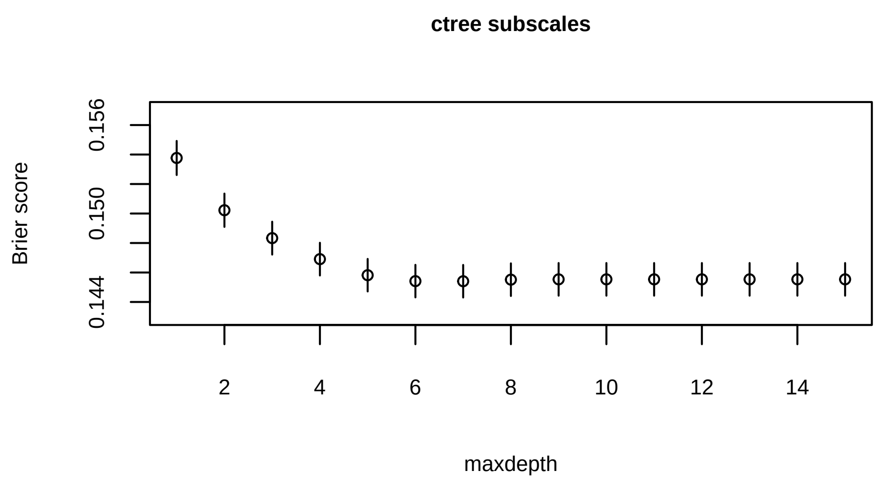
For conditional inference trees and k nearest neighbours, we wrote custom code for tuning the parameters:

```r
library("partykit")
dat <- data[train_ids, ]

## Subscales
varnames <- c("Real", "Inve", "Arti", "Soci", "Ente", "Conv")
ct_form <- formula(paste("major ~", paste(varnames, collapse = "+")))
set.seed(42)
fold_ids <- sample(1:10, size = nrow(dat), replace = TRUE)
ct_preds <- data.frame(matrix(rep(NA, times = nrow(dat)*15), nrow = nrow(dat)))
names(ct_preds) <- paste0("m", 1:15)

set.seed(43)
for (i in 1:10) {
  cat("Fold", i, ". ")
  for (j in 1:15) {
    ct <- ctree(ct_form, data = dat[fold_ids != i, ], maxdepth = j)
    ct_preds[fold_ids == i, paste0("m", j)] <- predict(
      ct, type = "prob", newdata = dat[fold_ids == i, ])[ , "psychology"]
  }
}
```

```r
## Print and plot results
br_ct <- sapply(ct_preds, function(x) mean((x - train_y)^2))
br_ct_se <- sapply(ct_preds, function(x) sd((x - train_y)^2)/sqrt(length(train_ids)))
plot(br_ct, xlab = "maxdepth", ylab = "Brier score", main = "ctree subscales",
     ylim = c(0.143, 0.157), cex = .7, cex.axis = .7, cex.main = .7, cex.lab = .7)
arrows(x0 = 1:15, y0 = br_ct - br_ct_se, y1 = br_ct + br_ct_se, length = 0)
```



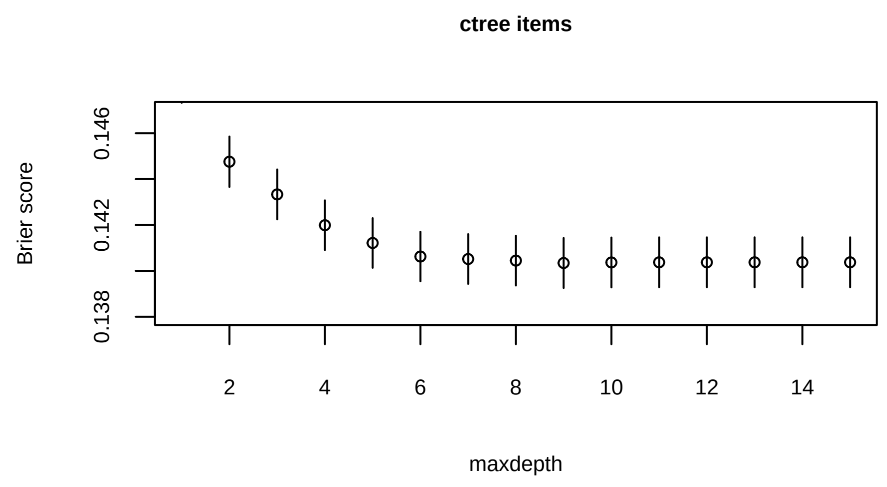
```
which(br_ct == min(br_ct))
```

```
## m7
##  7
```

```
## Items
varnames <- paste0(rep(c("R", "I", "A", "S", "E", "C"), each = 8), 1:8)
ct_form <- formula(paste("major ~", paste(varnames, collapse = "+")))
set.seed(42)
fold_ids <- sample(1:10, size = nrow(dat), replace = TRUE)
ct_preds <- data.frame(matrix(rep(NA, times = nrow(dat)*15), nrow = nrow(dat)))
names(ct_preds) <- paste0("m", 1:15)

set.seed(43)
for (i in 1:10) {
  cat("Fold", i, ". ")
  for (j in 1:15) {
    ct <- ctree(ct_form, data = dat[fold_ids != i, ], maxdepth = j)
    ct_preds[fold_ids == i, paste0("m", j)] <- predict(
      ct, type = "prob", newdata = dat[fold_ids == i, ])[ , "psychology"]
  }
}
```

```
## Print and plot results
br_ct <- sapply(ct_preds, function(x) mean((x - train_y)^2))
br_ct_se <- sapply(ct_preds, function(x) sd((x - train_y)^2)/sqrt(length(train_ids)))
plot(br_ct, xlab = "maxdepth", ylab = "Brier score", main = "ctree items",
     ylim = c(0.138, 0.147), cex = .7, cex.axis = .7, cex.main = .7, cex.lab = .7)
arrows(x0 = 1:15, y0 = br_ct - br_ct_se, y1 = br_ct + br_ct_se, length = 0)
```



```
which(br_ct == min(br_ct))
```

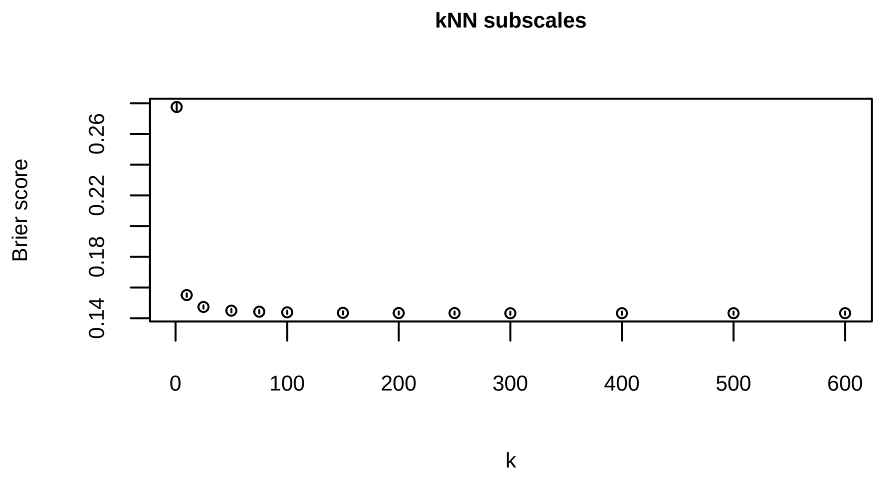
```
## m9
##  9
```

3.6 k Nearest Neighbours

```
library("class")

## Subscales
varnames <- c("Real", "Inve", "Arti", "Soci", "Ente", "Conv")
set.seed(42)
fold_ids <- sample(1:10, size = nrow(dat), replace = TRUE)
names(ct_preds) <- paste0("m", 1:15)
k <- c(1L, 10L, 25L, 50L, 75L, 100L, 150L, 200L, 250L, 300L, 400L, 500L, 600L)
knn_preds <- data.frame(matrix(rep(NA, times = nrow(dat)*length(k)),
                                   nrow = nrow(dat)))
names(knn_preds) <- as.character(k)
set.seed(43)
for (i in 1:10) {
  cat("Fold", i, ". ")
  for (j in k) {
    try(
      knn_mod <- knn(dat[fold_ids != i, varnames], dat[fold_ids == i, varnames],
          cl = as.factor(dat[fold_ids != i, "major"]),
          k = j, use.all = TRUE, prob = TRUE)
    )
    ## Need to obtain predicted probability for second class
    knn_preds[fold_ids == i, as.character(j)] <- ifelse(
      knn_mod == "psychology", attr(knn_mod, "prob"), 1 - attr(knn_mod, "prob"))
  }
}
```

```
## Print and plot results
br_knn <- sapply(knn_preds, function(x) mean((x - train_y)^2))
br_knn_se <- sapply(knn_preds, function(x) sd((x - train_y)^2)/sqrt(length(train_ids)))
plot(k, br_knn, main = "kNN subscales", ylab = "Brier score",
     cex = .7, cex.axis = .7, cex.main = .7, cex.lab = .7)
arrows(x0 = k, y0 = br_knn - br_knn_se, y1 = br_knn + br_knn_se, length = 0)
```



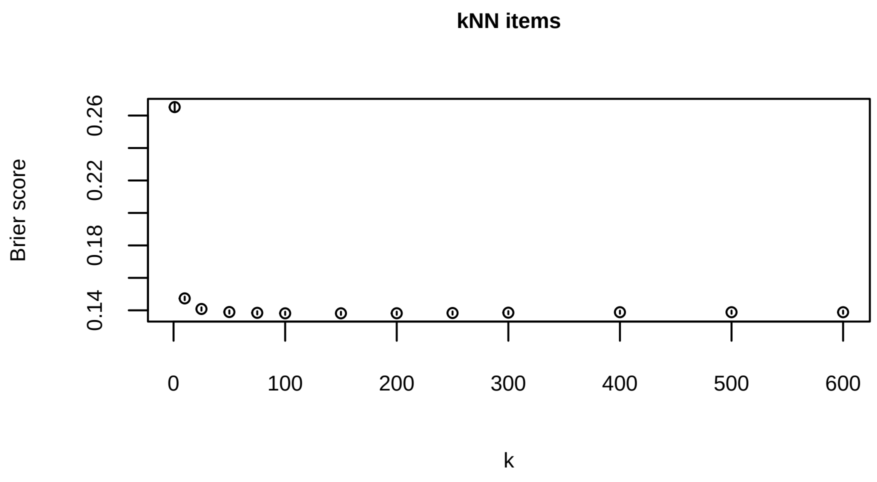
```
which(br_knn == min(br_knn))
```

```
## 300
##  10
```

```
## Items
varnames <- paste0(rep(c("R", "I", "A", "S", "E", "C"), each = 8), 1:8)
set.seed(42)
fold_ids <- sample(1:10, size = nrow(dat), replace = TRUE)
k <- c(1L, 10L, 25L, 50L, 75L, 100L, 150L, 200L, 250L, 300L, 400L, 500L, 600L)
knn_preds <- data.frame(matrix(rep(NA, times = nrow(dat)*length(k)),
                               nrow = nrow(dat)))
names(knn_preds) <- as.character(k)

set.seed(43)
for (i in 1:10) {
  cat("Fold", i, ". ")
  for (j in k) {
    try(
      knn_mod <- knn(dat[fold_ids != i, varnames], dat[fold_ids == i, varnames],
                     cl = as.factor(dat[fold_ids != i, "major"]),
                     k = j, use.all = TRUE, prob = TRUE)
    )
    ## Need to obtain predicted probability for second class
    knn_preds[fold_ids == i, as.character(j)] <- ifelse(
      knn_mod == "psychology", attr(knn_mod, "prob"), 1 - attr(knn_mod, "prob"))
  }
}
```

```
## Print and plot results
br_knn <- sapply(knn_preds, function(x) mean((x - train_y)^2))
br_knn_se <- sapply(knn_preds, function(x) sd((x - train_y)^2)/sqrt(length(train_ids)))
plot(k, br_knn, main = "kNN items", ylab = "Brier score",
     cex = .7, cex.axis = .7, cex.main = .7, cex.lab = .7)
arrows(x0 = k, y0 = br_knn - br_knn_se, y1 = br_knn + br_knn_se, length = 0)
```



```
which(br_knn == min(br_knn))
```

```
## 100
##   6
```

4 R version and package info

```
sessionInfo()
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19042)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=Dutch_Netherlands.1252  LC_CTYPE=Dutch_Netherlands.1252
## [3] LC_MONETARY=Dutch_Netherlands.1252 LC_NUMERIC=C
## [5] LC_TIME=Dutch_Netherlands.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] caret_6.0-89    lattice_0.20-44 ggplot2_3.3.5   glmnet_4.1-2
## [5] Matrix_1.3-4
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.7           lubridate_1.7.10     listenv_0.8.0
##  [4] class_7.3-19         assertthat_0.2.1     digest_0.6.27
##  [7] ipred_0.9-12         foreach_1.5.1        utf8_1.2.1
## [10] parallelly_1.28.1    R6_2.5.0             plyr_1.8.6
## [13] stats4_4.1.0         evaluate_0.14        highr_0.9
## [16] pillar_1.6.1         rlang_0.4.11         data.table_1.14.0
## [19] rpart_4.1-15         rmarkdown_2.11       labeling_0.4.2
## [22] splines_4.1.0        gower_0.2.2          stringr_1.4.0
## [25] munsell_0.5.0        compiler_4.1.0       xfun_0.29
## [28] pkgconfig_2.0.3      shape_1.4.6          globals_0.14.0
## [31] htmltools_0.5.1.1    nnet_7.3-16          tidyselect_1.1.1
## [34] tibble_3.1.2         prodlim_2019.11.13   codetools_0.2-18
## [37] fansi_0.5.0          future_1.22.1        crayon_1.4.1
## [40] dplyr_1.0.7          withr_2.4.2          MASS_7.3-54
## [43] recipes_0.1.17       ModelMetrics_1.2.2.2 grid_4.1.0
## [46] nlme_3.1-152         gtable_0.3.0         lifecycle_1.0.0
## [49] DBI_1.1.1            magrittr_2.0.1       pROC_1.18.0
## [52] scales_1.1.1         future.apply_1.8.1   stringi_1.6.2
## [55] farver_2.1.0         reshape2_1.4.4       timeDate_3043.102
## [58] ellipsis_0.3.2       generics_0.1.0       vctrs_0.3.8
## [61] lava_1.6.10          iterators_1.0.13     tools_4.1.0
## [64] glue_1.4.2          purrr_0.3.4          parallel_4.1.0
## [67] survival_3.2-11      yaml_2.2.1           colorspace_2.0-2
## [70] knitr_1.37
```

References

Fokkema, M. (2020). Fitting prediction rule ensembles with R package pre. *Journal of Statistical Software*, *92*(1), 1–30.

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*(1), 1.

Greenwell, B., Boehmke, B., Cunningham, J., & Developers, G. (2020). *Gbm: Generalized boosted regression models*. https://CRAN.R-project.org/package=gbm

Hothorn, T., Hornik, K., & Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, *15*(3), 651–674. https://doi.org/10.1198/106186006X133933

Hothorn, T., & Zeileis, A. (2015). partykit: A modular toolkit for recursive partytioning in R. *Journal of Machine Learning Research*, *16*, 3905–3909. https://jmlr.org/papers/v16/hothorn15a.html

Kuhn, M. (2021). *Caret: Classification and regression training*. https://CRAN.R-project.org/package=caret

R Core Team. (2021). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with S* (4th Edition). Springer. https://www.stats.ox.ac.uk/pub/MASS4/

Wood, S. N. (2017). *Generalized additive models: An introduction with R*. CRC press.

Wright, M. N., & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, *77*(1), 1–17. https://doi.org/10.18637/jss.v077.i01