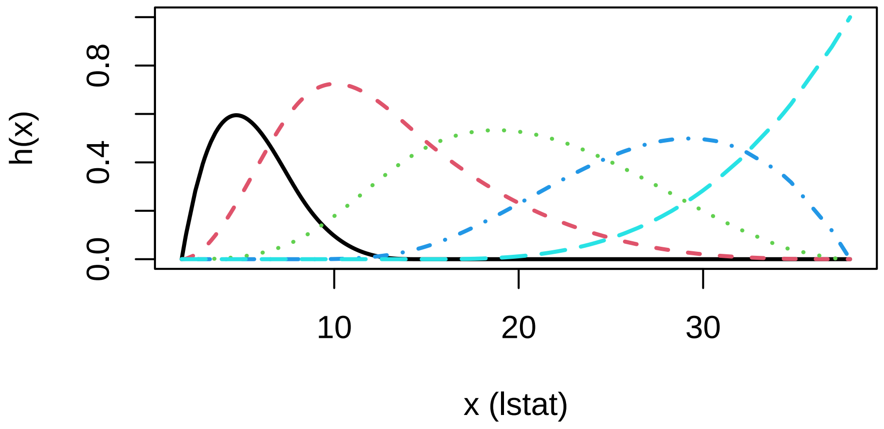
Answers to exercises Session 5

Exercise 1: Cubic and natural splines

a, b, c)

```r
library("MASS")
data(Boston)
library("splines")
Boston <- Boston[order(Boston$lstat), ]
bs_x <- bs(Boston$lstat, df = 5)
matplot(Boston$lstat[order(Boston$lstat)], bs_x[order(Boston$lstat), ],
        type = "l", lwd = 2, xlab = "x (lstat)", ylab = "h(x)")
```



```r
head(bs_x)
```

```
##              1           2            3 4 5
## [1,] 0.0000000 0.000000000 0.000000e+00 0 0
## [2,] 0.0828165 0.001247442 2.216045e-06 0 0
## [3,] 0.1074457 0.002148435 5.048215e-06 0 0
## [4,] 0.2824476 0.018017765 1.309221e-04 0 0
## [5,] 0.3928704 0.041199495 4.786657e-04 0 0
## [6,] 0.3952802 0.041885740 4.913730e-04 0 0
```

```r
attr(bs_x, "knots")
```

```
## 33.33333% 66.66667%
##  8.316667 14.696667
```

```r
attr(bs_x, "degree")
```

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

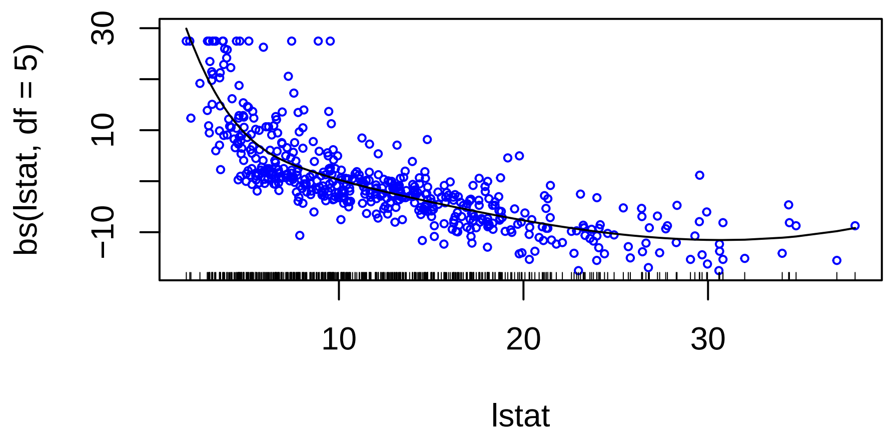
With `df = 5`, we obtain a design matrix of 5 columns. With a cubic spline, we use up 3 df for the first three expansions. Thus, with 5 df, we have 2 df ‘left’ to spend on the knots. Each knot introduces one additional basis function. Thus, with 5 df for a cubic spline, we can use 2 knots. Note that the knots are placed based on the univariate distribution of the predicton.

d)

```r
library("gam")
mod_df5 <- gam(medv ~ bs(lstat, df = 5), data = Boston)
summary(mod_df5)
```

```
##
## Call: gam(formula = medv ~ bs(lstat, df = 5), data = Boston)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -15.1774  -3.1790  -0.7981   2.0964  26.6755
##
## (Dispersion Parameter for gaussian family taken to be 27.109)
##
##     Null Deviance: 42716.3 on 505 degrees of freedom
## Residual Deviance: 13554.52 on 500 degrees of freedom
## AIC: 3113.663
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                    Df Sum Sq Mean Sq F value    Pr(>F)
## bs(lstat, df = 5)   5  29162  5832.4  215.14 < 2.2e-16 ***
## Residuals         500  13554    27.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

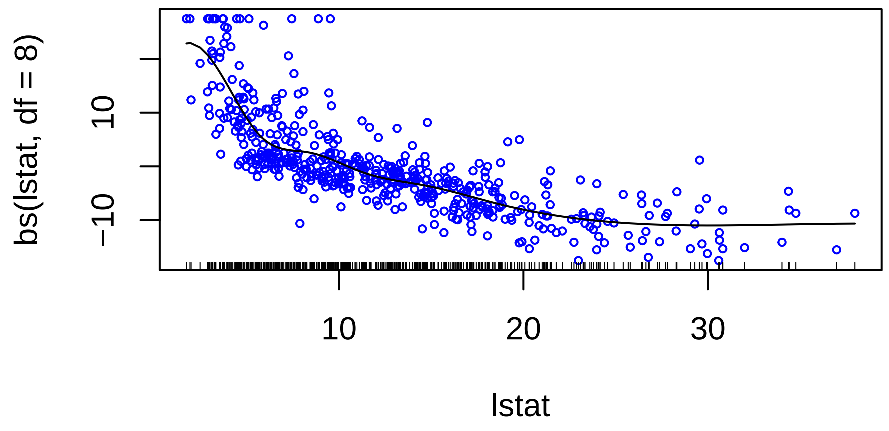
```r
plot(mod_df5, residuals = TRUE, col = "blue", cex = .5)
```



```r
mod_df8 <- gam(medv ~ bs(lstat, df = 8), data = Boston)
summary(mod_df8)
```

```
##
## Call: gam(formula = medv ~ bs(lstat, df = 8), data = Boston)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -14.9627  -3.1253  -0.6612   2.0831  26.0972
##
## (Dispersion Parameter for gaussian family taken to be 26.7118)
##
##     Null Deviance: 42716.3 on 505 degrees of freedom
## Residual Deviance: 13275.77 on 497 degrees of freedom
## AIC: 3109.148
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                    Df Sum Sq Mean Sq F value    Pr(>F)
## bs(lstat, df = 8)   8  29441  3680.1  137.77 < 2.2e-16 ***
## Residuals         497  13276    26.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
plot(mod_df8, residuals = TRUE, col = "blue", cex = .5)
```



```
BIC(mod_df5)
```

```
## [1] 3143.249
```

```
BIC(mod_df8)
```

```
## [1] 3151.414
```

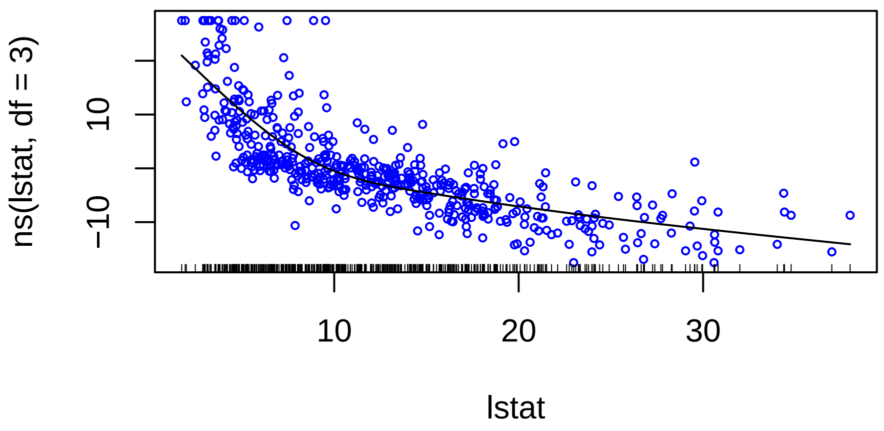
The 5 df cubic spline fits best according to BIC, the plots suggest similar: 8 df yields a slightly too wiggly function. Note that models with different (number of location of) knots are not nested, so we cannot use stattistical testing to compare the model fit.

e)

```
mod_ns3 <- gam(medv ~ ns(lstat, df = 3), data = Boston)
summary(mod_ns3)
```

```
##
## Call: gam(formula = medv ~ ns(lstat, df = 3), data = Boston)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -13.7595  -3.3628  -0.6468   2.3062  27.2857
##
## (Dispersion Parameter for gaussian family taken to be 28.4261)
##
##     Null Deviance: 42716.3 on 505 degrees of freedom
## Residual Deviance: 14269.9 on 502 degrees of freedom
## AIC: 3135.688
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                    Df Sum Sq Mean Sq F value    Pr(>F)
## ns(lstat, df = 3)   3  28446  9482.1  333.57 < 2.2e-16 ***
## Residuals         502  14270    28.4
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

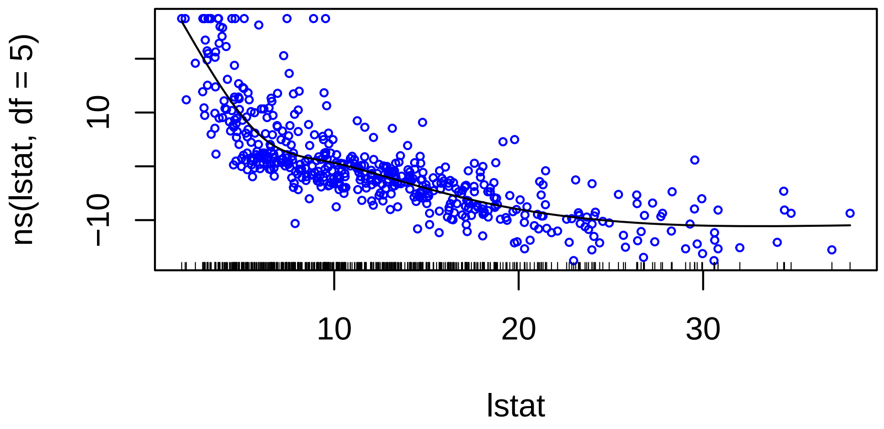
```r
plot(mod_ns3, residuals = TRUE, col = "blue", cex = .5)
```



```r
mod_ns5 <- gam(medv ~ ns(lstat, df = 5), data = Boston)
summary(mod_ns5)
```

```
##
## Call: gam(formula = medv ~ ns(lstat, df = 5), data = Boston)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -13.9811  -3.0266  -0.7252   2.1416  26.5111
##
## (Dispersion Parameter for gaussian family taken to be 26.9021)
##
##     Null Deviance: 42716.3 on 505 degrees of freedom
## Residual Deviance: 13451.03 on 500 degrees of freedom
## AIC: 3109.785
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                    Df Sum Sq Mean Sq F value    Pr(>F)
## ns(lstat, df = 5)   5  29265  5853.1  217.57 < 2.2e-16 ***
## Residuals         500  13451    26.9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
plot(mod_ns5, residuals = TRUE, col = "blue", cex = .5)
```



```
BIC(mod_ns3)
```

```
## [1] 3156.82
```

```
BIC(mod_ns5)
```

```
## [1] 3139.37
```

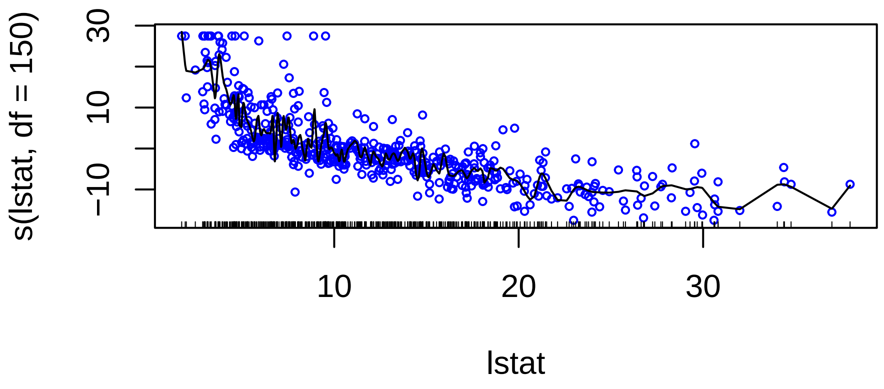
The lowest BIC value was obtained for the natural spline with 5 df. Visually, both the 3 and 5 df natural splines seem to provide a good fit to the data.

Exercise 2: Smoothing spline

```r
mod_sc <- gam(medv ~ s(lstat, df = 150), data = Boston) # complex fit
summary(mod_sc)
```

```
##
## Call: gam(formula = medv ~ s(lstat, df = 150), data = Boston)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -14.0712  -2.7340  -0.4702   2.0649  21.9654
##
## (Dispersion Parameter for gaussian family taken to be 27.6915)
##
##     Null Deviance: 42716.3 on 505 degrees of freedom
## Residual Deviance: 10772.01 on 389 degrees of freedom
## AIC: 3219.4
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                     Df Sum Sq Mean Sq F value    Pr(>F)
## s(lstat, df = 150)   1  23244 23243.9  839.39 < 2.2e-16 ***
## Residuals          389  10772    27.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                    Npar Df Npar F     Pr(F)
## (Intercept)
## s(lstat, df = 150)     115 2.7321 2.103e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
plot(mod_sc, residuals = TRUE, cex = .5, col = "blue")
```

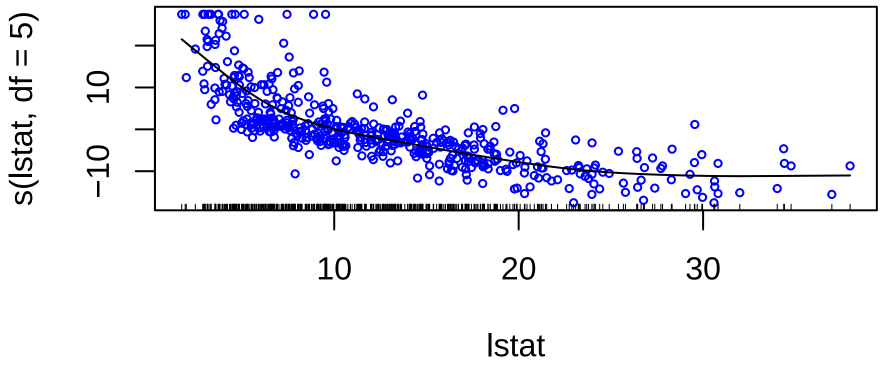


The results present both a parametric and non-parametric effect of `lstat`. The parametric effect represent the linear slope, thus using only 1 df. The non-parametric effects represent the non-linear effects. The non-linear part of the smoothing spline for `lstat` took up 115 degrees of freedom. Note that this is less than the requested degrees of freedom, because by default the knots are placed at a subset of the observations, for computational considerations. In addition, >115 knots would be rarely needed to approximate a curve.

```
mod_ss <- gam(medv ~ s(lstat, df = 5), data = Boston) # more simple fit
summary(mod_ss)
```

```
##
## Call: gam(formula = medv ~ s(lstat, df = 5), data = Boston)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -13.6332  -3.2159  -0.6577   2.2051  26.8386
##
## (Dispersion Parameter for gaussian family taken to be 27.6492)
##
##     Null Deviance: 42716.3 on 505 degrees of freedom
## Residual Deviance: 13824.6 on 499.9999 degrees of freedom
## AIC: 3123.646
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                   Df Sum Sq Mean Sq F value    Pr(>F)
## s(lstat, df = 5)   1  23244 23243.9  840.67 < 2.2e-16 ***
## Residuals        500  13825    27.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                  Npar Df Npar F     Pr(F)
## (Intercept)
## s(lstat, df = 5)       4 51.065 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
plot(mod_ss, residuals = TRUE, col = "blue", cex = .5)
```



With 5 df, the flexibility is much lower, and we obtain a much smoother fit.

```
BIC(mod_ss)
```

```
## [1] 3136.326
```

```
BIC(mod_sc)
```

```
## [1] 3232.079
```

```
BIC(mod_ns5)
```

```
## [1] 3139.37
```

According to the BIC, the more heavily penalized smoothing spline (i.e., with df = 5) has better fit than the less reularized smoothing spline. This is in accordance with what we can conclude from the visual inspection of the fitted smoothing splines. The smoothing spline with 5 df also outperforms the natural spline with 5 df from the previous exercise. Thus, the non-parametric smoothing spline approach appears to improve on the parametric natural and cubic spline approaches.

Exercise 3: Multiple predictors, binary outcome

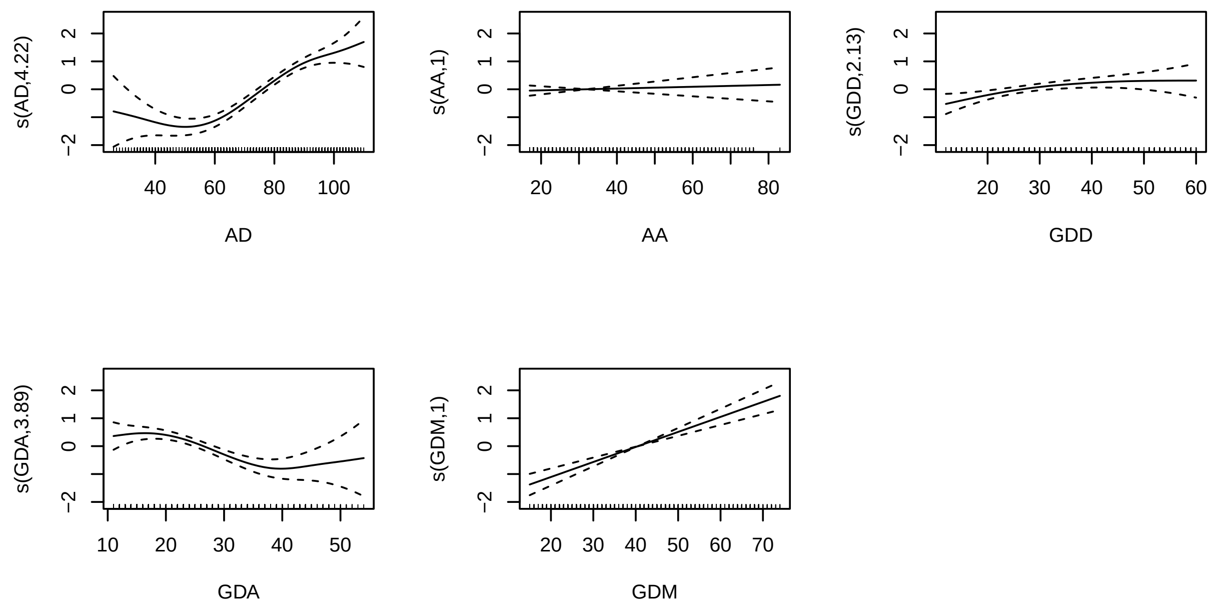
```
detach("package:gam", unload=TRUE)
library("mgcv")
MASQ <- read.table("MASQ.txt")
set.seed(1)
train <- sample(1:nrow(MASQ), size = nrow(MASQ)*.8)
summary(MASQ)
```

```
##     D_DEPDYS            AD               AA             GDD
##  Min.   :0.0000   Min.   : 26.00   Min.   :17.00   Min.   :12.00
##  1st Qu.:0.0000   1st Qu.: 64.00   1st Qu.:22.00   1st Qu.:20.00
##  Median :0.0000   Median : 77.00   Median :28.00   Median :29.00
##  Mean   :0.4643   Mean   : 75.05   Mean   :32.01   Mean   :30.64
##  3rd Qu.:1.0000   3rd Qu.: 88.00   3rd Qu.:39.00   3rd Qu.:40.00
##  Max.   :1.0000   Max.   :110.00   Max.   :83.00   Max.   :60.00
##       GDA            GDM          leeftijd      geslacht
##  Min.   :11.0   Min.   :15.0   Min.   :17.0   Length:3597
##  1st Qu.:19.0   1st Qu.:31.0   1st Qu.:28.0   Class :character
##  Median :24.0   Median :40.0   Median :38.0   Mode  :character
##  Mean   :25.4   Mean   :40.6   Mean   :38.8
##  3rd Qu.:31.0   3rd Qu.:50.0   3rd Qu.:48.0
##  Max.   :54.0   Max.   :75.0   Max.   :91.0
##      D_TOT
##  Min.   :0.000
##  1st Qu.:1.000
##  Median :2.000
##  Mean   :2.127
##  3rd Qu.:4.000
##  Max.   :7.000
```

```
GAM <- gam(D_DEPDYS ~ s(AD) + s(AA) + s(GDD) + s(GDA) + s(GDM),
           data = MASQ[train, ], method = "REML", family = "binomial")
summary(GAM)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## D_DEPDYS ~ s(AD) + s(AA) + s(GDD) + s(GDA) + s(GDM)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.25204    0.04854  -5.192 2.08e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df  Chi.sq  p-value
## s(AD)  4.221  5.213 156.700  < 2e-16 ***
## s(AA)  1.001  1.002   0.272   0.6026
## s(GDD) 2.132  2.725   9.315   0.0216 *
## s(GDA) 3.891  4.847  34.126 3.71e-06 ***
## s(GDM) 1.001  1.001  51.882  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.325   Deviance explained = 26.4%
## -REML = 1482.6  Scale est. = 1         n = 2877
```

```
par(mfrow = c(2, 3))
plot(GAM)
```



We compute the mean squared error and misclassification rate using predicted probabilities, for both training and test observations:

```
## Training data
GAM_preds_train <- predict(GAM, newdata = MASQ[train, ], type = "response")
mean((MASQ[train, "D_DEPDYS"] - GAM_preds_train)^2) ## Brier score
```

```
## [1] 0.1669075
```

```
tab_train <- prop.table(table(MASQ[train, "D_DEPDYS"], GAM_preds_train > .5)) ## confusion matrix
1 - sum(diag(tab_train)) ## MCR
```

```
## [1] 0.2440042
```

```
## Test data
GAM_preds_test <- predict(GAM, newdata = MASQ[-train, ], type = "response")
mean((MASQ[-train, "D_DEPDYS"] - GAM_preds_test)^2) ## Brier score
```

```
## [1] 0.1690516
```

```
tab_test <- prop.table(table(MASQ[-train, "D_DEPDYS"], GAM_preds_test > .5)) ## confusion matrix
1 - sum(diag(tab_test)) ## MCR
```

```
## [1] 0.2444444
```

The Brier score and confusion matrices are quite similar between training and test data, indicating little overfitting.

Exercise 4: Fit an SVM

```
library("e1071")
```

```
## Warning: package 'e1071' was built under R version 4.1.1
```

```
cost <- c(.001, .01, .1, 1, 5, 10, 100)
set.seed(42)
names(MASQ)
```

```
## [1] "D_DEPDYS" "AD"       "AA"       "GDD"      "GDA"      "GDM"      "leeftijd"
## [8] "geslacht" "D_TOT"
```

```
MASQ <- MASQ[ , -9]
names(MASQ)
```

```
## [1] "D_DEPDYS" "AD"       "AA"       "GDD"      "GDA"      "GDM"      "leeftijd"
## [8] "geslacht"
```

```
MASQ$D_DEPDYS <- factor(MASQ$D_DEPDYS)
```

```
tune.out <- tune(svm, D_DEPDYS ~ ., data = MASQ[train, ], kernel = "linear",
                 ranges = list(cost = cost))
```

```
tune.out$best.parameters
```

```
##   cost
## 3  0.1
```

```
svmfit <- svm(D_DEPDYS ~ ., data = MASQ[train,], kernel = "linear",
              cost = 0.1)
```

```
tab_train <- table(MASQ[train, "D_DEPDYS"],
                   predict(svmfit, newdata = MASQ[train, ]))
1 - sum(diag(prop.table(tab_train))) ## misclassification rate
```

```
## [1] 0.2415711
```

```
tab_test <- table(MASQ[-train, "D_DEPDYS"],
                  predict(svmfit, newdata = MASQ[-train, ]))
1 - sum(diag(prop.table(tab_test))) ## misclassification rate
```

```
## [1] 0.2486111
```

Radial basis kernel

Perhaps we can further improve predictions by using a non-linear kernel. We try the radial basis kernel:

```
gamma <- c(0.5, 1, 2, 3, 4)
tune.out <- tune(svm, D_DEPDYS ~ ., data = MASQ[train, ],
                 kernel = "radial", ranges = list(
                   cost = cost, gamma = gamma))
```

```
tune.out$best.parameters
```

```
##   cost gamma
## 4    1   0.5
```

```
rbkfit <- svm(D_DEPDYS ~ ., data = MASQ[train, ],
              kernel = "radial", gamma = 0.5,
              cost = 1)
```

```
tab_train <- table(MASQ[train, "D_DEPDYS"],
                   predict(rbkfit, newdata = MASQ[train, ]))
1 - sum(diag(prop.table(tab_train))) ## misclassification rate
```

```
## [1] 0.2102885
```

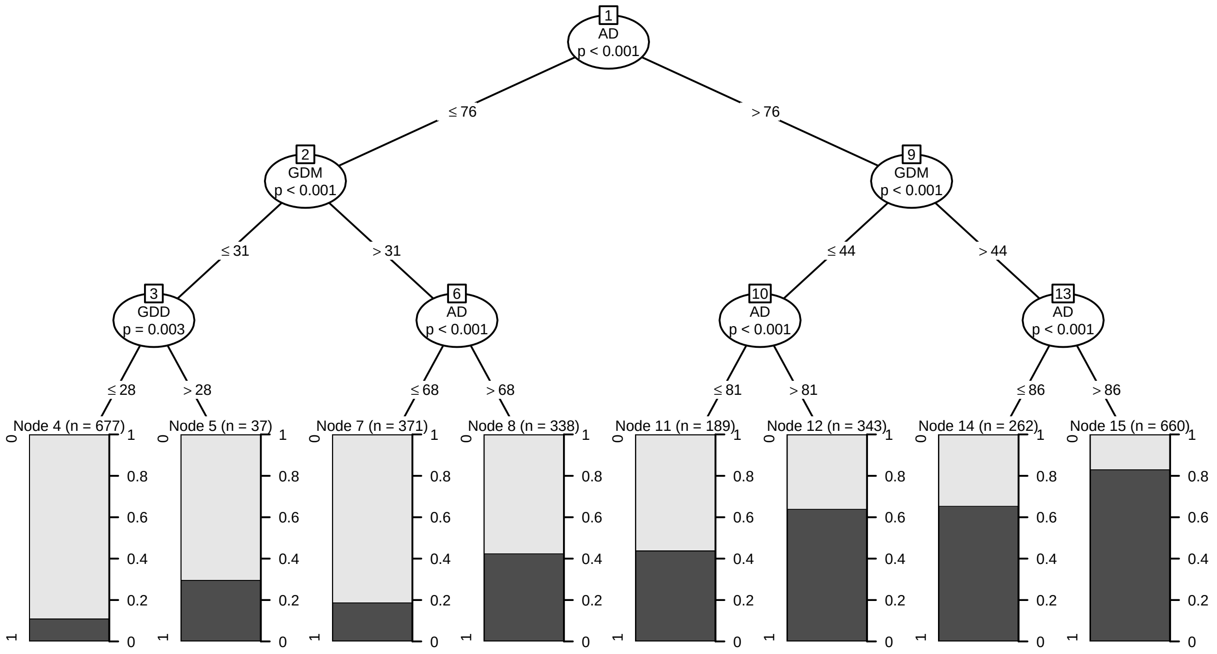
```
tab_test <- table(MASQ[-train, "D_DEPDYS"],
                   predict(rbkfit, newdata = MASQ[-train, ]))
1 - sum(diag(prop.table(tab_test))) ## misclassification rate
```

```
## [1] 0.2486111
```

The radial basis kernel did not improve performance on test data.

Exercise 5: Fit a ctree to MASQ data

```
library("partykit")
MASQ$geslacht <- factor(MASQ$geslacht)
ct <- ctree(D_DEPDYS ~ . , data = MASQ[train, ])
plot(ct, gp = gpar(cex = .5))
```



The conditional inference tree indicates a positive effect of the AD, GDM and GDD subscales on the probability of having a depressive / dysthymic disorder.

```
y_train <- as.numeric(MASQ[train, "D_DEPDYS"]) - 1
## Training data
ct_preds_train <- predict(ct, newdata = MASQ[train, ], type = "prob")[ , 2]
mean((y_train - ct_preds_train)^2) ## Brier score
```

```
## [1] 0.1705674
```

```
tab_train <- prop.table(table(MASQ[train, "D_DEPDYS"], ct_preds_train > .5)) ## confusion matrix
1 - sum(diag(tab_train)) ## MCR
```

```
## [1] 0.2457421
```

```
## Test data
y_test <- as.numeric(MASQ[-train, "D_DEPDYS"]) - 1
ct_preds_test <- predict(ct, newdata = MASQ[-train, ], type = "prob")[ , 2]
mean((y_test - ct_preds_test)^2) ## Brier score
```

```
## [1] 0.1738697
```

```
tab_test <- prop.table(table(MASQ[-train, "D_DEPDYS"], ct_preds_test > .5)) ## confusion matrix
1 - sum(diag(tab_test)) ## MCR
```

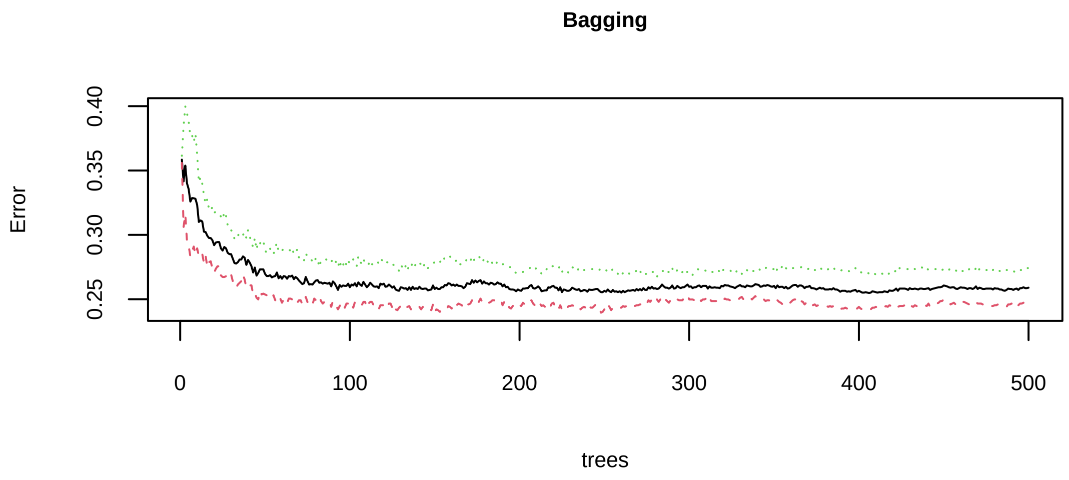
```
## [1] 0.2388889
```

The conditional inference tree yields slightly lower test error than the pruned CART tree. Thus, the conditional inference tree provided best predictive accuracy of the single trees.

Exercise 6: Fit a random forest to the MASQ data

We fit the ensembles:

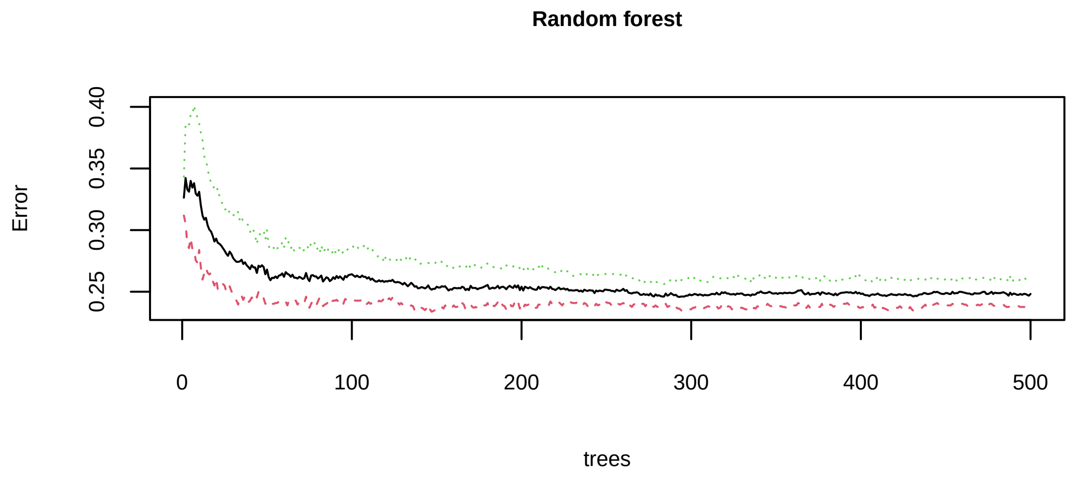
```r
library("randomForest")
set.seed(1)
bag.ens <- randomForest(D_DEPDYS ~ AD + AA + GDD + GDA + GDM + leeftijd + geslacht,
                        data = MASQ[train,], importance = TRUE, mtry = 7)
set.seed(1)
rf.ens <- randomForest(D_DEPDYS ~ AD + AA + GDD + GDA + GDM + leeftijd + geslacht,
                       data = MASQ[train,], importance = TRUE, mtry = sqrt(7))
plot(bag.ens, cex.lab = .7, cex.axis = .7, cex.main = .7, main = "Bagging")
```



For these data, the out-of-bag (OOB) error decreases fast with the first 100 trees. After 200-300 trees, the OOB error stabilizes.

Note that for binary classification, three curves are provided: The black curve shows the misclassification error, the green and red curves show the classification error in each of the classes (comparable to sensitivity and specificity).

```r
plot(rf.ens, cex.lab = .7, cex.axis = .7, cex.main = .7, main = "Random forest")
```



The OOB error plotted against the number of trees shows a very similar pattern as with the bagged ensemble.

Compute train MCR:

```
tab <- prop.table(table(MASQ[train, "D_DEPDYS"],
                        predict(bag.ens, newdata = MASQ[train,])))
1 - sum(diag(tab))  ## misclassification rate for bagging
```

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

```
tab <- prop.table(table(MASQ[train, "D_DEPDYS"],
                        predict(rf.ens, newdata = MASQ[train,])))
1 - sum(diag(tab)) ## misclassification rate for RF
```

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

Compute test MCR:

```
tab <- prop.table(table(MASQ[-train, "D_DEPDYS"],
predict(bag.ens, newdata = MASQ[-train,])))
1 - sum(diag(tab))  ## misclassification rate for bagging
```

```
## [1] 0.2388889
```

```
tab <- prop.table(table(MASQ[-train, "D_DEPDYS"],
predict(rf.ens, newdata = MASQ[-train,])))
1 - sum(diag(tab)) ## misclassification rate for RF
```

```
## [1] 0.2388889
```

We compute train SEL:

```
mean(((as.numeric(MASQ[train, "D_DEPDYS"]) - 1) -
        predict(bag.ens, newdata = MASQ[train,], type = "prob")[ , 2])^2)
```

```
## [1] 0.02466598
```

```
mean(((as.numeric(MASQ[train, "D_DEPDYS"]) - 1) -
        predict(rf.ens, newdata = MASQ[train,], type = "prob")[ , 2])^2)
```

```
## [1] 0.02431729
```

We compute test SEL:

```
mean(((as.numeric(MASQ[-train, "D_DEPDYS"]) - 1) -
        predict(bag.ens, newdata = MASQ[-train,], type = "prob")[ , 2])^2)
```

```
## [1] 0.1765018
```

```
mean(((as.numeric(MASQ[-train, "D_DEPDYS"]) - 1) -
        predict(rf.ens, newdata = MASQ[-train,], type = "prob")[ , 2])^2)
```

```
## [1] 0.1723298
```

Note that the predict method returns predicted probabilities for both classes, for objects of class `randomForest`. Therefore, we selected the second column of the returned probabilities (`[ , 2]`).

Test MCRs are identical for the bagged ensemble and random forest. Test SEL is lower for the random forest.

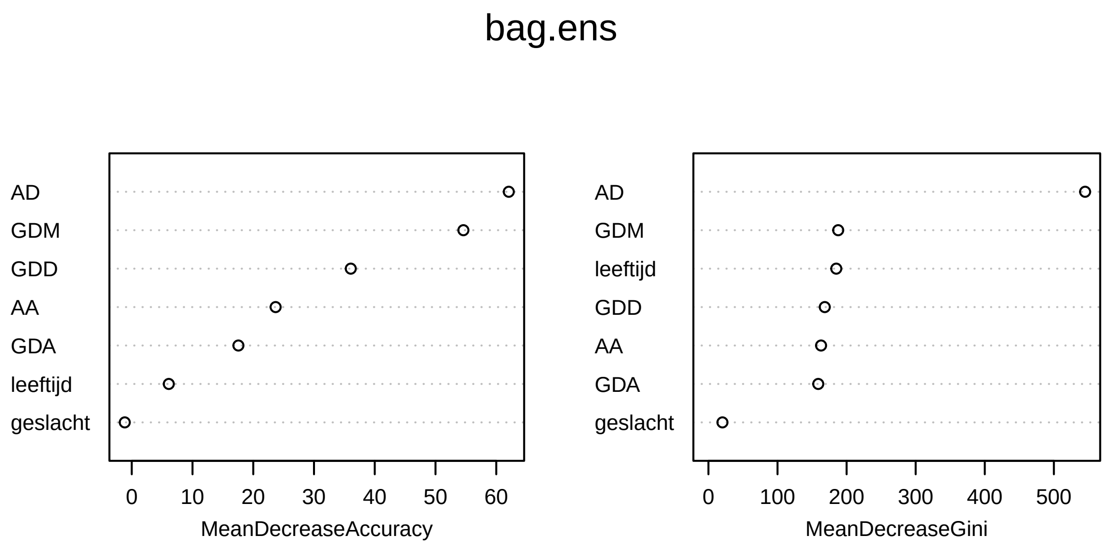
Interpretation

We inspect variable importances:

```
importance(bag.ens)
```

```
##                   0          1 MeanDecreaseAccuracy MeanDecreaseGini
## AD        26.008074 46.6261372            62.061330        545.20309
## AA        11.210061 19.5952066            23.683674        163.04057
## GDD       24.875687 18.2097877            36.050773        168.41636
## GDA       18.096720  3.1288186            17.552676        158.92755
## GDM       24.393229 39.5161589            54.590675        187.79771
## leeftijd   2.752565  5.6442541             6.098998        185.04514
## geslacht -1.634730 -0.1247933            -1.150591         20.31331
```

```
varImpPlot(bag.ens, cex = .7, cex.main = .7)
```



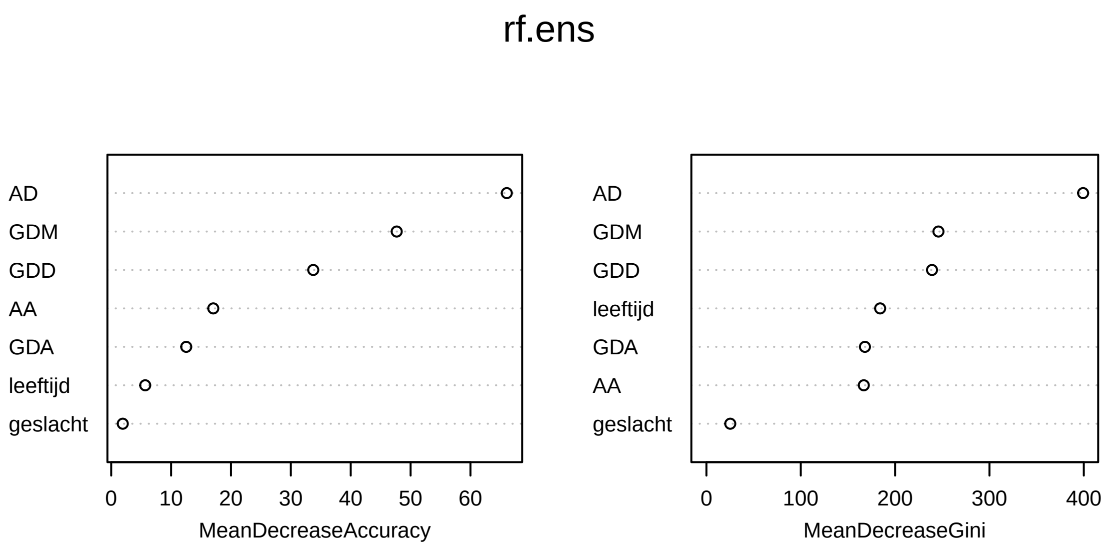
According to the reduction in MSE for the out-of-bag observations (left panel) if the values of each predictor variable are permuted, the AD (anhedonic depression), GDM (general distress mixed), and GDD (general distress depression) are the most important predictors of a depressive disorder diagnosis.

According to the improvement in node purity (i.e., training error; right panel), AD, leeftijd (age) and GDM are the most important predictors of of a depressive disorder diagnosis.

```
importance(rf.ens)
```

```
##                  0         1 MeanDecreaseAccuracy MeanDecreaseGini
## AD       33.630440 48.266024            66.064052        399.27306
## AA        6.599139 15.579649            17.047234        166.79307
## GDD      22.394085 16.774428            33.745150        239.10990
## GDA      13.584839  1.184124            12.532481        168.00515
## GDM      21.001515 36.662315            47.677333        245.91414
## leeftijd  1.773007  5.834744             5.682049        184.13887
## geslacht  1.344184  1.423987             1.931168         25.26403
```

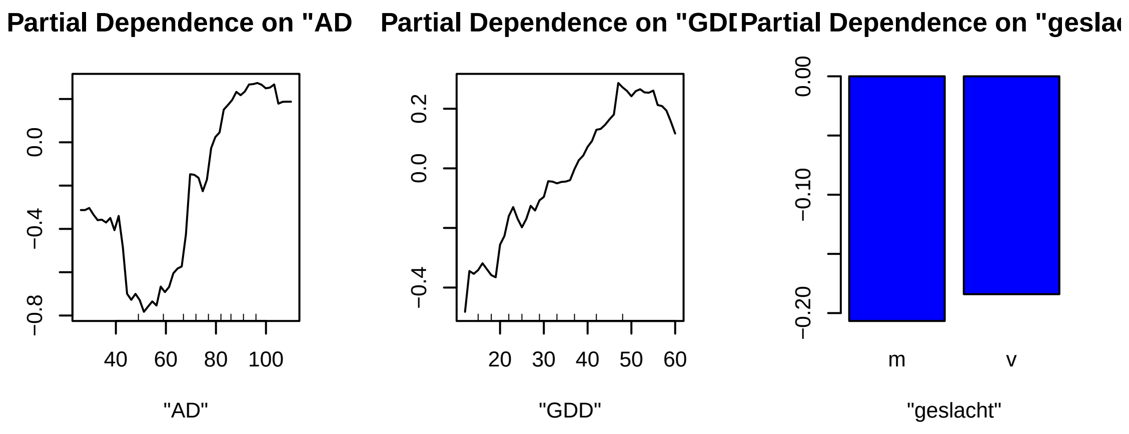
```
varImpPlot(rf.ens, cex = .7, cex.main = .7)
```



The AD, GDM and GDD scales appear most important in the random forest.

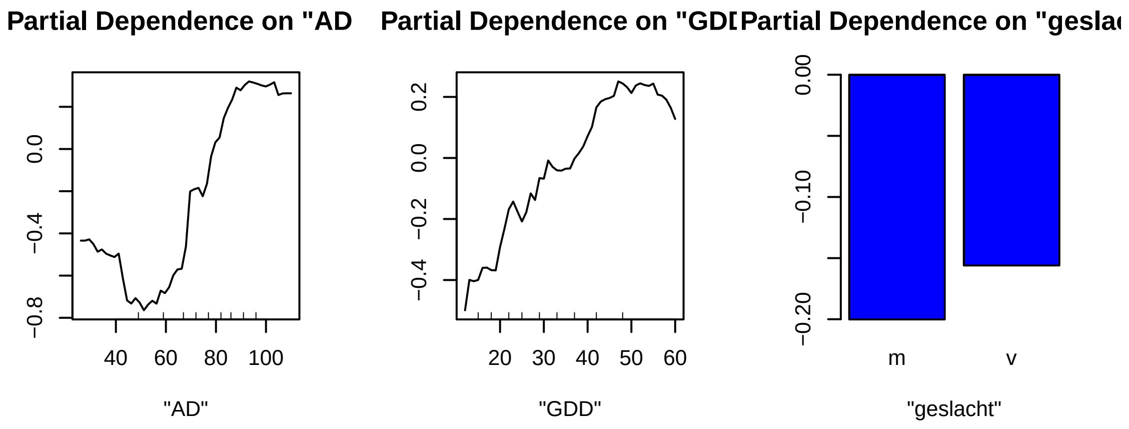
We request partial dependence plots for the bagged ensemble:

```
par(mfrow = c(1, 3))
partialPlot(bag.ens, x.var = "AD", pred.data = MASQ[train,], which.class = "1")
partialPlot(bag.ens, x.var = "GDD", pred.data = MASQ[train,], which.class = "1")
partialPlot(bag.ens, x.var = "geslacht", pred.data = MASQ[train,], which.class = "1")
```



Note that we have to specify the appropriate class label for these plots if we perform classification, otherwise we get partial dependence plots for the effect on the probability of belonging to the first ("0", non-depressed) class.

```r
par(mfrow = c(1, 3))
partialPlot(rf.ens, x.var = "AD", pred.data = MASQ[train,], which.class = "1")
partialPlot(rf.ens, x.var = "GDD", pred.data = MASQ[train,], which.class = "1")
partialPlot(rf.ens, x.var = "geslacht", pred.data = MASQ[train,], which.class = "1")
```



Exercise 7: Fit a gradient boosted ensemble to the MASQ data

```r
library("gbm")
```

```
## Warning: package 'gbm' was built under R version 4.1.1
```

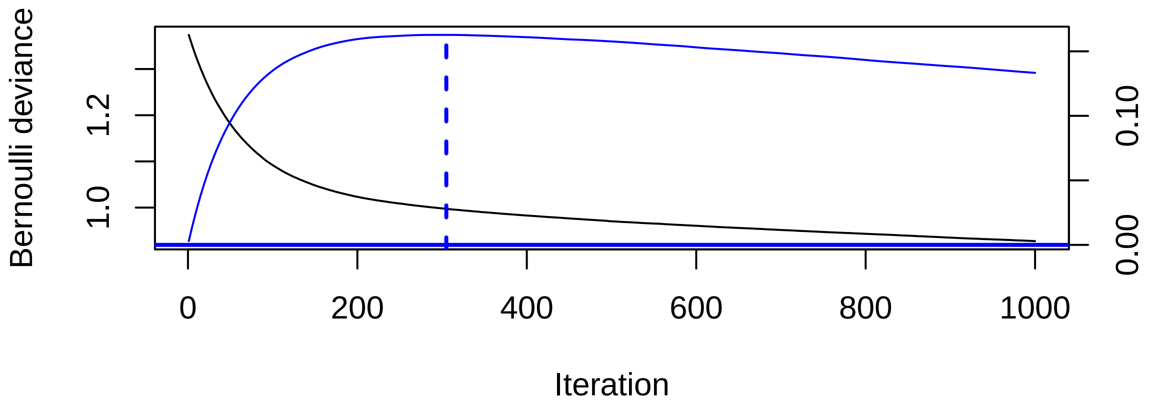
```
## Loaded gbm 2.1.8
```

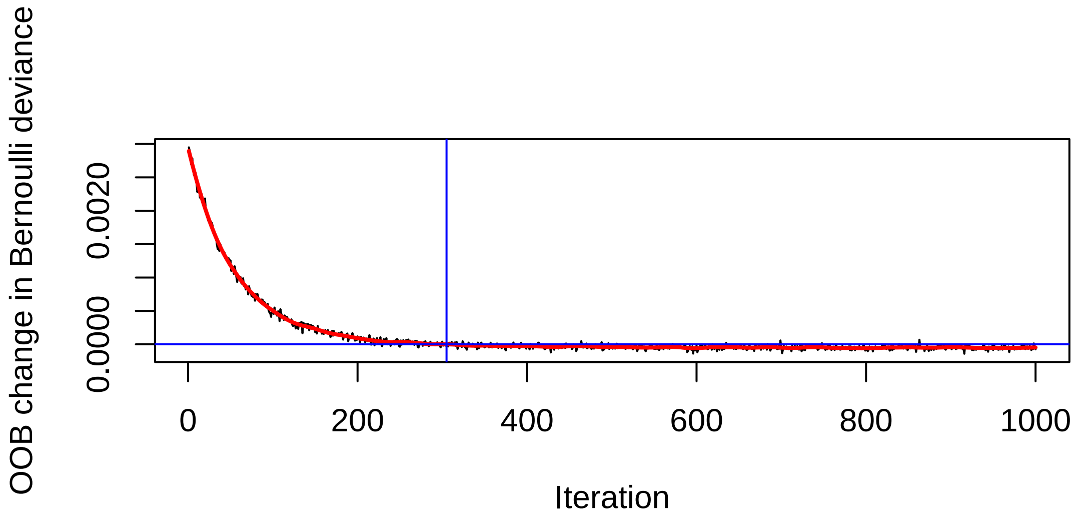
```r
set.seed(1)
MASQ$D_DEPDYS <- as.numeric(MASQ$D_DEPDYS) - 1 ## gbm wants a numeric response
boost.ens <- gbm(D_DEPDYS ~ ., data = MASQ[train, ], n.trees = 1000,
                 shrinkage = .01, interaction.depth = 4,
                 distribution = "bernoulli")
```

The `n.trees` argument controls the number of generated trees, which defaults to 100. The `bag.fraction` argument controls the fraction of training set observations randomly generated to fit each tree in the ensemble. Tree depth is controlled by argument `interaction.depth`, which defaults to 1 (trees with a single split, i.e., 2 terminal nodes, i.e., main effects only). The learning rate is controlled by the `shrinkage argument`, which defaults to 0.001.

```r
gbm.perf(boost.ens, method = "OOB", oobag.curve = TRUE)
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
```





```
## [1] 305
## attr(,"smoother")
## Call:
## loess(formula = object$oobag.improve ~ x, enp.target = min(max(4,
##     length(x)/10), 50))
##
## Number of Observations: 1000
## Equivalent Number of Parameters: 40
## Residual Standard Error: 3.229e-05
```

The black curve in the first plot represents training error, which decreases as a function of the number of iterations (fitted trees). The blue curve represents the estimated cumulative improvement in the deviance as estimated based on OOB observations (we requested this through specifying `oobag.curve = TRUE`).

In the first plot, the vertical blue dotted line indicates at which iteration the OOB error starts increasing (instead of decreasing). In the second plot, we see that this is where the OOB change in deviance becomes negative instead of positive. Thus, this appears the optimal number of iterations (according to the OOB deviance).

We also obtained a warning that OOB generally underestimates the number of required iterations, so the initial value of 1000 might not be bad, also because the second plot indicates no big risk of overfitting (i.e., although the OOB change in deviance becomes negative, but it remains very close to 0).

```
test_y <- MASQ$D_DEPDYS[-train]
train_y <- MASQ$D_DEPDYS[train]
test_N <- length(test_y)
train_N <- nrow(MASQ)-test_N
preds_test <- predict(boost.ens, newdata = MASQ[-train,], type = "response")
```

```
## Using 1000 trees...
```

```
preds_train <- predict(boost.ens, newdata = MASQ[train,], type = "response")
```

```
## Using 1000 trees...
```

```
tab_test <- table(true = test_y, predicted = preds_test > .5)
1 - sum(diag(tab_test)) / test_N ## misclassification rate
```

```
## [1] 0.2375
```

```r
mean((test_y - preds_test)^2) ## brier score
```

```
## [1] 0.1675675
```

```r
tab_train <- table(true = train_y, predicted = preds_train > .5)
1 - sum(diag(tab_train)) / train_N ## misclassification rate
```

```
## [1] 0.2210636
```

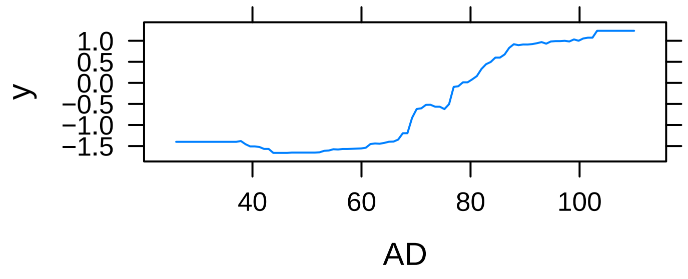
```r
mean((train_y - preds_train)^2) ## brier score
```

```
## [1] 0.1502326
```

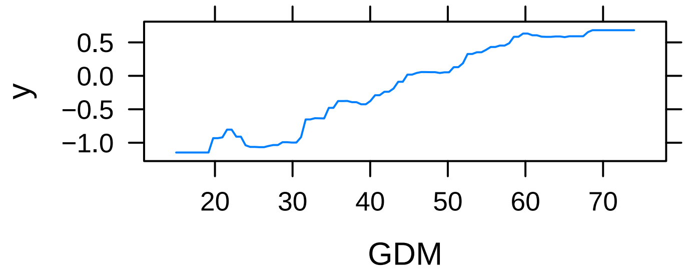
This seems to be the lowest test error we obtained thus far.

Interpretation

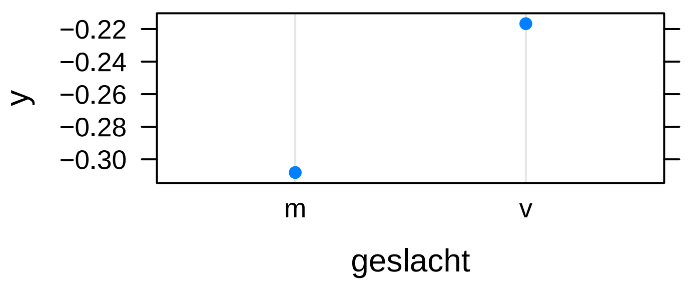
```r
plot(boost.ens, i.var = "AD")
```



```r
plot(boost.ens, i.var = "GDM")
```



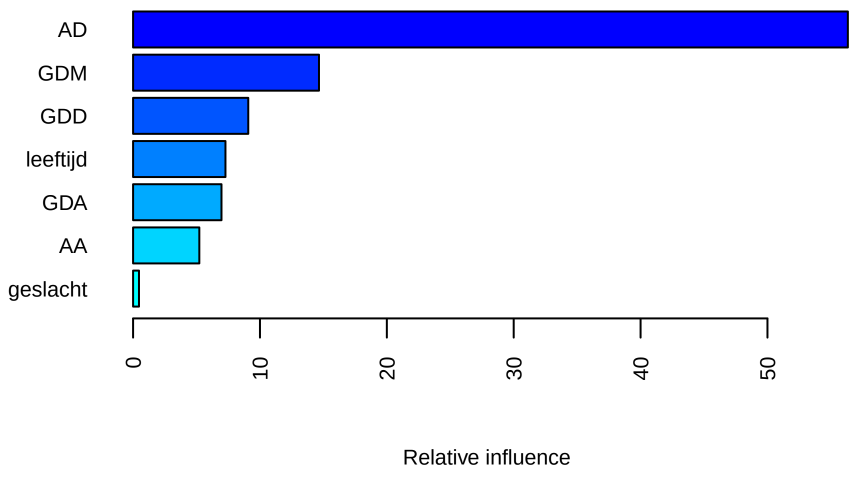
```r
plot(boost.ens, i.var = "geslacht")
```



The partial dependence plot suggest that the higher the AD and GDM scale scores, the higher the probability of having depression or dysthymia. Men appear to have a slightly lower probability of having a diagnosis, compared to women.

We request a summary of the model in order to obtain variable importances:

```
summary(boost.ens, cex.lab = .7, cex.axis = .7, cex.sub = .7, cex = .7, las = 2)
```



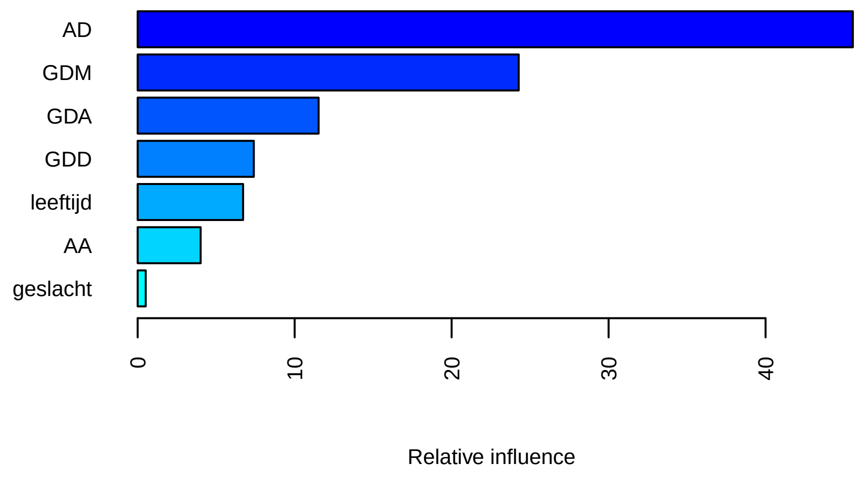
```
##                var    rel.inf
## AD              AD 56.3372131
## GDM            GDM 14.6522092
## GDD            GDD  9.0783275
## leeftijd leeftijd  7.2809713
## GDA            GDA  6.9666723
## AA              AA  5.2209153
## geslacht geslacht  0.4636913
```

Through the various `cex` arguments, we set the size of text and plotting symbols. Through the `las` argument, we specify the orientation of the axis labels (see `?par` for more explanation).

Like with the bagged and random forest ensembles, again we find that the AD variable is the strongest predictor of depressive disorder, followed by GDM and GDD.

Function `gbm()` return importances based on training error, by default (see `?summary.gbm`). We can obtain permutation importances (but note: these are computed using both in-bag and OOB observations, see also `?summary.gbm`) through specifying the method argument:

```
summary(boost.ens, cex.lab = .7, cex.axis = .7, cex.sub = .7, cex = .7,
        method = permutation.test.gbm, las = 2)
```



```
##        var    rel.inf
## 1       AD 45.5601747
## 2      GDM 24.2750184
## 3      GDA 11.5252439
## 4      GDD  7.3987442
## 5 leeftijd  6.7159431
## 6       AA  4.0113324
## 7 geslacht  0.5135434
```

Tuning parameters

```
library("caret")
grid <- expand.grid(shrinkage = c(.1, .01, .001),
                    n.trees = c(10, 100, 1000),
                    interaction.depth = 1:4,
                    n.minobsinnode = 10)
head(grid, 6)
```

```
##   shrinkage n.trees interaction.depth n.minobsinnode
## 1     0.100      10                 1             10
## 2     0.010      10                 1             10
## 3     0.001      10                 1             10
## 4     0.100     100                 1             10
## 5     0.010     100                 1             10
## 6     0.001     100                 1             10
```

```r
tail(grid, 6)
```

```
##    shrinkage n.trees interaction.depth n.minobsinnode
## 31     0.100     100                 4             10
## 32     0.010     100                 4             10
## 33     0.001     100                 4             10
## 34     0.100    1000                 4             10
## 35     0.010    1000                 4             10
## 36     0.001    1000                 4             10
```

Above, we created a grid of tuning parameters that predictive accuracy will be assessed over. As the `train()` function from package caret employs sub sampling to assess performance of the models, we have to set the random seed to allow for future replication of our results. Note that running the following code fits repeatedly fits boosted models for each set of parameter values, so it will take some time to run.

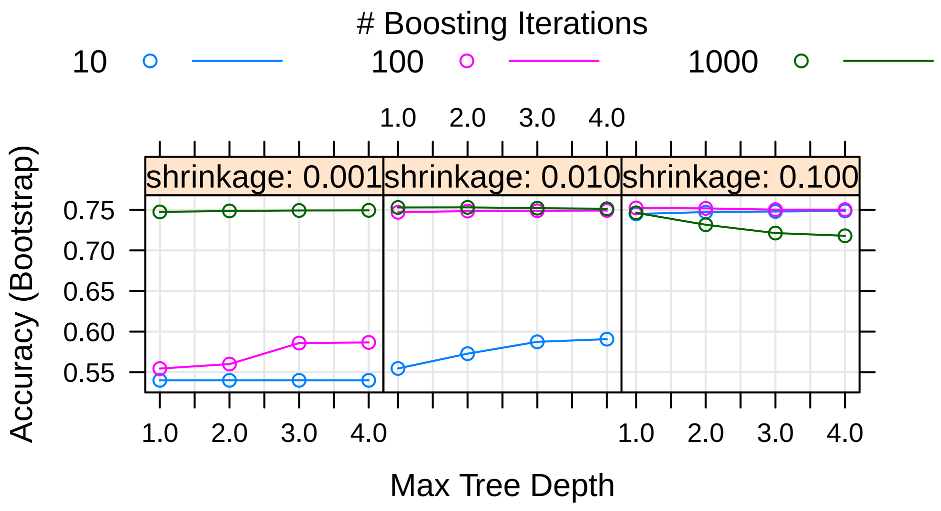
Note that `train()` requires a factor as the response for classification task, (unlike function `gbm()`), so I set the response to be a factor:

```r
traindat <- MASQ[train, ]
traindat$D_DEPDYS <- as.factor(traindat$D_DEPDYS)
set.seed(42)
gbmFit <- train(D_DEPDYS ~ . , data = traindat, tuneGrid = grid,
                distribution = "bernoulli", method = "gbm", verbose = FALSE)
```

Check out `?train` and `?trainControl` (which explains the arguments passed to argument `trControl`) to see what we did with this code. The default of bootstrap sampling with 25 repeats was used, (see `method` and `number` arguments of in `?trainControl`). Note that these predictive accuracies are estimated on test observations (i.e., 'OOB' observations).

We plot the results:

```r
plot(gbmFit)
```



Note that the highest accuracies are close to what we have obtained with the models we fitted before. The plot suggests that with higher values of shrinkage, we need less boosting iterations, which is as expected. Note that several combinations of parameter settings appear to yield similar accuracy.

Increasing tree depth to values > 1 seems not to make much different, only seems beneficial when there are less trees, suggesting mostly main effects of the predictor variables.

The best accuracy is obtained with:

```
gbmFit$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 18    1000                 2      0.01             10
```

These optimal settings differ, but not by much, from our original parameter settings.

We refit the ensemble using the parameter values that can be expected to optimize predictive accuracy:

```
set.seed(42)
boost.ens2 <- gbm(D_DEPDYS ~ ., data = MASQ[-train,], n.trees = 1000,
                  shrinkage = .01, interaction.depth = 2,
                  distribution = "bernoulli")
preds_test <- predict(boost.ens2, newdata = MASQ[-train,], type = "response")
```

```
## Using 1000 trees...
```

```
preds_train <- predict(boost.ens2, newdata = MASQ[train,], type = "response")
```

```
## Using 1000 trees...
```

```
tab_test <- table(true = test_y, predicted = preds_test > .5)
1 - sum(diag(tab_test)) / test_N
```

```
## [1] 0.1875
```

```
mean((test_y - preds_test)^2)
```

```
## [1] 0.135182
```

```
tab_train <- table(true = train_y, predicted = preds_train > .5)
1 - sum(diag(tab_train)) / train_N
```

```
## [1] 0.2502607
```

```
mean((train_y - preds_train)^2)
```

```
## [1] 0.1759091
```

Both MCR and SEL improved compared to the earlier boosted ensemble. The boosted ensemble with tuned parameters also outperformed the random forest and bagged ensemble fitted to these data previously. All fitted tree ensembles require all variables for making a prediction (because all variables have non-zero importances).

The CART tree requires only the AD scale for prediction. The ctree requires only the variables AD and GDD, and in some cases also GDD for prediction. They thus require substantially less information for prediction, but at a cost to predictive accuracy.

Who won?

The comparison is not totally fair: With SVMs and boosting, we tuned the parameter values using CV on the training data. Also, we gave the GAM two predictors less. Refitting the GAM is easy:

```
GAM2 <- gam(D_DEPDYS ~ s(AD) + s(AA) + s(GDD) + s(GDA) + s(GDM) +
              s(leeftijd) + geslacht,
           data = MASQ[train, ], method = "REML", family = "binomial")
summary(GAM2)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## D_DEPDYS ~ s(AD) + s(AA) + s(GDD) + s(GDA) + s(GDM) + s(leeftijd) +
##     geslacht
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.33822    0.07755  -4.361 1.29e-05 ***
## geslachtv    0.13314    0.09517   1.399    0.162
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##               edf Ref.df  Chi.sq  p-value
## s(AD)       4.243  5.238 148.912  < 2e-16 ***
## s(AA)       1.001  1.002   0.115  0.73584
## s(GDD)      2.185  2.793  11.185  0.00971 **
## s(GDA)      3.855  4.806  33.279 4.62e-06 ***
## s(GDM)      1.001  1.001  52.723  < 2e-16 ***
## s(leeftijd) 3.050  3.817  17.530  0.00137 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.331   Deviance explained =   27%
## -REML = 1478.1  Scale est. = 1         n = 2877
```

```
GAM2_preds_train <- predict(GAM2, newdata = MASQ[train, ], type = "response")
mean((MASQ[train, "D_DEPDYS"] - GAM2_preds_train)^2) ## Brier score
```

```
## [1] 0.1653101
```

```
tab_train <- prop.table(table(MASQ[train, "D_DEPDYS"], GAM2_preds_train > .5))
1 - sum(diag(tab_train)) ## MCR
```

```
## [1] 0.2384428
```

```
GAM2_preds_test <- predict(GAM2, newdata = MASQ[-train, ], type = "response")
mean((MASQ[-train, "D_DEPDYS"] - GAM2_preds_test)^2) ## Brier score
```

```
## [1] 0.1666466
```

```
tab_test <- prop.table(table(MASQ[-train, "D_DEPDYS"], GAM2_preds_test > .5))
1 - sum(diag(tab_test)) ## MCR
```

```
## [1] 0.2402778
```

The performance of bagging and random forest would have profited from tuning of parameters; most notably tree size. The defaults for arguments `maxnodes` and `nodesize` give very deep trees, which tends to not benefit predictive accuracy, especially with larger datasets. Also, the single tree might have benefitted from tuning tree depth.

As a benchmark for the Brier score, we take the Brier score and MCR that would result from using the mean and mode of the response variable on the training data for prediction:

```
p_hat_train <- mean(train_y)
mean(test_y != round(p_hat_train)) ## MCR
```

```
## [1] 0.4805556
```

```
mean((train_y - p_hat_train)^2) ## Brier
```

```
## [1] 0.2484161
```

```
mean(train_y != round(p_hat_train)) ## MCR
```

```
## [1] 0.4602016
```

```
mean((test_y - p_hat_train)^2) ## Brier
```

```
## [1] 0.2500362
```

	Brier train	MCR train	Brier test	MCR test	Brier test-train[a]
Benchmark	0.2484	0.4806	0.2500	0.4602	-0.0016
GAM	0.1653	0.2384	**0.1666**	0.2403	**0.0019**
SVM linear	-	0.2416	-	0.2486	
SVM radial basis	-	0.2103	-	0.2486	
tree	0.1706	0.2457	0.1739	**0.2389**	**0.0033**
random forest	0.0243	0.0000	0.1723	**0.2389**	0.1480
bagging	0.0247	0.0000	0.1765	**0.2389**	0.1518
boosting	0.1502	0.2211	**0.1676**	**0.2375**	**0.0174**
boosting (tuned)	0.1759	0.2503	**0.1352**	**0.1875**	-0.0407

[a] The higher the value of (Brier score on test data) - (Brier score on train data), the more overfitting has occurred (i.e., the model more closely adopted to the training data, than could be generalized to the test data)

Boldfaced values indicate the top three methods. We can draw the following conclusions:

- Tuned boosting outperforms all methods, followed by untuned boosting and then the GAM.
- All models do substantially better than the benchmark. Between models, the performance differences are much smaller. We thus gain the most by fitting *any* reasonable model. Sophisticated methods only provide marginal improvements.
- The GAM and tree perform very well out of the box. Given the very simple structure of the tree and that it uses only three of the seven possible predictors, it has pretty impressive performance.
- Random forest and bagging show a strong tendency to overfit here. Likely, their performance can be much improved and overfitting much reduced by carefully tuning tree size.
- The rankings according to misclassification rate (classification quality) and Brier score (quality of predicted probabilities) differ somewhat.