

# Answers to exercises Session 5

Marjolein Fokkema

## Exercise 1: Fit a smoothing spline

Read in data:

```
load("MASQ.Rda")
set.seed(1)
train <- sample(1:nrow(MASQ), size = nrow(MASQ)*.8)
summary(MASQ)
```

```
## D_DEPDYS      AD      AA      GDD      GDA
## 0:1927  Min.   : 26.00  Min.   :17.00  Min.   :12.00  Min.   :11.0
## 1:1670  1st Qu.: 64.00  1st Qu.:22.00  1st Qu.:20.00  1st Qu.:19.0
##          Median : 77.00  Median :28.00  Median :29.00  Median :24.0
##          Mean   : 75.05  Mean   :32.01  Mean   :30.64  Mean   :25.4
##          3rd Qu.: 88.00  3rd Qu.:39.00  3rd Qu.:40.00  3rd Qu.:31.0
##          Max.   :110.00  Max.   :83.00  Max.   :60.00  Max.   :54.0
##      GDM      leeftijd      geslacht
## Min.   :15.0    Min.   :17.0    m:1317
## 1st Qu.:31.0    1st Qu.:28.0    v:2280
## Median :40.0    Median :38.0
## Mean   :40.6    Mean   :38.8
## 3rd Qu.:50.0    3rd Qu.:48.0
## Max.   :75.0    Max.   :91.0
```

```
round(cor(MASQ[train, sapply(MASQ, is.numeric)]), digits = 2)
```

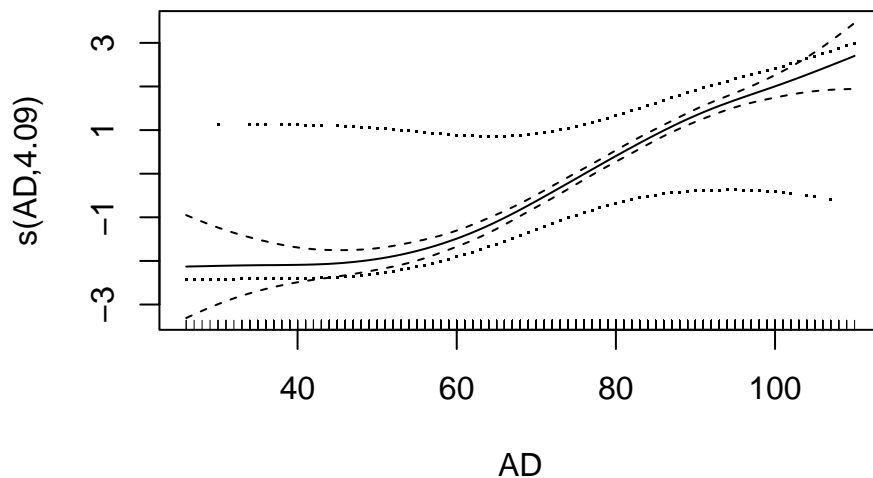
```
##      AD      AA      GDD      GDA      GDM leeftijd
## AD      1.00 0.51 0.79 0.61 0.74 0.01
## AA      0.51 1.00 0.58 0.79 0.70 0.01
## GDD      0.79 0.58 1.00 0.72 0.81 -0.07
## GDA      0.61 0.79 0.72 1.00 0.80 -0.05
## GDM      0.74 0.70 0.81 0.80 1.00 -0.04
## leeftijd 0.01 0.01 -0.07 -0.05 -0.04 1.00
```

Fit a smoothing spline of the AD variable to predict D\_DEPDYS:

```
library("mgcv")
GAM <- gam(D_DEPDYS ~ s(AD, bs = "cr"),
           data = MASQ[train, ], method = "REML", family = "binomial")
summary(GAM)
```

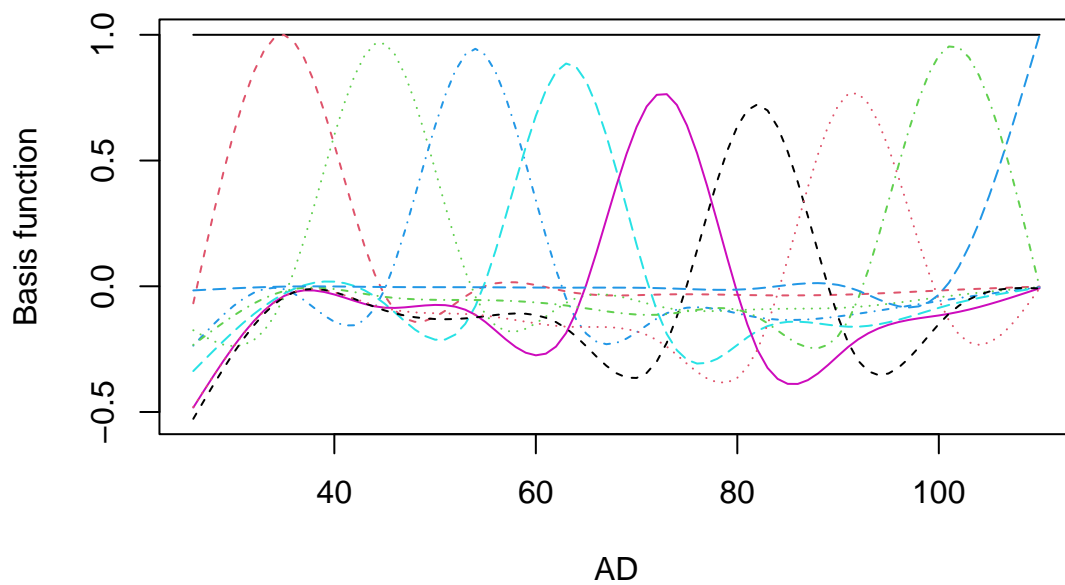
```
##
## Family: binomial
## Link function: logit
##
## Formula:
## D_DEPDYS ~ s(AD, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.24089    0.04751  -5.07 3.98e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##             edf Ref.df Chi.sq p-value
## s(AD) 4.09  5.041  672.4  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.297   Deviance explained = 23.7%
## -REML = 1522.2   Scale est. = 1           n = 2877
```

```
plot(GAM, residuals = TRUE)
```



Inspect the basis functions that were created for AD:

```
mod_mat <- model.matrix(GAM)
matplot(sort(MASQ$AD[train]), mod_mat[order(MASQ$AD[train]), ], type = "l",
        xlab = "AD", ylab = "Basis function")
```



The curves show the basis functions that were created to fit the spline. Each basis function obtained a coefficient, in order to fit the smoothing spline of AD. If you would be interested in the coefficients for each function (but normally we would simply look at results of `summary` and `plot`), you can extract them as follows:

```
coef(GAM)
```

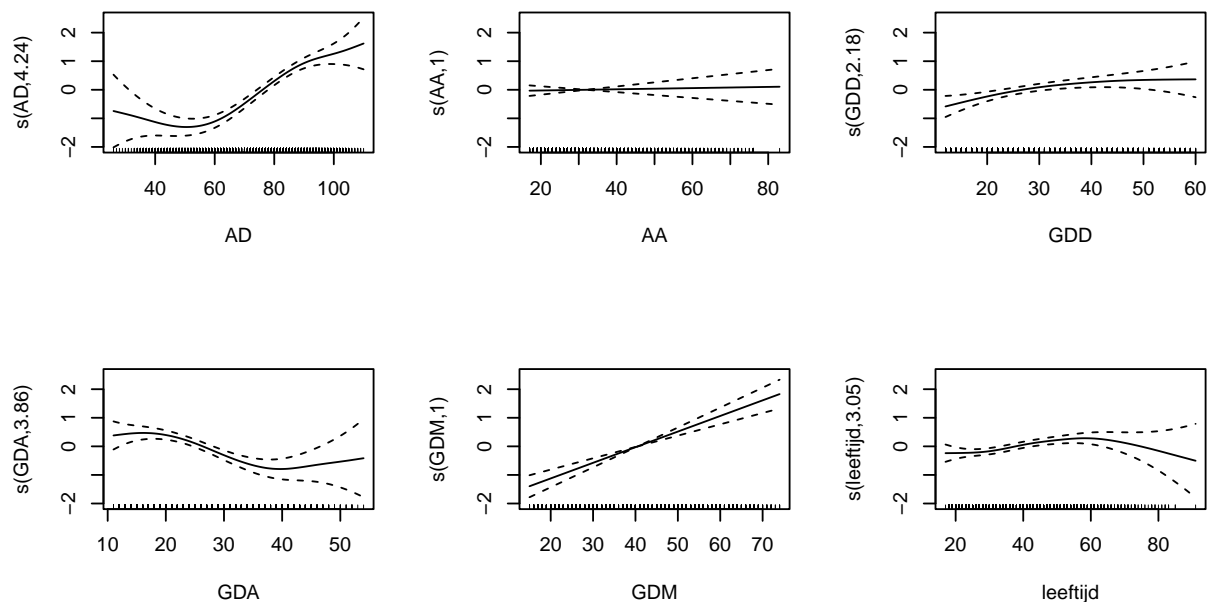
```
## (Intercept)      s(AD).1      s(AD).2      s(AD).3      s(AD).4      s(AD).5
## -0.2408904 -1.9544694 -1.6885841 -1.3148046 -0.5268088  0.6694791
##      s(AD).6      s(AD).7      s(AD).8      s(AD).9
##   1.7240745   2.4539899   2.5401186   2.7354293
```

## Exercise 2: Multiple predictors

```
library("mgcv")
GAM <- gam(D_DEPDYS ~ s(AD) + s(AA) + s(GDD) + s(GDA) + s(GDM) + s(leeftijd) + geslacht,
           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) + 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  5.238 148.910 < 2e-16 ***
## s(AA)         1.001  1.002   0.115  0.73591
## 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

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



All predictors except `geslacht` have a significant effect. The chi-square indicates AD explains the most variance, followed by GDM. The `edf` (empirical degrees of freedom) suggest strongest non-linearity for the AD scale. The AA and GDM scales have effects close to linear, as also suggested by the plots.

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

```
y_train <- as.numeric(MASQ[train, "D_DEPDYS"]) - 1
y_test  <- as.numeric(MASQ[-train, "D_DEPDYS"]) - 1

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

## [1] 0.1653101

tab_train <- prop.table(table(MASQ[train, "D_DEPDYS"], gam_preds_train > .5)) ## confusion matrix
tab_train

##
##      FALSE      TRUE
## 0 0.4174487 0.1223497
## 1 0.1160932 0.3441084

1 - sum(diag(tab_train)) ## MCR

## [1] 0.2384428
```

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

## [1] 0.1666467

tab_test <- prop.table(table(MASQ[-train, "D_DEPDYS"], gam_preds_test > .5)) ## confusion matrix
tab_test

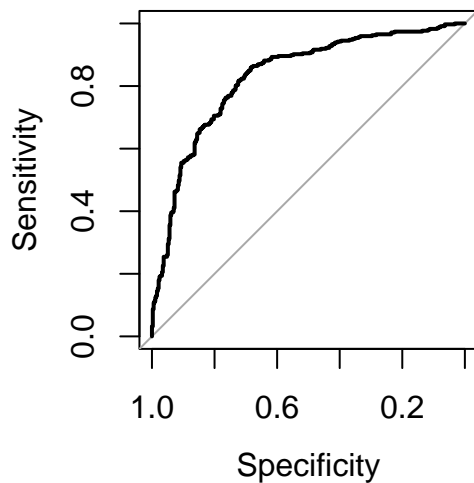
##
##      FALSE      TRUE
## 0 0.4027778 0.1166667
## 1 0.1236111 0.3569444

1 - sum(diag(tab_test)) ## MCR

## [1] 0.2402778
```

The Brier score and confusion matrices are quite similar between training and test data, indicating little overfitting. Alternatively, we may want to compute an ROC curve:

```
library("pROC")
plot(roc(resp = y_test, pred = gam_preds_test))
```



```
auc(resp = y_test, pred = gam_preds_test)
```

```
## Area under the curve: 0.8295
```

## Exercise 3: Fit support vector machines

```
library("e1071")
cost <- c(.001, .01, .1, 1, 5, 10, 100)
set.seed(42)
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)
```

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

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

```
## Test performance
svm_preds_test <- predict(svmfit, newdata = MASQ[-train, ])
tab_test <- prop.table(table(MASQ[-train, "D_DEPDYS"], svm_preds_test))
tab_test
```

```
## svm_preds_test
##      0      1
## 0 0.4000000 0.1194444
## 1 0.1291667 0.3513889
```

```
1 - sum(diag(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

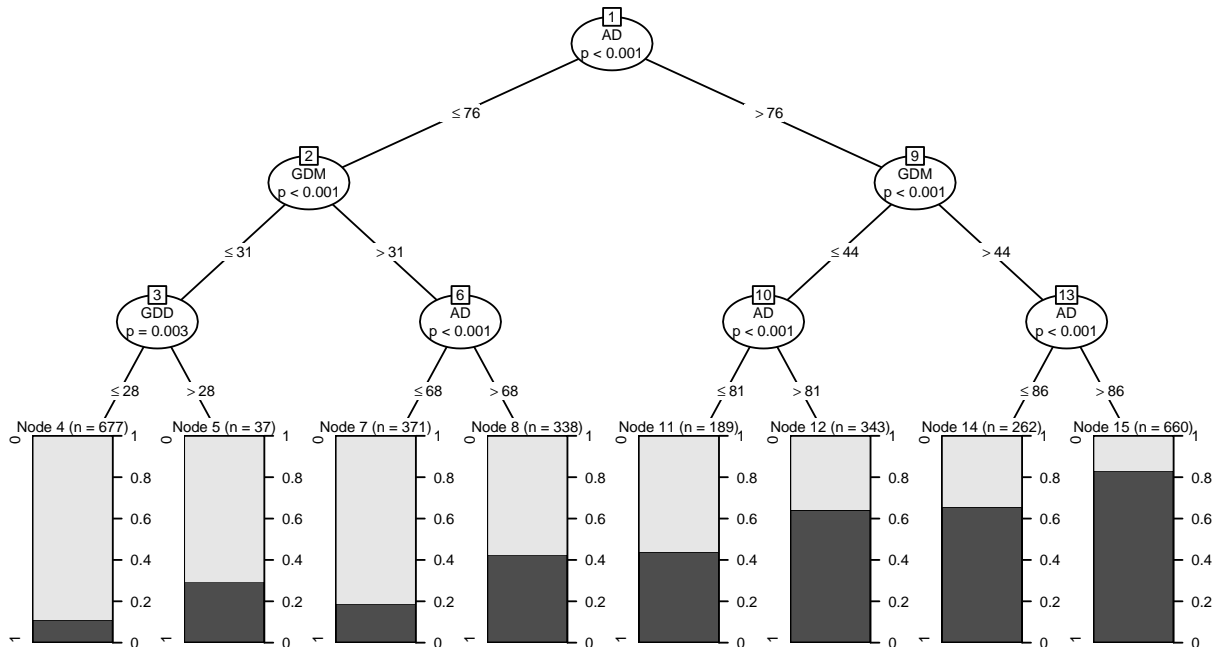
## [1] 0.2486111
```

The radial basis kernel did not improve performance on test data, over the linear kernel.



## Exercise 4: Fit a conditional inference tree

```
library("partykit")
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.

```
## 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
```

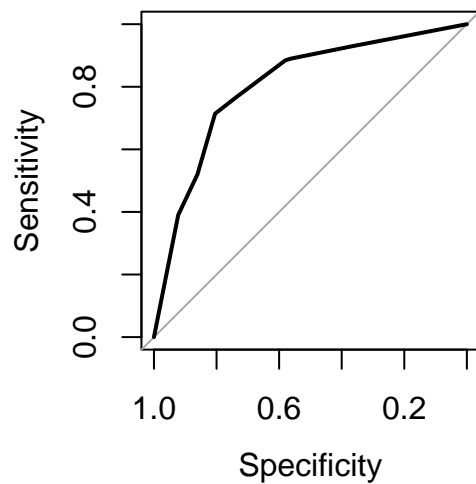
```
tab_test <- prop.table(table(MASQ[-train, "D_DEPDYS"], ct_preds_test > .5)) ## confusion matrix
1 - sum(diag(tab_test)) ## MCR
```

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

```
## AUC on test observations
plot(roc(resp = y_test, pred = ct_preds_test))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```



```
auc(resp = y_test, pred = ct_preds_test)
```

```
## Setting levels: control = 0, case = 1
```

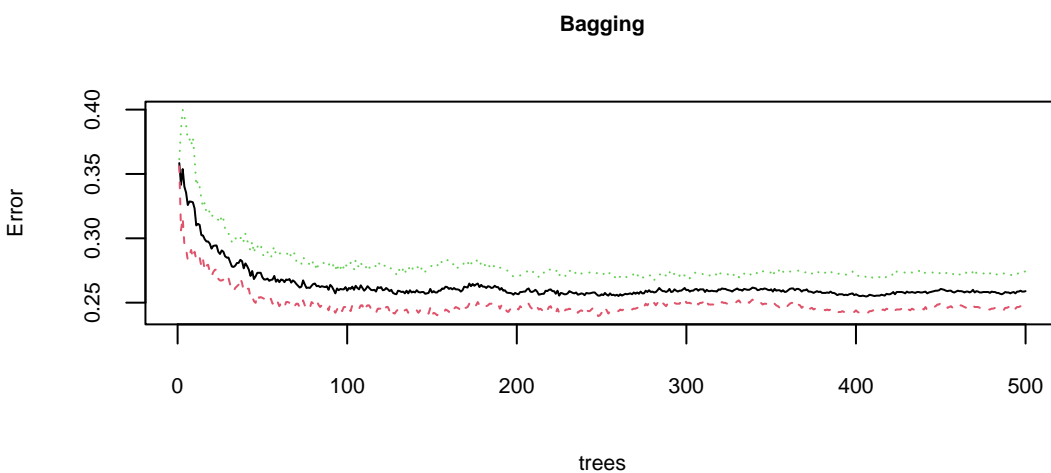
```
## Setting direction: controls < cases
```

```
## Area under the curve: 0.805
```

## Exercise 5: Fit a bagged ensemble and random forest

Fit the ensembles:

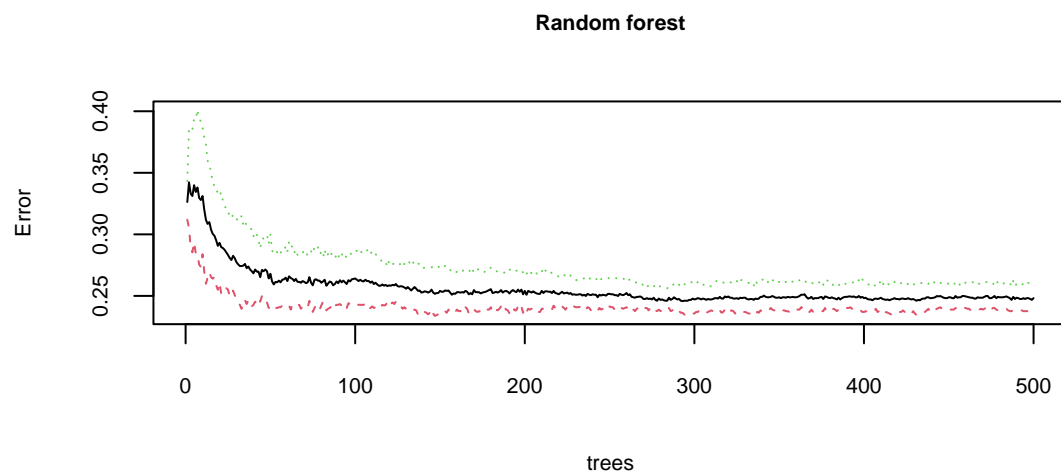
```
library("randomForest")
set.seed(1)
bag.ens <- randomForest(D_DEPDYS ~ AD + AA + GDD + GDA + GDM + leeftijd + geslacht,
                        data = MASQ[train,], importance = TRUE, mtry = 7)
set.seed(1)
rf.ens <- randomForest(D_DEPDYS ~ AD + AA + GDD + GDA + GDM + leeftijd + geslacht,
       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

```
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,])))
tab
```

```
##
##           0           1
##  0 0.5397984 0.0000000
##  1 0.0000000 0.4602016
```

```
1 - sum(diag(tab)) ## misclassification rate for bagging
```

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

```
tab <- prop.table(table(MASQ[train, "D_DEPDYS"],
                        predict(rf.ens, newdata = MASQ[train,])))
tab
```

```
##
##           0           1
##  0 0.5397984 0.0000000
##  1 0.0000000 0.4602016
```

```
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,])))
tab
```

```
##
##           0           1
##  0 0.3958333 0.1236111
##  1 0.1152778 0.3652778
```

```
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,])))
tab
```

```
##
##           0           1
##  0 0.3972222 0.1222222
##  1 0.1166667 0.3638889
```

```
1 - sum(diag(tab)) ## misclassification rate for RF
```

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

We compute squared error on predicted probabilities (Brier score) for the training data:

```
predict(bag.ens, newdata = MASQ[train,], type = "prob")[1:10, ]
```

```
##           0           1
## 1017 0.982 0.018
##  679 0.816 0.184
## 2177 0.960 0.040
##  930 0.058 0.942
## 1533 0.788 0.212
##  471 0.998 0.002
## 2347 0.812 0.188
##  270 0.966 0.034
## 1211 0.922 0.078
## 3379 0.878 0.122
```

Note that the predict method returns predicted probabilities for both classes, for objects of class randomForest.

Therefore, we select the second column of the returned probabilities ([ , 2]):

```
bag_preds_train <- predict(bag.ens, newdata = MASQ[train,], type = "prob")[ , 2]
rf_preds_train <- predict(rf.ens, newdata = MASQ[train,], type = "prob")[ , 2]
mean((y_train - bag_preds_train)^2)
```

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

```
mean((y_train - rf_preds_train)^2)
```

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

And for the test data:

```
bag_preds_test <- predict(bag.ens, newdata = MASQ[-train,], type = "prob")[ , 2]
rf_preds_test <- predict(rf.ens, newdata = MASQ[-train,], type = "prob")[ , 2]
mean((y_test - bag_preds_test)^2)
```

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

```
mean((y_test - rf_preds_test)^2)
```

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

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

## Interpretation

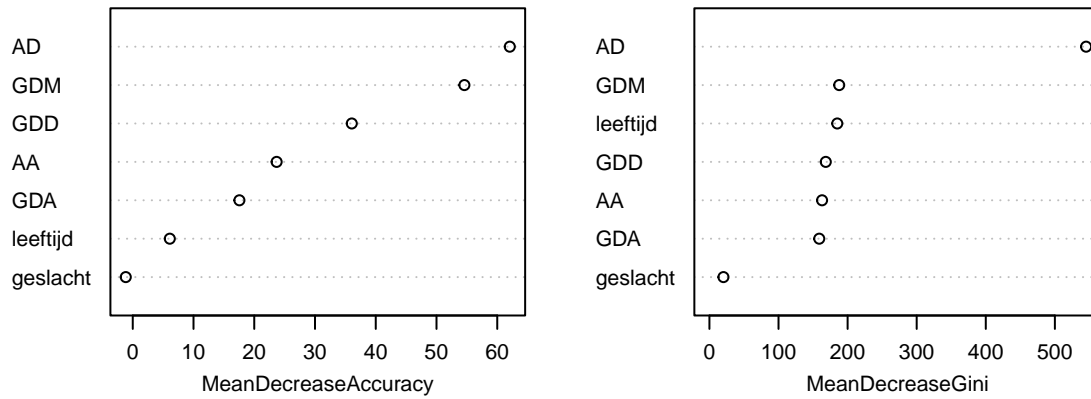
We inspect variable importances:

```
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)
```

## bag.ens



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.

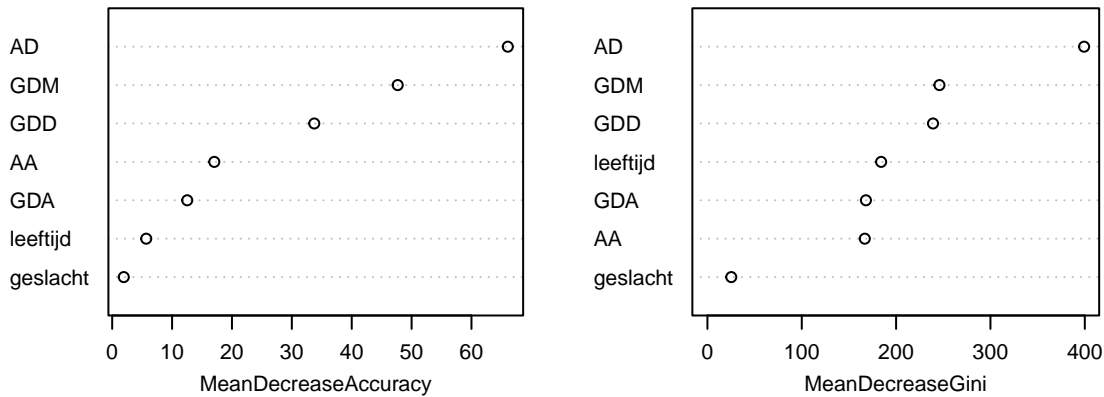
According to the improvement in node purity (i.e., training error; right panel), AD, leeftijd (age) and GDM are the most important predictors of a depressive disorder diagnosis.

```
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)
```

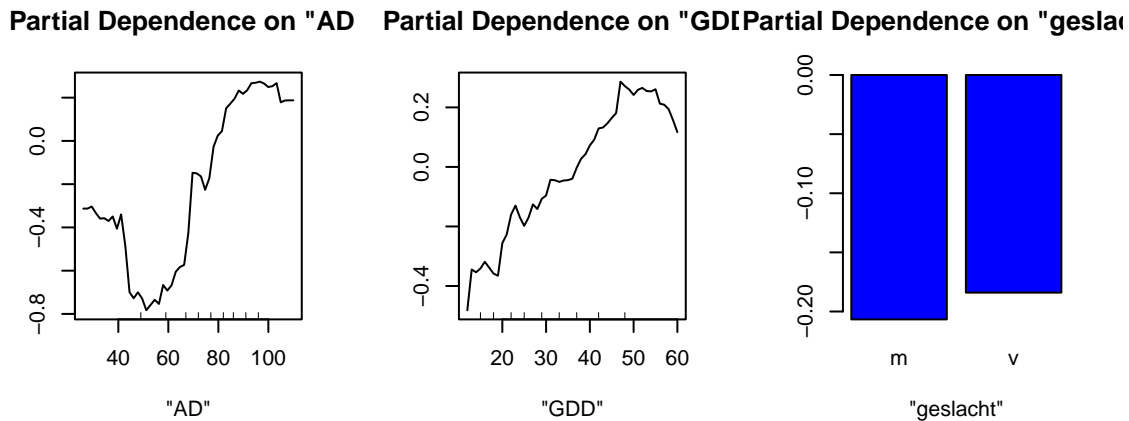
## rf.ens



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")
```

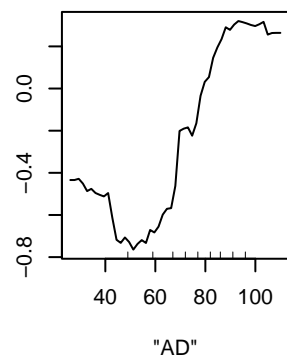


Note that we have to specify the appropriate class label for these plots if we perform classification, otherwise we get partial dependence plots for the effect on the probability of belonging to the first ("0", non-depressed) class.

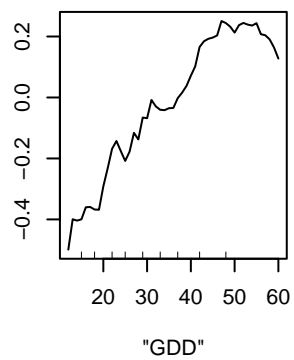
```
par(mfrow = c(1, 3))
partialPlot(rf.ens, x.var = "AD", pred.data = MASQ[train,], which.class = "1")
partialPlot(rf.ens, x.var = "GDD", pred.data = MASQ[train,], which.class = "1")
partialPlot(rf.ens, x.var = "geslacht", pred.data = MASQ[train,], which.class = "1")
```



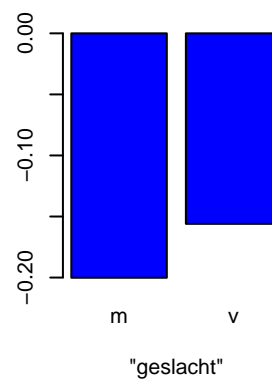
Partial Dependence on "AD"



Partial Dependence on "GDD"



Partial Dependence on "geschlecht"



## Exercise 6: Fit a gradient boosted ensemble

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

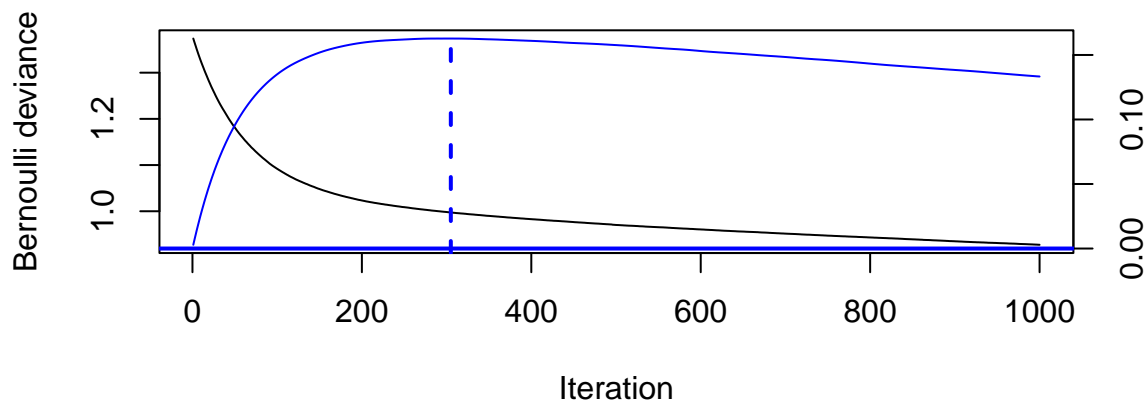
```
## Loaded gbm 2.1.8.1
```

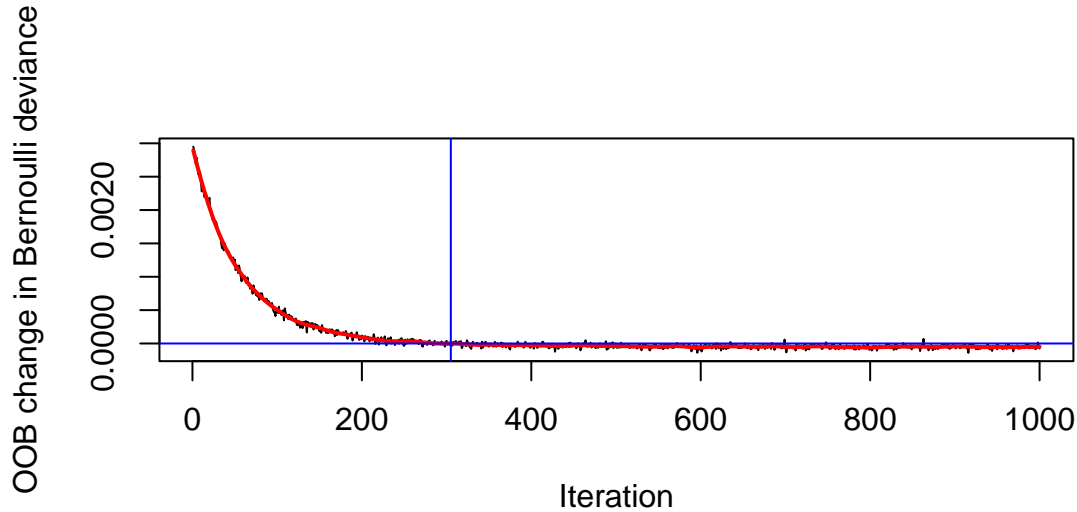
```
set.seed(1)
MASQ$D_DEPDYS <- as.numeric(MASQ$D_DEPDYS) - 1 ## note: gbm wants a numeric response
boost.ens <- gbm(D_DEPDYS ~ ., data = MASQ[train, ], n.trees = 1000,
                 shrinkage = .01, interaction.depth = 4,
                 distribution = "bernoulli")
```

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.

```
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).

```
## Train performance
gbm_preds_train <- predict(boost.ens, newdata = MASQ[train,], type = "response")
```

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

```
tab_train <- prop.table(table(true = y_train, predicted = gbm_preds_train > .5))
tab_train
```

```
##      predicted
## true   FALSE    TRUE
##  0 0.4209246 0.1188738
##  1 0.1021898 0.3580118
```

```
1 - sum(diag(tab_train)) ## misclassification rate
```

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

```
mean((y_train - gbm_preds_train)^2) ## brier score
```

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

```
## Test performance
```

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

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

```
tab_test <- prop.table(table(true = y_test, predicted = gbm_preds_test > .5))  
tab_test
```

```
##      predicted  
## true      FALSE      TRUE  
##  0 0.4000000 0.1194444  
##  1 0.1180556 0.3625000
```

```
1 - sum(diag(tab_test)) ## misclassification rate
```

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

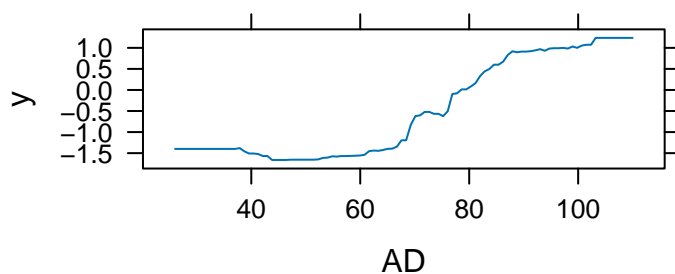
```
mean((y_test - gbm_preds_test)^2) ## brier score
```

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

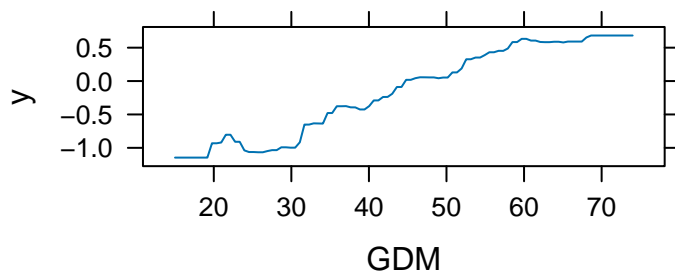
This seems to be the lowest test error we obtained thus far. We will further improve by tuning the parameters in the next exercise.

## Interpretation

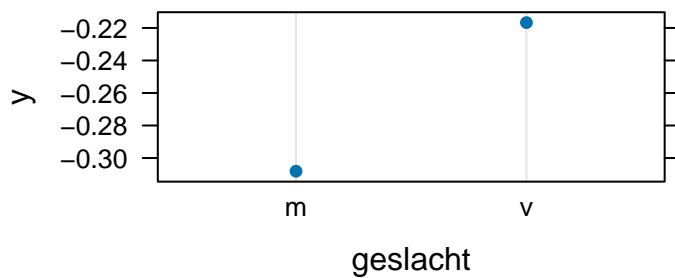
```
plot(boost.ens, i.var = "AD")
```



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



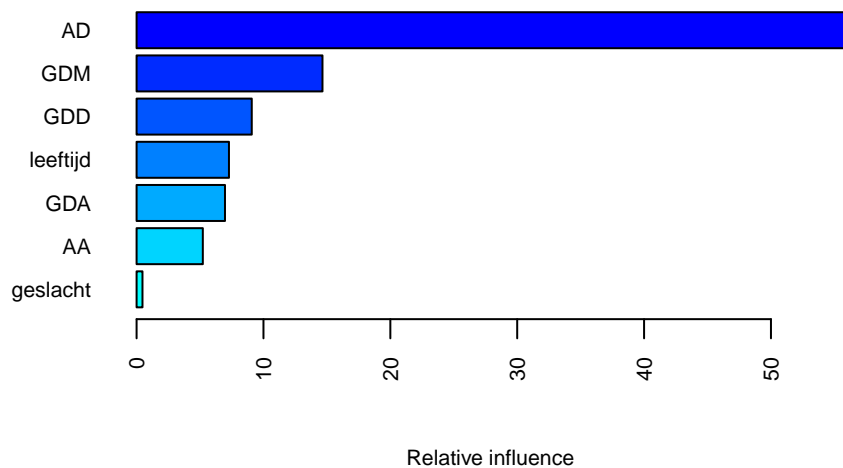
```
plot(boost.ens, i.var = "geslacht")
```



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)
```



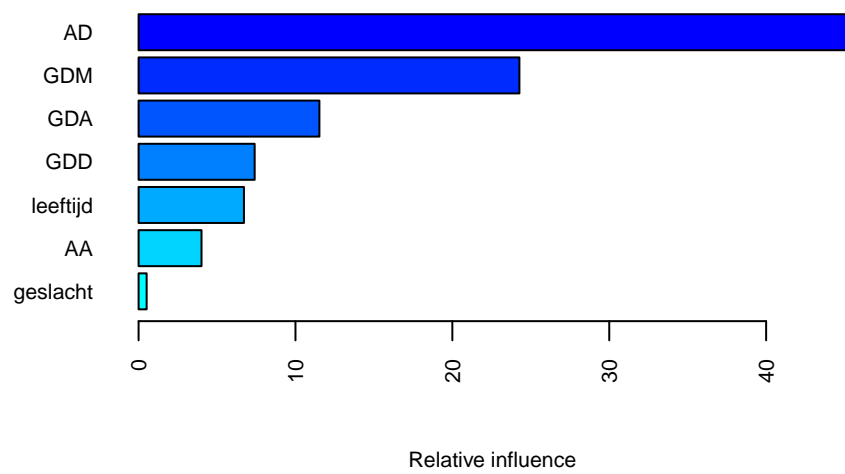
```
##          var    rel.inf
## AD          AD 56.3372131
## GDM         GDM 14.6522092
## GDD         GDD  9.0783275
## leeftijd leeftijd 7.2809713
## GDA         GDA  6.9666723
## AA          AA  5.2209153
## geslacht geslacht 0.4636913
```

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
```

## Exercise 7: Tuning parameters of gradient boosting

```
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
```

```
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:

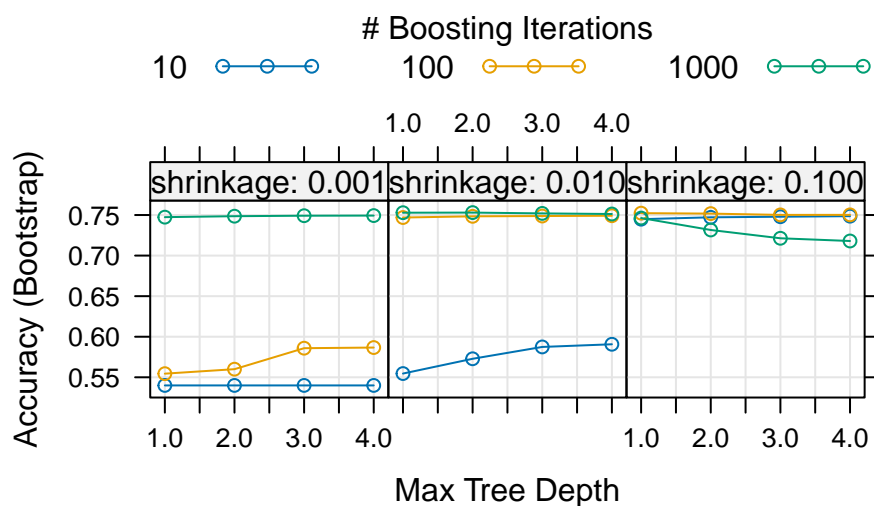
```
traindat <- MASQ[train, ]
traindat$D_DEPDYS <- as.factor(traindat$D_DEPDYS)
set.seed(42)
gbmFit <- train(D_DEPDYS ~ ., data = traindat, tuneGrid = grid,
               distribution = "bernoulli", method = "gbm", verbose = FALSE)
```

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:

```
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")

## Train performance
gbm2_preds_train <- predict(boost.ens2, newdata = MASQ[train,], type = "response")
```

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

```
tab_train <- prop.table(table(true = y_train, predicted = gbm2_preds_train > .5))
1 - sum(diag(tab_train))
```

```
## [1] 0.2311436
```

```
mean((y_train - gbm2_preds_train)^2)
```

```
## [1] 0.1597586
```

```
## Test performance
```

```
gbm2_preds_test <- predict(boost.ens2, newdata = MASQ[-train,], type = "response")
```

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

```
tab_test <- prop.table(table(true = y_test, predicted = gbm2_preds_test > .5))
```

```
tab_test
```

```
##      predicted
## true    FALSE    TRUE
##    0 0.4000000 0.1194444
##    1 0.1194444 0.3611111
```

```
1 - sum(diag(tab_test))
```

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

```
mean((y_test - gbm2_preds_test)^2)
```

```
## [1] 0.1671311
```

Only SEL slightly improved compared to the earlier boosted ensemble.

## Conclusion

## Warning: package 'knitr' was built under R version 4.3.2

	MCR_train	SEL_train	MCR_test	SEL_test
GAM	0.238	0.165	0.240	0.167
SVM	0.242	0.242	0.249	0.249
ctree	0.246	0.171	0.239	0.174
bagging	0.000	0.025	0.239	0.177
RF	0.000	0.024	0.239	0.172
GBM	0.221	0.150	0.238	0.168
GBM tuned	0.231	0.160	0.239	0.167

There are only minor performance differences between the methods. With a small number of predictors, and lack of interaction effects, this is perhaps not where prediction methods like SVMs, random forests or gradient boosting will shine. Results on other prediction problems may vary, and the current course simply aims to show you how to fit, interpret and apply these methods.

However, often marginal improvements may be expected for any of these sophisticated methods. Often, logistic and linear regression are hard to beat, as we also found in Fokkema et al. (2022).

In term of the information required for making a prediction and in terms of interpretability, GAMs and single decision trees perform well. All fitted tree ensembles require all variables for making a prediction (because all variables have non-zero importances). The ctree requires only the variables AD and GDD, and in some cases also GDD for prediction.

Note that random forests and bagging show the biggest difference in performance between train and test error. That is also why OOB predictions and OOB errors are very useful for these methods, as they do not present with the same over-optimism as the training errors.

Fokkema, M., Iliescu, D., Greiff, S., & Ziegler, M. (2022). Machine Learning and Prediction in Psychological Assessment: Some Promises and Pitfalls. *European Journal of Psychological Assessment*, 38(3), 165-175.<https://doi.org/10.1027/1015-5759/a000714>