SLP answers exercises session 5

Exercise: Splines

Load data:

```
library("foreign")
NESDA <- read.spss("nesda.sav", use.value.labels = FALSE, to.data.frame = TRUE)
summary(NESDA)
```

```
##      pident            Sexe            Age             aedu
##  Min.   :100013   Min.   :1.000   Min.   :18.00   Min.   : 5.00
##  1st Qu.:110833   1st Qu.:1.000   1st Qu.:30.00   1st Qu.:10.00
##  Median :210300   Median :2.000   Median :43.00   Median :12.00
##  Mean   :207229   Mean   :1.685   Mean   :41.59   Mean   :12.25
##  3rd Qu.:310132   3rd Qu.:2.000   3rd Qu.:53.00   3rd Qu.:15.00
##  Max.   :330392   Max.   :2.000   Max.   :64.00   Max.   :18.00
##     aframe02         neurot          extrave          openes
##  Min.   :1.000   Min.   :12.00   Min.   :18.00   Min.   :12.00
##  1st Qu.:1.000   1st Qu.:29.00   1st Qu.:32.00   1st Qu.:27.00
##  Median :1.000   Median :37.00   Median :37.00   Median :31.00
##  Mean   :1.652   Mean   :35.97   Mean   :37.22   Mean   :31.23
##  3rd Qu.:2.000   3rd Qu.:43.00   3rd Qu.:43.00   3rd Qu.:35.00
##  Max.   :3.000   Max.   :57.00   Max.   :54.00   Max.   :47.00
##     agreeab         aconscie          dyst              mdd
##  Min.   :24.00   Min.   :17.00   Min.   :0.00000   Min.   :0.00
##  1st Qu.:41.00   1st Qu.:34.00   1st Qu.:0.00000   1st Qu.:0.00
##  Median :44.00   Median :38.00   Median :0.00000   Median :0.00
##  Mean   :44.01   Mean   :37.53   Mean   :0.09333   Mean   :0.34
##  3rd Qu.:48.00   3rd Qu.:42.00   3rd Qu.:0.00000   3rd Qu.:1.00
##  Max.   :58.00   Max.   :55.00   Max.   :1.00000   Max.   :1.00
##       gad              sp              pd
##  Min.   :0.000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.000   Median :0.0000   Median :0.0000
##  Mean   :0.135   Mean   :0.2233   Mean   :0.2933
##  3rd Qu.:0.000   3rd Qu.:0.0000   3rd Qu.:1.0000
##  Max.   :1.000   Max.   :1.0000   Max.   :1.0000
```

```
NESDA$mdd <- factor(NESDA$mdd)
train <- NESDA[1:400,]
test <- NESDA[401:600,]
```

```
library("mgcv")
```

```
## Loading required package: nlme
```

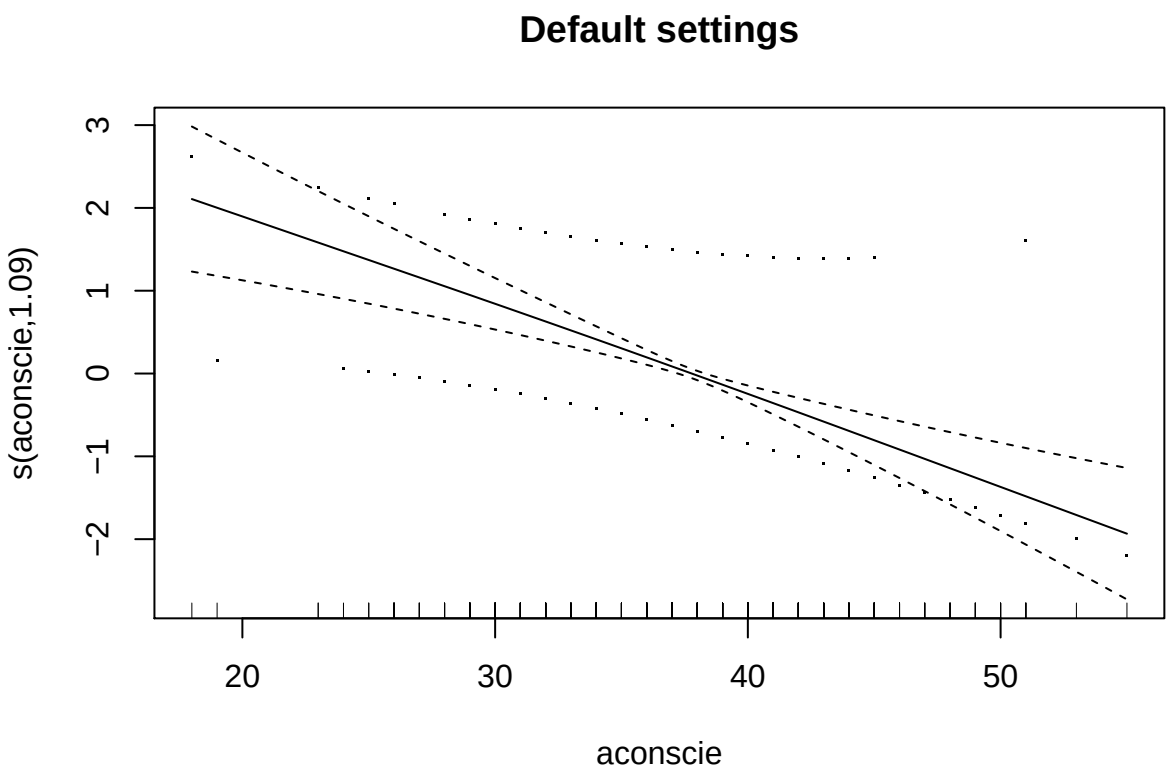
```
## This is mgcv 1.8-31. For overview type 'help("mgcv-package")'.
```

```
gam1 <- gam(mdd ~ s(aconscie), data = train,  family = "binomial",
            method = "REML")
```

```r
summary(gam1)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## mdd ~ s(aconscie)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.7745     0.1133  -6.836 8.15e-12 ***
## ---
## 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(aconscie) 1.085  1.166  29.03 1.54e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0775   Deviance explained = 6.64%
## -REML = 239.39  Scale est. = 1         n = 400
```

```r
plot(gam1, residuals = TRUE, main = "Default settings")
```



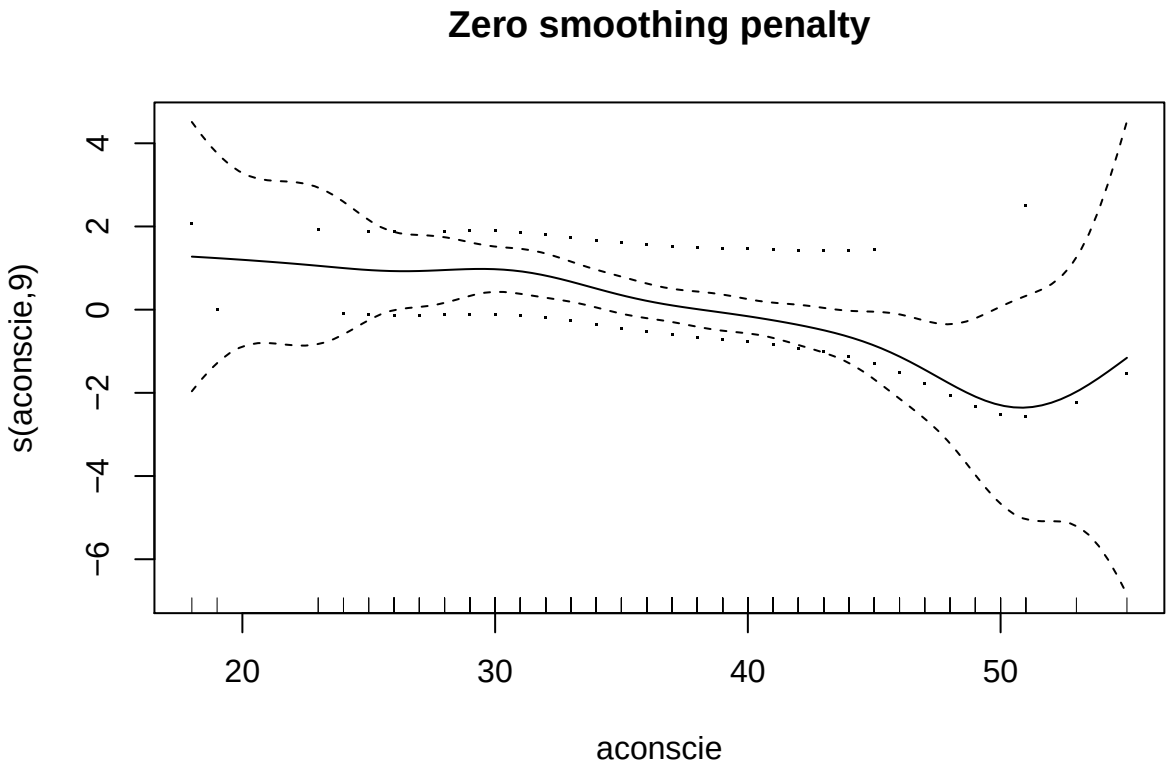
We get a curve that indicates a mostly linear effect. We also see that the edf is close to 1. The statistical test

indicates the effect of conscientiousness on mdd is significant.

```
gam2 <- gam(mdd ~ s(aconscie, sp = 0), data = train,  family = "binomial", method = "REML")
summary(gam2)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## mdd ~ s(aconscie, sp = 0)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.8047     0.1198  -6.718 1.85e-11 ***
## ---
## 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(aconscie)   9      9  29.34 0.000569 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0635   Deviance explained =  7.1%
## -REML =  375.2  Scale est. = 1         n = 400
```

```
plot(gam2, residuals = TRUE, main = "Zero smoothing penalty")
```

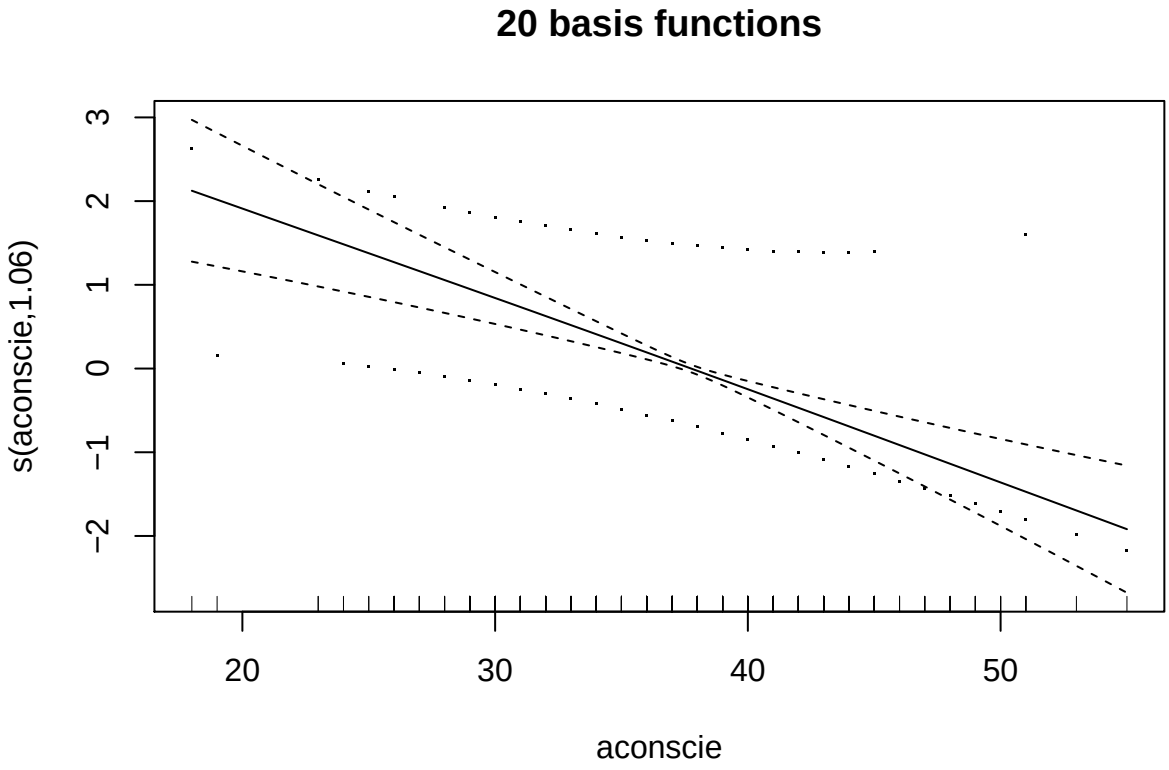


If we set the smoothing penalty to zero, we get a very wiggly line. By default, 9 basis functions are fitted, and we see from the edf that none of these functions is penalized out of the fitted function.

```
gam3 <- gam(mdd ~ s(aconscie, k = 20), data = train,  family = "binomial")
summary(gam3)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## mdd ~ s(aconscie, k = 20)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.7740     0.1132  -6.835 8.21e-12 ***
## ---
## 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(aconscie) 1.056   1.11  29.26 1.08e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0774   Deviance explained = 6.63%
## UBRE = 0.19454  Scale est. = 1         n = 400
```

```r
plot(gam3, residuals = TRUE, main = "20 basis functions")
```



Increasing the number of basis functions does not change our results. The smoothing penalty reduces the complexity of the fitted function back to something similar as we got with less basis functions.

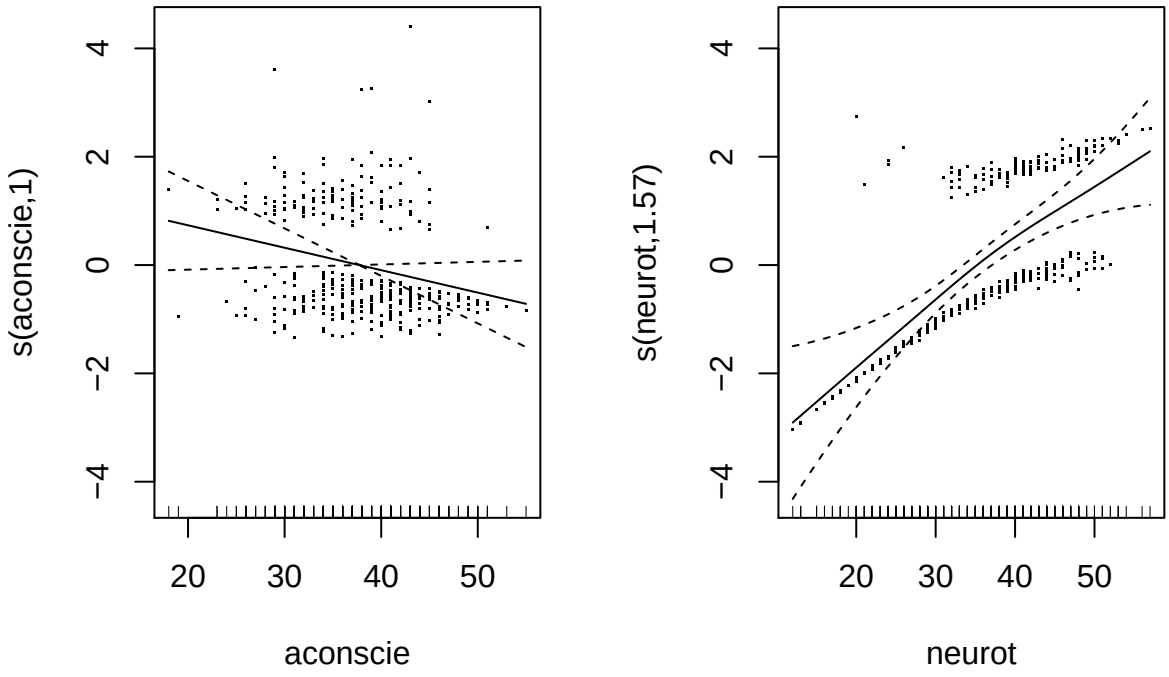
In sum: Knot location does not matter much when you specify more knots than you probably need, the smoothness penalty will penalize them down.

```r
gam4 <- gam(mdd ~ s(aconscie) +  s(neurot), data = train,  family = "binomial")
summary(gam4)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## mdd ~ s(aconscie) + s(neurot)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.9541     0.1354  -7.044 1.87e-12 ***
## ---
## 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(aconscie) 1.000  1.000  3.214    0.073 .
```

```
## s(neurot)   1.569  1.962 38.581 4.51e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.177   Deviance explained = 16.4%
## UBRE = 0.078594  Scale est. = 1         n = 400
```

```
par(mfrow = c(1, 2))
plot(gam4, residuals = TRUE)
```



When we add neuroticism as a predictor, the effect of conscientiousnous is reduced to a linear effect, while neuroticism has a slightly non-linear effect.

```
gam.preds <- predict(gam4, newdata = test, type = "response")
head(gam.preds)
```

```
##       401       402       403       404       405       406
## 0.6172342 0.5874856 0.3559893 0.1404975 0.7028163 0.5034647
```

```
gam.tab <- table(test$mdd, gam.preds > .5)
gam.tab
```

```
##
##     FALSE TRUE
##   0   111   17
##   1    43   29
```

```
1 - sum(diag(prop.table(gam.tab)))
```

```
## [1] 0.3
```

Note: REML estimation

By default, the `gam()` function will use the default generalized cross-validation criterion for smoothness selection. For finite sample sizes, the (default) GCV citerion may overfit (undersmooth) and REML is often preferred.

REML and GCV try to do the same thing: make the smooths in your model just wiggly enough and no wigglier. GCV will select optimal smoothing parameters when the sample size is infinite.

There is some finite sample size at which this asymptotic result kicks in, but we rarely have data of that size. At smaller sample sizes GCV can develop multiple minima, making optimisation difficult, yielding more variable estimates of the smoothing parameter.

GCV also tends to undersmooth, as it penalizes overfit weakly.

REML penalizes overfitting more, yielding more prounounced optima, leading to fewer optimisation issues and less variable estimates of the smoothing parameter.

See also:

Reiss, P.T. and Ogden, R.T. (2009) Smoothing parameter selection for a class of semiparametric linear models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 71, 505–523.

Wood, S.N. (2011) Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 73, 3–36.

Exercise: Support vectors

Linear kernel

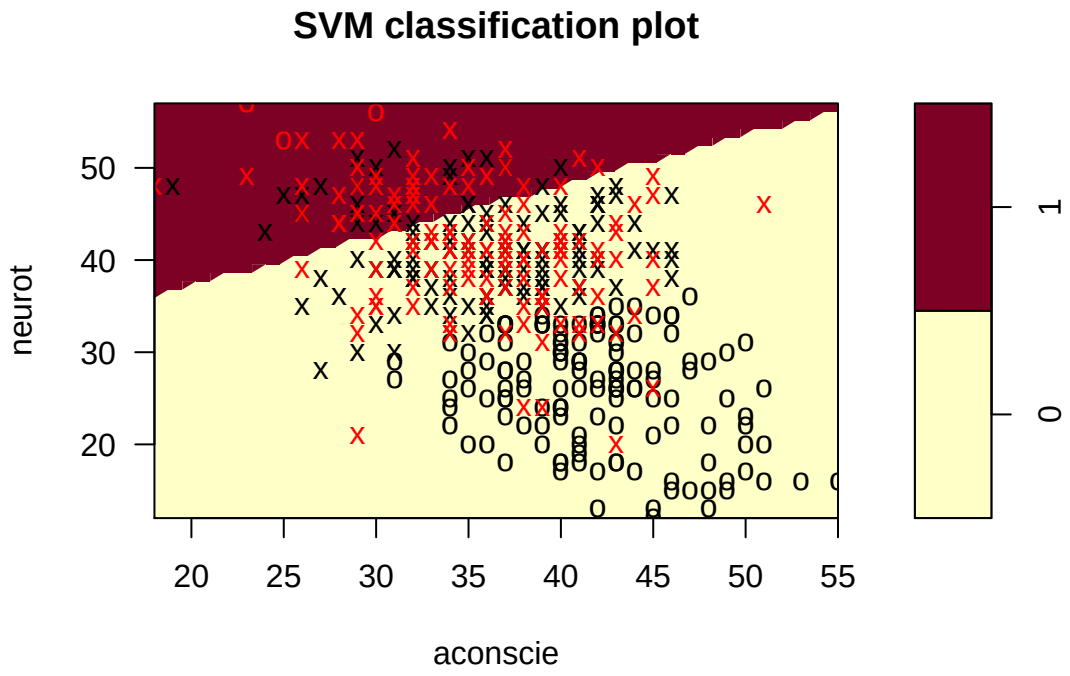
```
library("e1071")
tune.out <- tune(svm, mdd ~ neurot + aconscie, data = train,
                 kernel = "linear", scale = TRUE,
                 ranges = list(cost = c(.001, .01, .1, 1, 5, 10, 100)))
tune.out
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.2925
```

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

```
##   cost
## 4    1
```

```
svmfit <- svm(mdd ~ neurot + aconscie, data = train, kernel = "linear",
              cost = tune.out$best.parameters$cost, scale = TRUE)
plot(svmfit, data = train, formula = neurot ~ aconscie)
```



```
svmfit
```

```
##
## Call:
## svm(formula = mdd ~ neurot + aconscie, data = train, kernel = "linear",
##     cost = tune.out$best.parameters$cost, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  258
```

```
svm.preds <- predict(svmfit, newdata = test)
head(svm.preds)
```

```
## 401 402 403 404 405 406
##   1   1   0   0   1   0
## Levels: 0 1
```

```
svm.tab <- table(test$mdd, svm.preds)
svm.tab
```

```
##    svm.preds
##       0   1
##   0 116  12
##   1  51  21
```

```
1 - sum(diag(prop.table(svm.tab)))
```

```
## [1] 0.315
```

In the plot, `x` is a support vector, `o` are for other data points. The colors represent the classes the observations belong to (black for the first class (non depressed), red for the second class (depressed)). The decision boundary / hyperplane looks somewhat wiggly, but should be interpreted as a straight line.

The plot indicates that with higher values of neuroticism and lower values of conscientiousness, the probability of MDD increases.

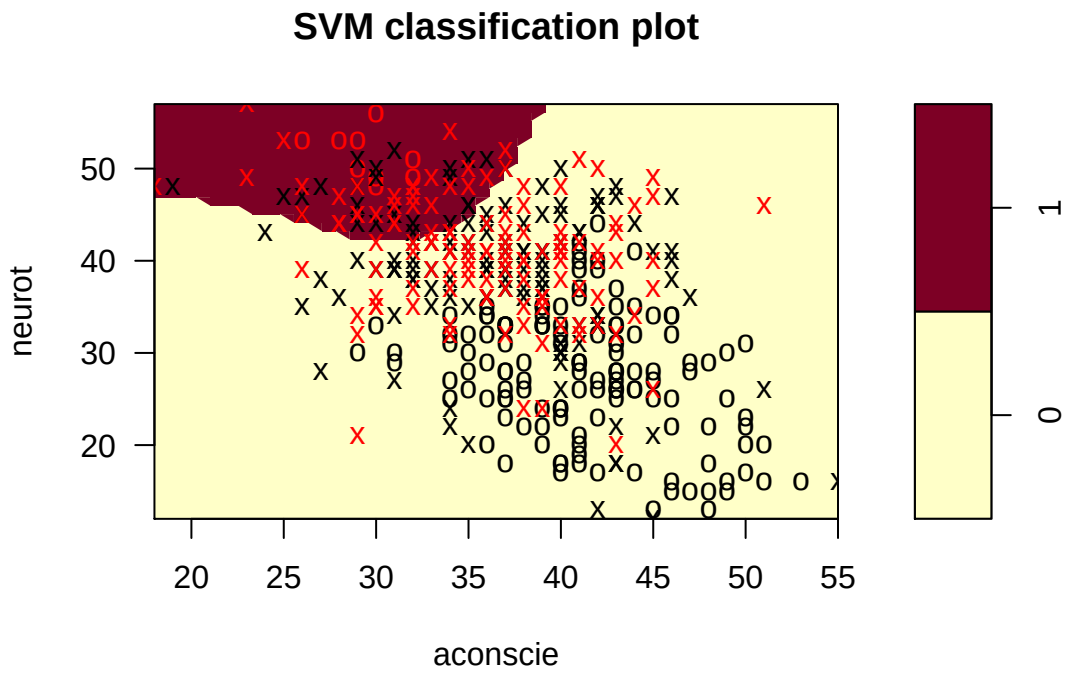
Radial basis kernel

```
tune.out <- tune(svm, mdd ~ neurot + aconscie, data = train,
                 kernel = "radial", ranges = list(
                   cost = c(.1, 1, 10, 100, 1000),
                   gamma = c(.5, 1, 2, 3, 4)))
tune.out
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
```

```
##     1   0.5
##
## - best performance: 0.3025
```

```
svmfit <- svm(mdd ~ neurot + aconscie, data = train,
              kernel = "radial", gamma = tune.out$best.parameters$gamma,
              cost = tune.out$best.parameters$cost)
plot(svmfit, train, formula = neurot ~ aconscie)
```



The radial basis kernel yields a similar pattern to what we obtained with the linear kernel, but we see obtained a non-linear decision boundary now.

The misclassification rate in the test data is:

```
svm.preds <- predict(svmfit, newdata = test)
svm.tab <- table(test$mdd, svm.preds)
svm.tab
```

```
##    svm.preds
##       0   1
##   0 118  10
##   1  52  20
```

```
1 - sum(diag(prop.table(svm.tab)))
```

```
## [1] 0.31
```

With the radial basis kernel, we achieved slightly lower misclassification rate on the test data, but still higher than the misclassification rate for the GAM with smoothing splines.

Exercise: Variable Selection Bias

a) We generate the variables:

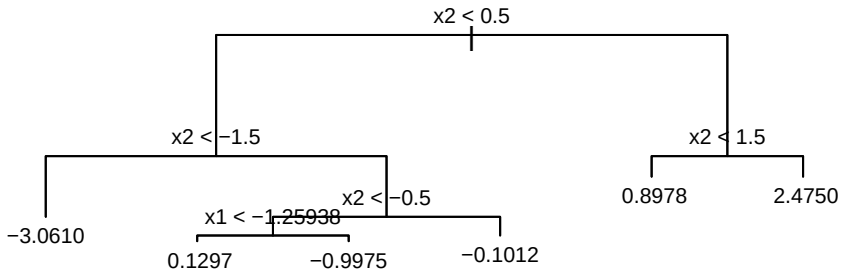
```
set.seed(42)
x1 <- rnorm(200)
x2 <- round(rnorm(200))
e <- rnorm(200)
```

b) We create the datasets

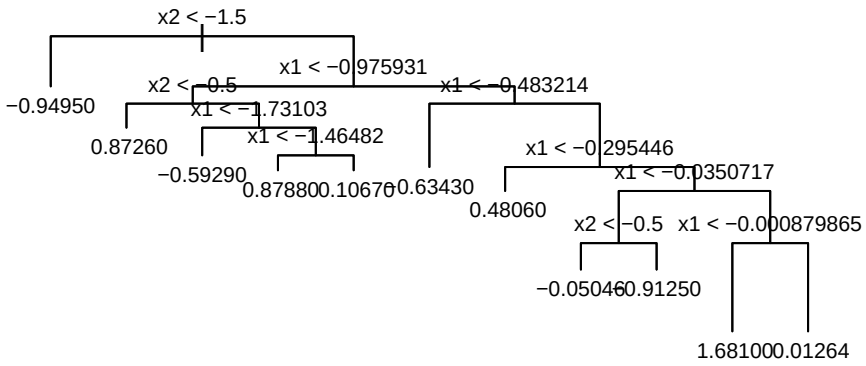
```
indep_data <- data.frame(x1, x2, y = e)
dep_data <- data.frame(x1, x2, y = x2 + e)
```

c) We fit an plot the trees:

```
library("tree")
cart_dep <- tree(y ~ x1 + x2,data = dep_data)
plot(cart_dep); text(cart_dep, cex = .7)
```



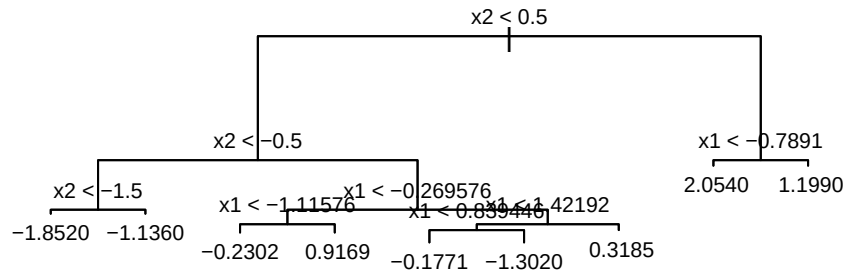

```
cart_indep <- tree(y ~ x1 + x2, data = indep_data)
plot(cart_indep); text(cart_indep, cex = .7)
```



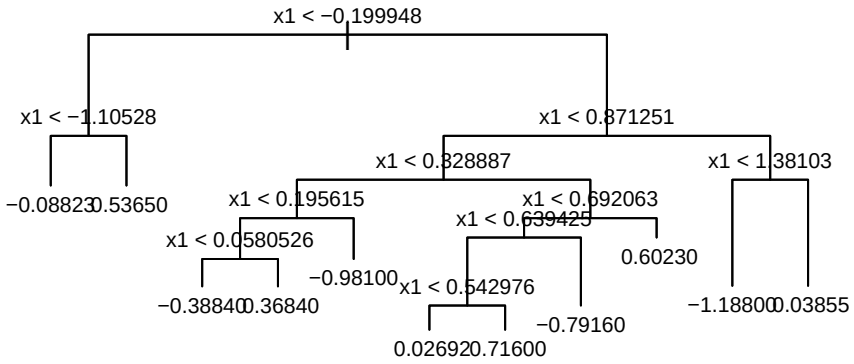
In the dependent data, x_1 is picked up erroneously once. In the independent data, we see a strong preference for splitting on x_1.

Note that in smaller datasets the variable selection bias would even become more pronounced:

```
set.seed(42)
x1 <- rnorm(100)
x2 <- round(rnorm(100))
e <- rnorm(100)
indep_data2 <- data.frame(x1, x2, y = e)
dep_data2 <- data.frame(x1, x2, y = x2 + e)
cart_dep2 <- tree(y ~ x1 + x2,data = dep_data2)
plot(cart_dep2); text(cart_dep2, cex = .7)
```

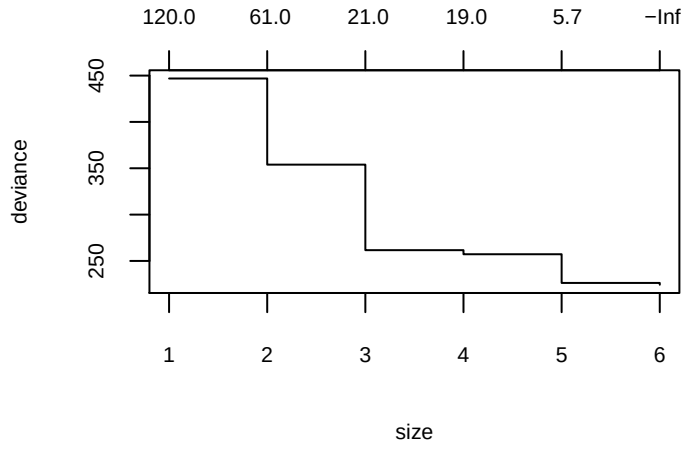


```
cart_indep2 <- tree(y ~ x1 + x2, data = indep_data2)
plot(cart_indep2); text(cart_indep2, cex = .7)
```



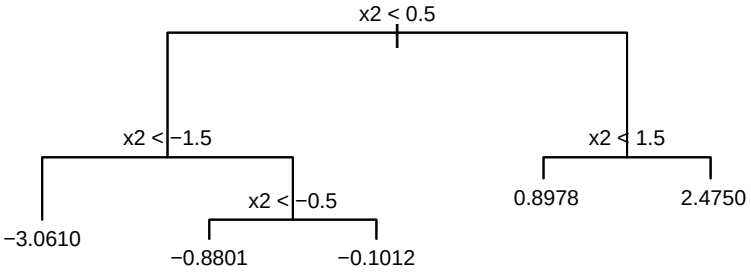
d) Apply CV to get the optimal tree sizes:

```
set.seed(3)
cv.dep <- cv.tree(cart_dep)
plot(cv.dep, cex.axis = .7, cex.lab = .7)
```

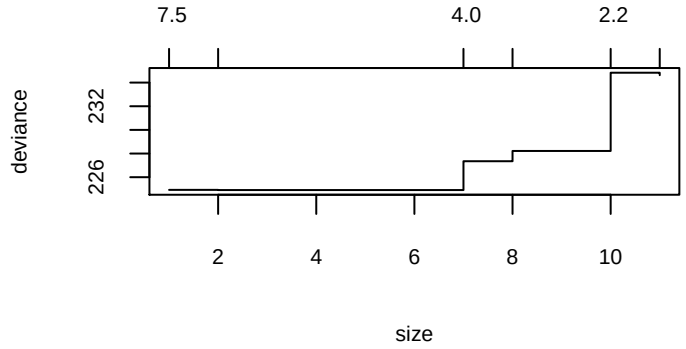


A tree with fisix nodes appears optimal for the dependent data. We prune the tree:

```
cart_dep2 <- prune.tree(cart_dep, best = 5)
plot(cart_dep2); text(cart_dep2, cex = .7)
```



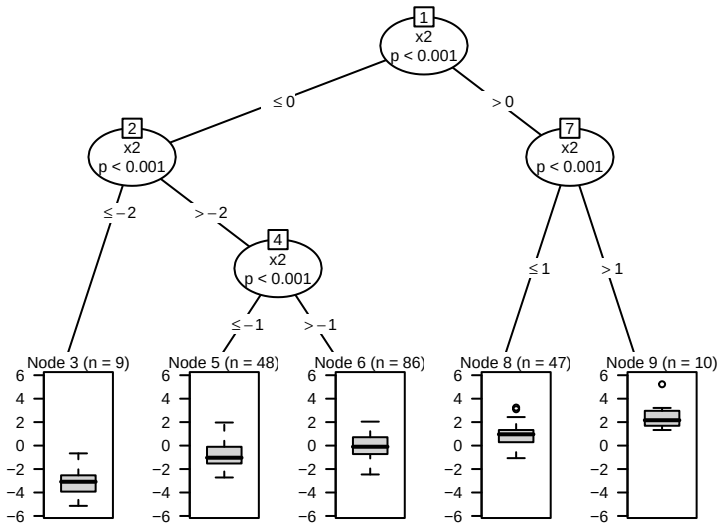

```
set.seed(3)
cv.indep <- cv.tree(cart_indep)
plot(cv.indep, cex.axis = .7, cex.lab = .7)
```



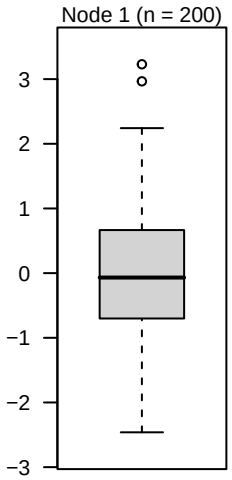
For the independent data, a tree with no splits would be optimal. So the pruning effectively mitigated overfitting and the variable selection bias in this dataset.

e) We fit conditional inference trees:

```
library("partykit")
ct_dep <- ctree(y ~ x1 + x2, data = dep_data)
plot(ct_dep, gp = gpar(cex = .5))
```




```
ct_indep <- ctree(y ~ x1 + x2, data = indep_data)
plot(ct_indep, gp = gpar(cex = .7))
```

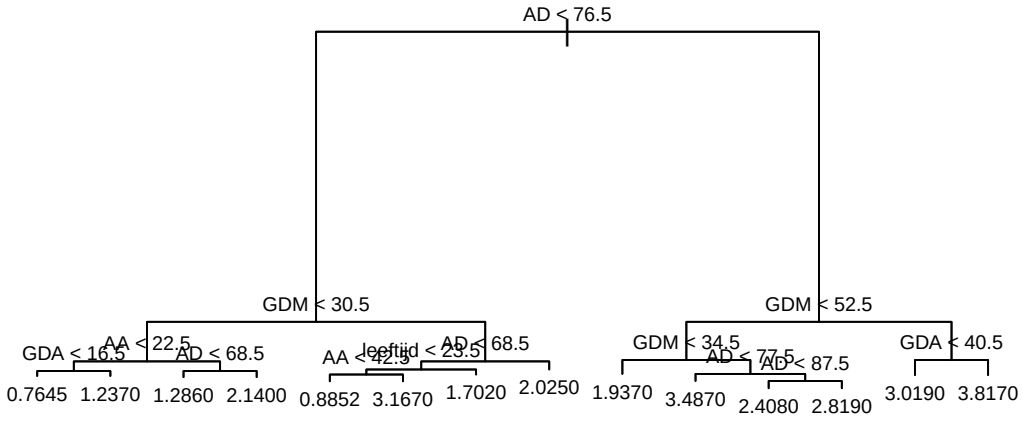


Exercise: Regression Trees on MASQ Data

```
MASQ <- read.table("MASQ.txt")
set.seed(1)
train <- sample(1:nrow(MASQ), size = nrow(MASQ)*.8)
```

a) We fit a CART tree on the data:

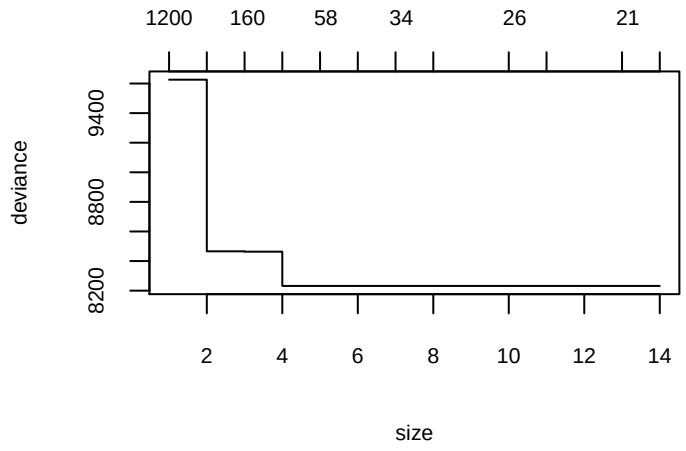
```
regtree <- tree(D_TOT ~ . - D_DEPDYS, data = MASQ[train,], mindev = .002)
plot(regtree); text(regtree, cex=.7)
```



Mostly, the subscale scores (except GDD; general distress depression) were selected for splitting. The splits indicate that higher values on any of the subscales are associatated with a higher number of diagnoses.

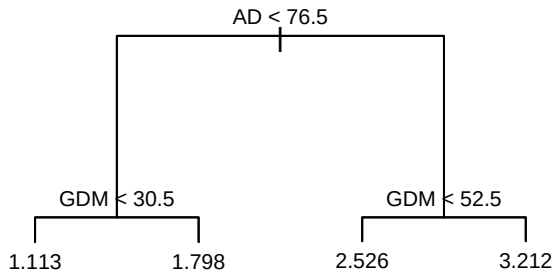
b) We perform cross validation to find the optimal value of the penalty parameter:

```
set.seed(2)
regtree.cv <- cv.tree(regtree)
plot(regtree.cv, cex.axis = .7, cex.lab = .7)
```



The upper x-axis represents the cost-complexity parameter (α in the lecture slides, the penalty parameter for tree size). The lower x-axis represents the size of the pruned tree fitted on the complete dataset, that would result with each of the values of the cost-complexity parameter. Next, we prune the tree:

```
regtree_p <- prune.tree(regtree, best = 4)
plot(regtree_p); text(regtree_p, cex=.7)
```



c) We compare theperformance of the (un)pruned trees:

```
mean((MASQ[-train, "D_TOT"] - predict(regtree, newdata = MASQ[-train,]))^2)
```

```
## [1] 2.728107
```

```
mean((MASQ[-train, "D_TOT"] - predict(regtree_p, newdata = MASQ[-train,]))^2)
```

```
## [1] 2.834377
```

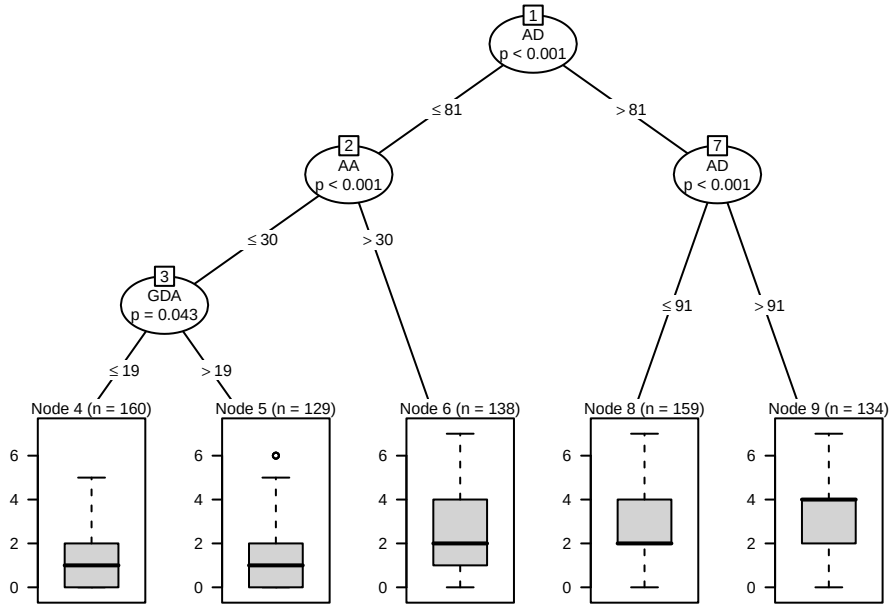
The pruned tree actually yielded a higher MSE on test data. But note that it is also much smaller

than the full tree, and we used only one train-test fold.

d) With a linear regression, we would assume a Gaussian error distribution, which allows for computing p value for each of the predictor variables. A regression tree does not make any distributional assumptions, so we also cannot violate them.

e) We fit a conditional inference tree:

```
ct_masq <- ctree(D_TOT ~ . - D_DEPDYS, data = MASQ[-train,])
plot(ct_masq, gp = gpar(cex = .5))
```




```
mean((MASQ[-train, "D_TOT"] - predict(ct_masq, newdata = MASQ[-train,]))^2)
```

```
## [1] 2.586634
```

The conditional inference tree provided lower prediction error on the test data than the CART tree.

Exercise: Bagging

We fit a bagged ensemble to the data using the `randomForest` function, and setting the `mtry` argument equal to the number of potential predictor variables:

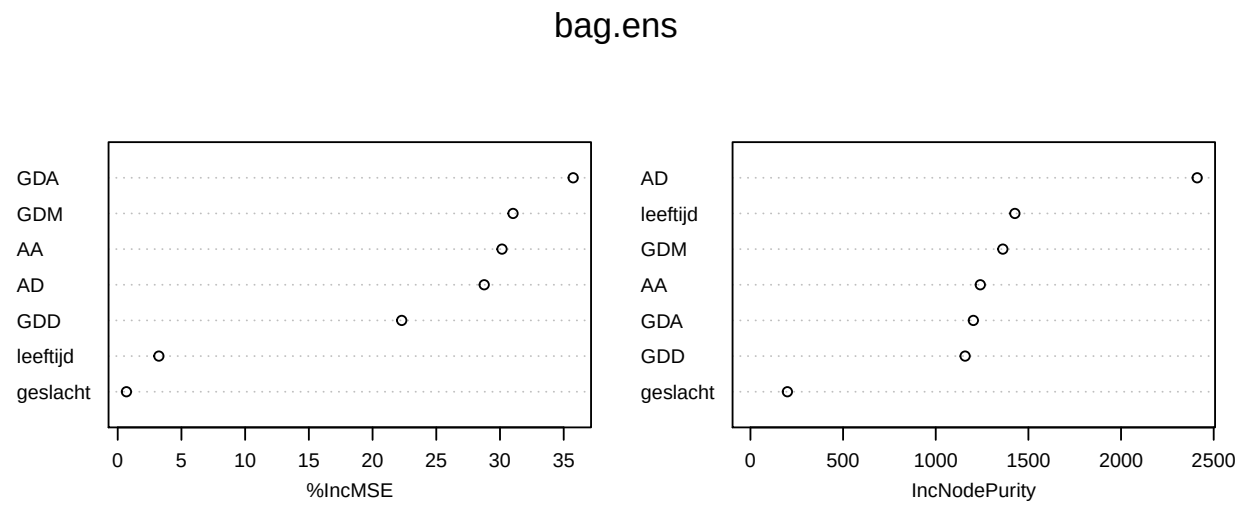
```
library("randomForest")
set.seed(1)
bag.ens <- randomForest(D_TOT ~ . - D_DEPDYS, data = MASQ[train,],
                        importance = TRUE, mtry = ncol(MASQ)-2L)
```

We inspect the variable importances of the fitted ensemble:

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

```
##             %IncMSE IncNodePurity
## AD       28.7603715     2410.9386
## AA       30.1559330     1240.4517
## GDD      22.2864949     1158.5113
## GDA      35.7423305     1202.9775
## GDM      31.0214453     1362.0140
## leeftijd  3.2290275     1427.0705
## geslacht  0.6840049      199.3395
```

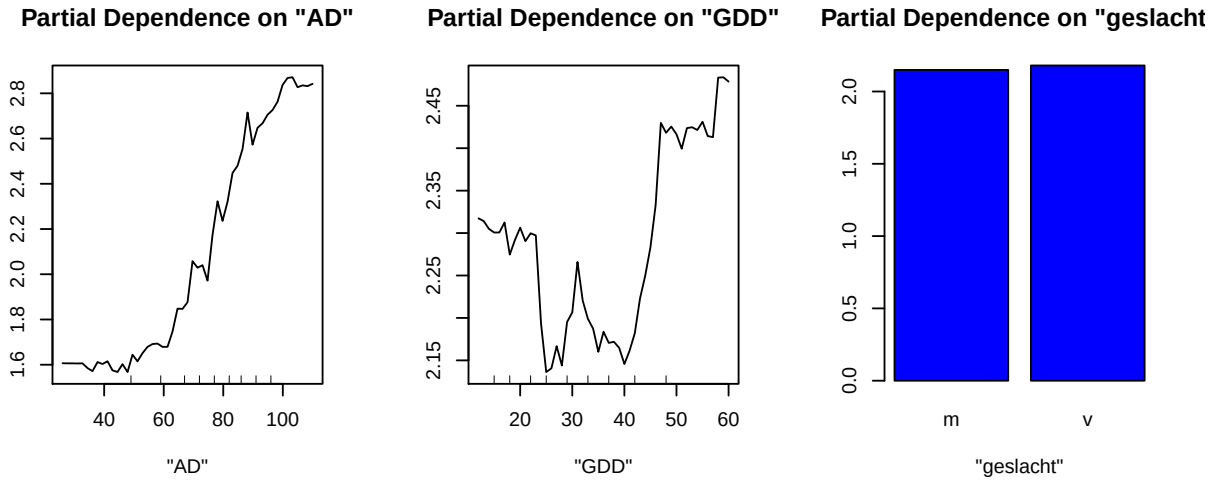
```
varImpPlot(bag.ens, cex = .7, cex.main = .7)
```



According to the reduction in MSE for the out-of-bag observations (left panel), when the values of each predictor variable are permuted, the GDA (general distress anxiety), GDM (general distress mixed), AA (anxious arousal) and AD (anhedonic depression) subscales are the most important predictors of the number of diagnoses.

According to the improvement in node purity (i.e., training error; right panel), AD (anhedonic depression) is the most important predictor.

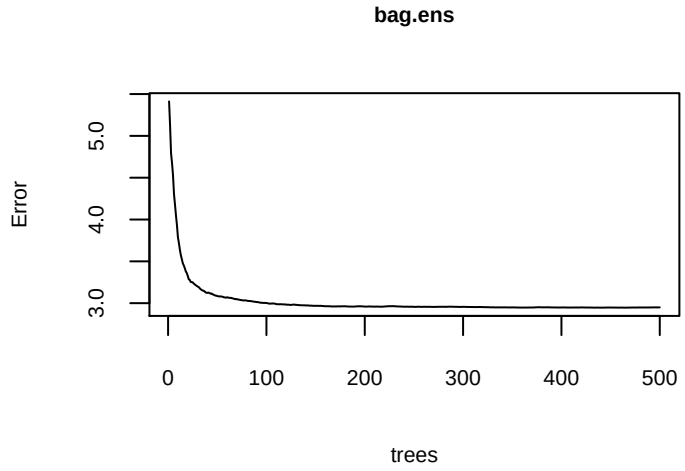
```
par(mfrow = c(1, 3))
partialPlot(bag.ens, x.var = "AD", pred.data = MASQ[train,])
partialPlot(bag.ens, x.var = "GDD", pred.data = MASQ[train,])
partialPlot(bag.ens, x.var = "geslacht", pred.data = MASQ[train,])
```



The partial dependence plots reveal the effects of a specified predictor variable, averaged over all possible values of the other predictor variables. We should be careful in interpreting them: if a predictor variable is involved in (higher) order interactions, the partial dependence plot will average over this interaction, while the actual effect of the variable will depend on the level of other variables.

In any case, the partial dependence plot suggest that the higher the AD scale score, the higher the number of diagnoses. The GDD subscale appears to have a non-linear, somewhat U-shaped, association with the number of diagnoses. Finally, men appear to have a very slightly lower number of diagnoses than women, on average.

```
plot(bag.ens, cex.lab = .7, cex.axis = .7, cex.main = .7)
```



For these data, the out-of-bag (OOB) error decreases fast with the first 100 trees. After 300 trees, the OOB error stabilizes and does not increase anymore.

```
mean((MASQ[-train, "D_TOT"] - predict(bag.ens, newdata = MASQ[-train,]))^2)
```

```
## [1] 2.797957
```

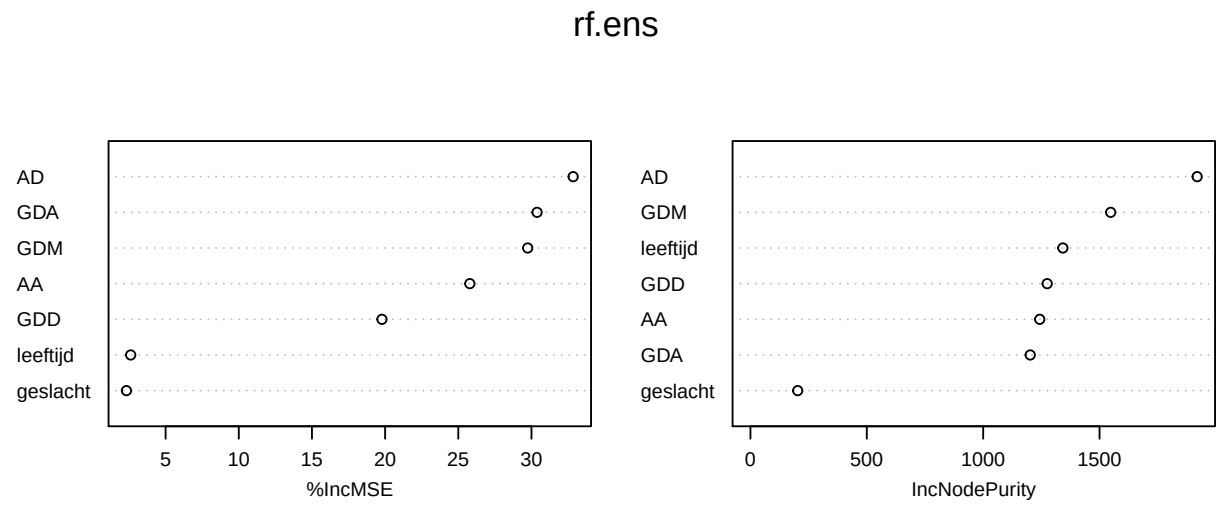
The bagged ensemble yields a higher test MSE than the conditional inference tree and the unpruned CART tree, but a lower test MSE than the pruned CART tree we fitted earlier.

Exercise: Random forest

```
set.seed(1)
rf.ens <- randomForest(D_TOT ~ . - D_DEPDYS, data = MASQ[train,],
                       importance = TRUE, mtry = sqrt(ncol(MASQ-2L)))
importance(rf.ens)
```

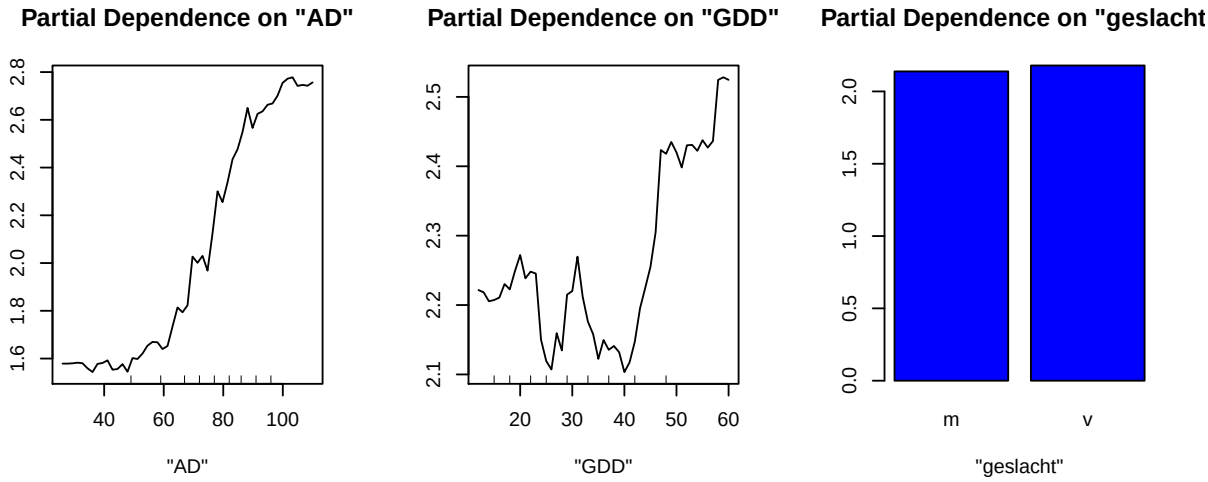
```
##            %IncMSE IncNodePurity
## AD       32.850142     1919.4988
## AA       25.789006     1242.9305
## GDD      19.783425     1274.9440
## GDA      30.382516     1202.1363
## GDM      29.743196     1547.8553
## leeftijd  2.613727     1342.4234
## geslacht  2.322494      203.1107
```

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



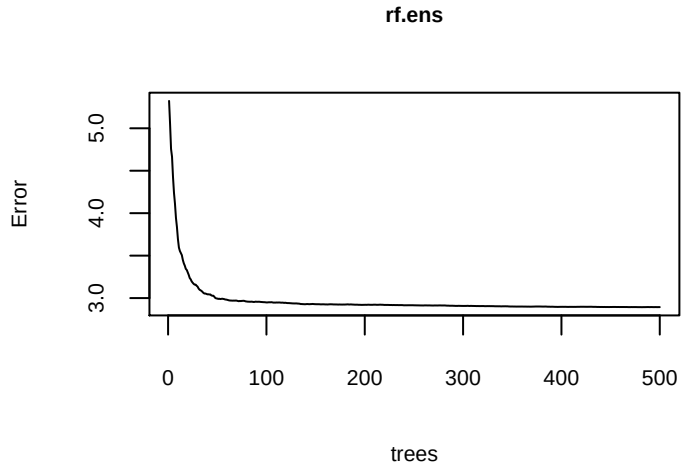
In the random forest, the AD scale is the most important predictor, according to both the OOB error (`%IncMSE`) and the training error (`IncNodePurity`). Similar variables appear most important in the bagged ensemble and the random forest, but the ordering is somewhat different.

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



For the random forest, we see very similar effects of the AD, GDD and geslacht variables, compared to the bagged ensemble.

```
plot(rf.ens, cex.lab = .7, cex.axis = .7, cex.main = .7)
```



The OOB error plotted against the number of trees shows a very similar pattern as with the bagged ensemble: the error stabilizes after about 300 trees.

```
mean((MASQ[-train, "D_TOT"] - predict(rf.ens, newdata = MASQ[-train,]))^2)
```

```
## [1] 2.761494
```

We observe slightly lower test error for the random forest than for the bagged ensemble. But still, both tree ensembles are outperformed by the conditional inference tree and the unpruned CART tree on the test dataset.

Exercise: Boosting

```
library("gbm")
set.seed(1)
boost.ens1 <- gbm(D_TOT ~ . - D_DEPDYS, data = MASQ[train,], n.trees = 1000,
                  shrinkage = .01, distribution = "gaussian")
set.seed(1)
boost.ens2 <- gbm(D_TOT ~ . - D_DEPDYS, data = MASQ[train,], n.trees = 1000,
                  shrinkage = .1, distribution = "gaussian")
```

The `n.trees` argument controls the number of generated trees, which defaults to 100.

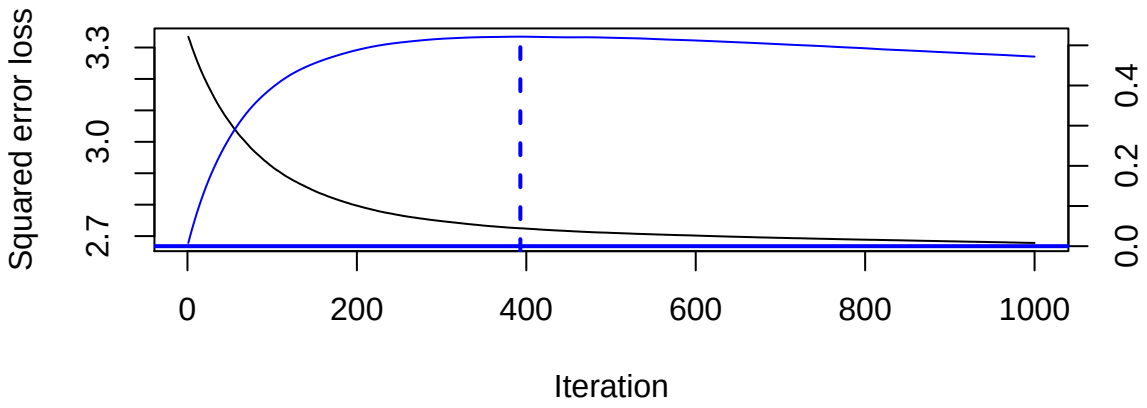
The `bag.fraction` argument controls the fraction of training set observations randomly generated to fit each tree in the ensemble.

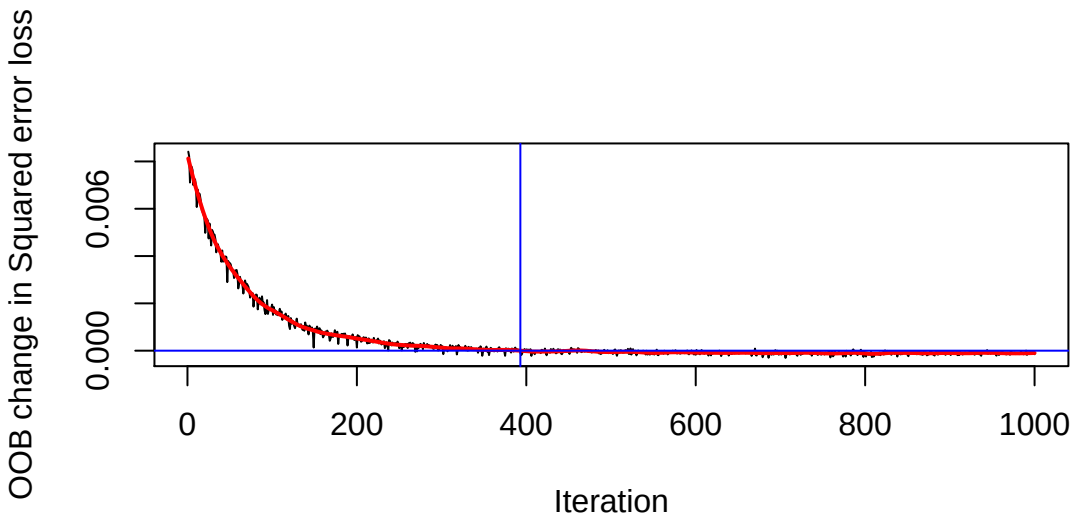
Tree depth is controlled by argument `interaction.depth`, which defaults to 1 (trees with a single split, i.e., 2 terminal nodes, i.e., main effects only).

The learning rate is controlled by the `shrinkage` argument, which defaults to 0.001.

```
opt_ntrees1 <- gbm.perf(boost.ens1, oobag.curve = TRUE)
```

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

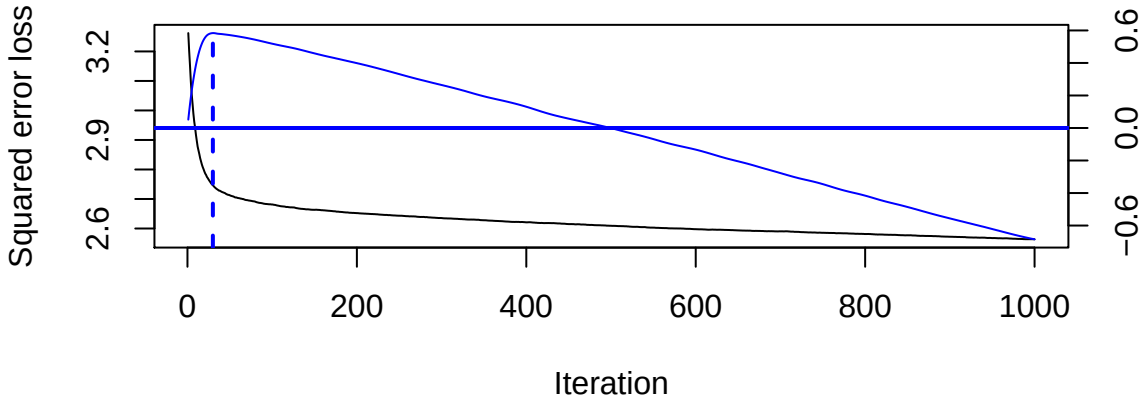


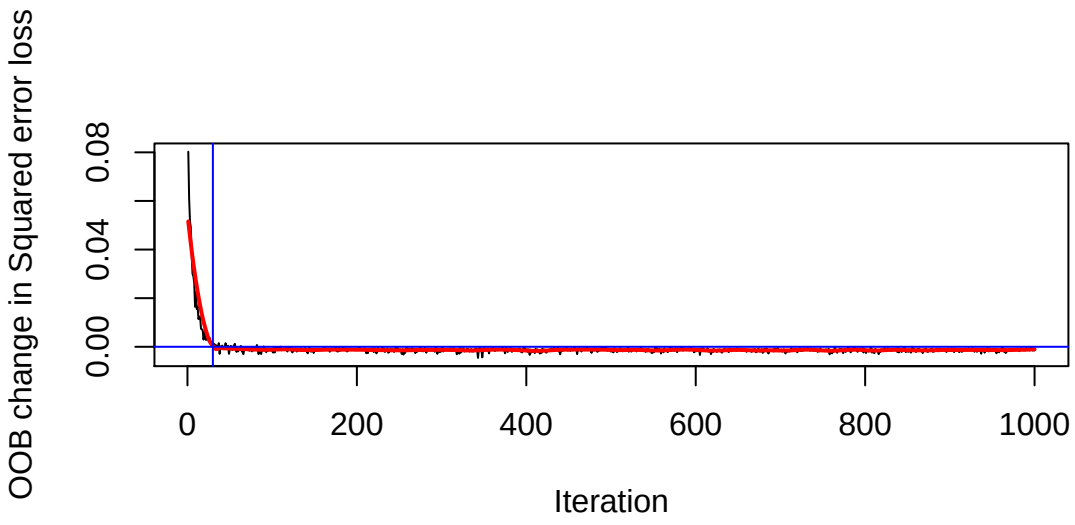


For the ensemble with a learning rate of .01, the optimal number of trees according to the OOB error is 393.

```
opt_ntrees2 <- gbm.perf(boost.ens2, oobag.curve = TRUE)
```

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





For the ensemble with a learning rate of .1, the optimal number of trees according to the OOB error is 30.

Thus, with a higher learning rate, a lower number of trees appears optimal.

We will now use the optimal number of trees according to the OOB error, and slightly higher values, to assess MSE on test data:

For the boosted ensemble with learning rate of .01:

```
mean((MASQ[-train, "D_TOT"] - predict(boost.ens1, newdata = MASQ[-train,],
                                      n.trees = as.numeric(opt_ntrees1)))^2)
```

```
## [1] 2.672199
```

```
mean((MASQ[-train, "D_TOT"] - predict(boost.ens1, newdata = MASQ[-train,],
                                      n.trees = 500))^2)
```

```
## [1] 2.668755
```

For the boosted ensemble with learning rate of .1:

```
mean((MASQ[-train, "D_TOT"] - predict(boost.ens2, newdata = MASQ[-train,],
                                      n.trees = as.numeric(opt_ntrees2)))^2)
```

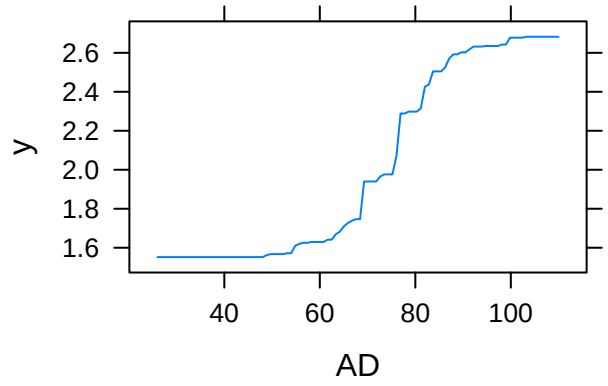
```
## [1] 2.701661
```

```
mean((MASQ[-train, "D_TOT"] - predict(boost.ens2, newdata = MASQ[-train,],
                                      n.trees = 50))^2)
```

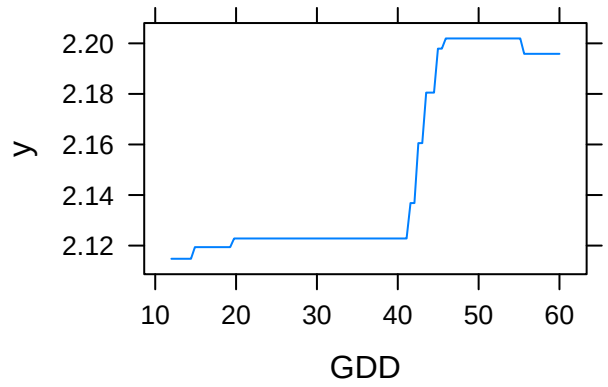
```
## [1] 2.67305
```

As the OOB error may underestimate the number of trees required for optimal predictive accuracy, I also tried somewhat higher values of `n.trees`, which indeed yield slightly better predictive accuracy. Depending on the value of the learning rate and the number of trees used for prediction, the boosted tree ensembles yielded lower test MSE than the (un)pruned CART trees, the bagged ensemble and random forest, but higher test MSE than the conditional inference tree.

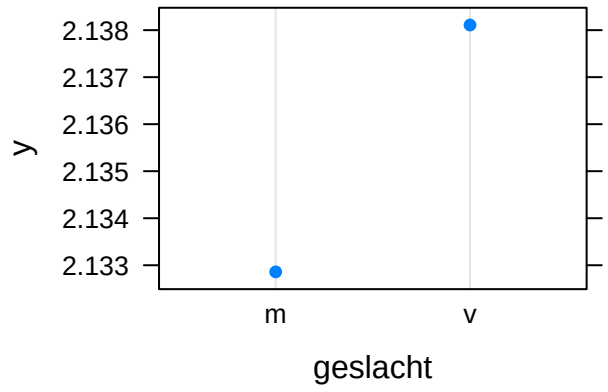
```
plot(boost.ens1, i.var = "AD", n.trees = 500)
```




```
plot(boost.ens1, i.var = "GDD", n.trees = 500)
```

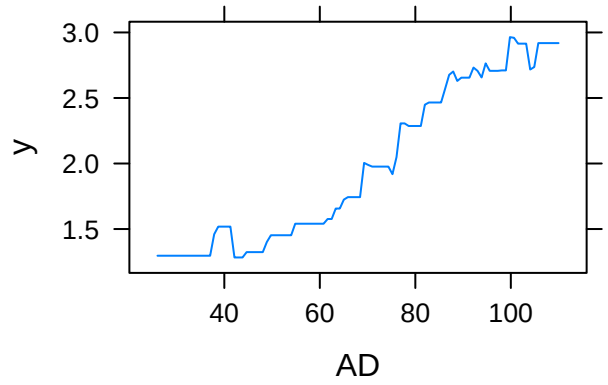



```
plot(boost.ens1, i.var = "geslacht", n.trees = 500)
```

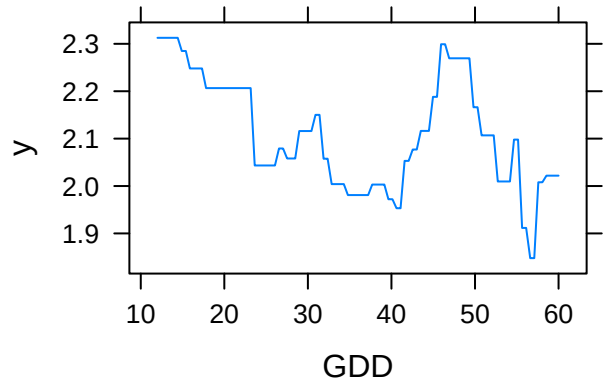


The partial dependence plot suggest that the higher the AD and GDD scale scores, the higher the number of diagnoses. Men appear to have a slightly lower number of diagnoses compared to women, on average.

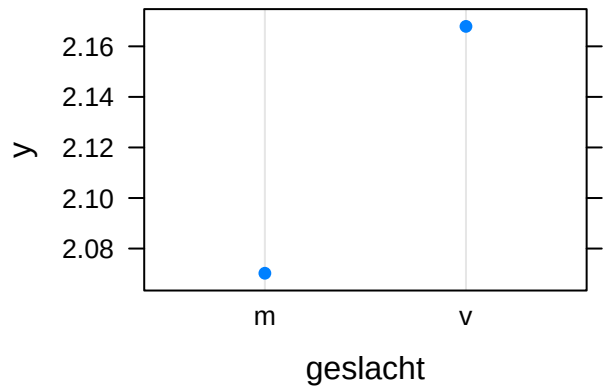
```
plot(boost.ens2, i.var = "AD", n.trees = 500)
```




```
plot(boost.ens2, i.var = "GDD", n.trees = 500)
```



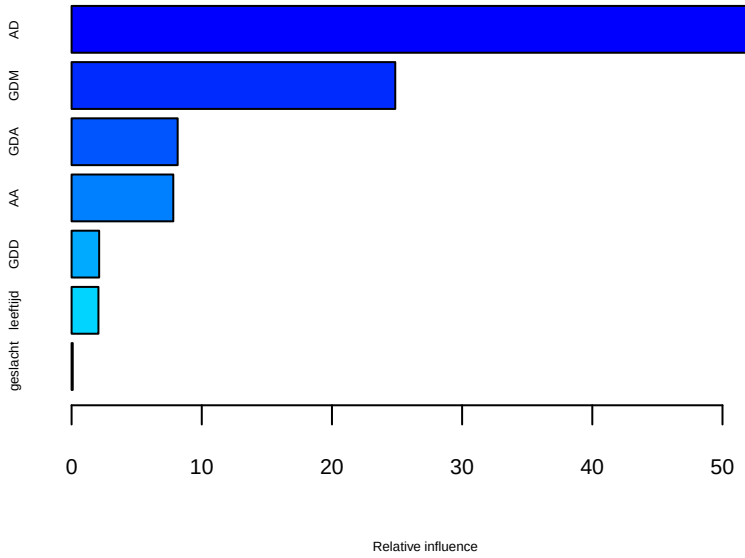

```
plot(boost.ens2, i.var = "geslacht", n.trees = 500)
```



The partial dependence plots suggest that the higher the AD score, the higher the number of diagnoses. The

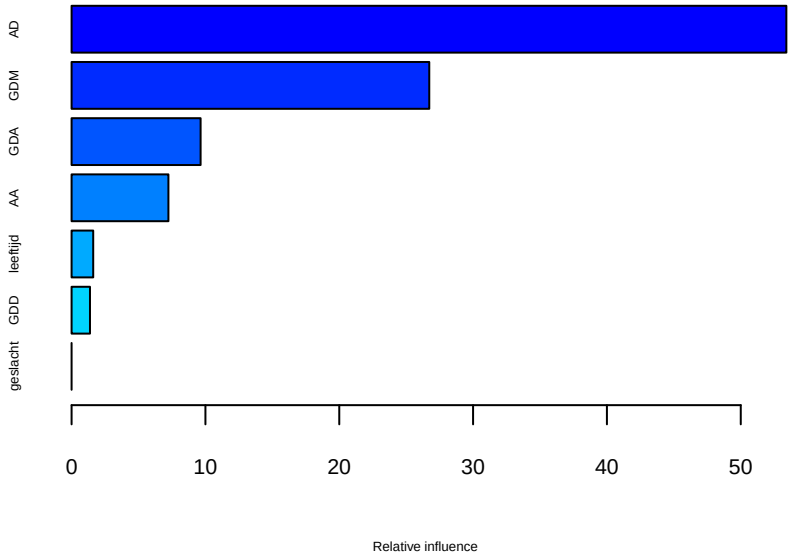
effect of the GDD variable seems less clear. Men appear to have a slightly lower number of diagnoses than women.

```
summary(boost.ens1, n.trees = 500, cex.lab = .4, cex.axis = .7, cex.sub = .4, cex = .4)
```




```
##                var     rel.inf
## AD              AD 54.90582667
## GDM            GDM 24.86464870
## GDA            GDA  8.14891079
## AA              AA  7.81249435
## GDD            GDD  2.11894211
## leeftijd leeftijd  2.06204650
## geslacht geslacht  0.08713089
```

```
summary(boost.ens2, n.trees = 50, cex.lab = .4, cex.axis = .7, cex.sub = .4, cex = .4)
```




```
##                var   rel.inf
## AD              AD 53.407776
## GDM            GDM 26.725987
## GDA            GDA  9.639157
## AA              AA  7.235854
## leeftijd leeftijd  1.615991
## GDD            GDD  1.375236
## geslacht geslacht  0.000000
```

In both boosted ensembles, the AD subscale appears most important in predicting the number of diagnoses.

Exercise: Tuning boosting parameters

Using package **caret**, we optimize the tuning parameters for a boosted ensemble:

```
library("caret")
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

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

```
##    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
## 7      0.100    1000                 1             10
## 8      0.010    1000                 1             10
## 9      0.001    1000                 1             10
## 10     0.100      10                 2             10
## 11     0.010      10                 2             10
## 12     0.001      10                 2             10
## 13     0.100     100                 2             10
## 14     0.010     100                 2             10
## 15     0.001     100                 2             10
## 16     0.100    1000                 2             10
## 17     0.010    1000                 2             10
## 18     0.001    1000                 2             10
## 19     0.100      10                 3             10
## 20     0.010      10                 3             10
## 21     0.001      10                 3             10
## 22     0.100     100                 3             10
## 23     0.010     100                 3             10
## 24     0.001     100                 3             10
## 25     0.100    1000                 3             10
## 26     0.010    1000                 3             10
## 27     0.001    1000                 3             10
## 28     0.100      10                 4             10
## 29     0.010      10                 4             10
## 30     0.001      10                 4             10
## 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 see the tuning parameter grid that predictive accuracy will be assessed over. As the `train()` function from package **caret** employs subsampling 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 36 (for each row of the tuning grid) times 25 (bootstrap replications) boosted models, so it will take some time to run:

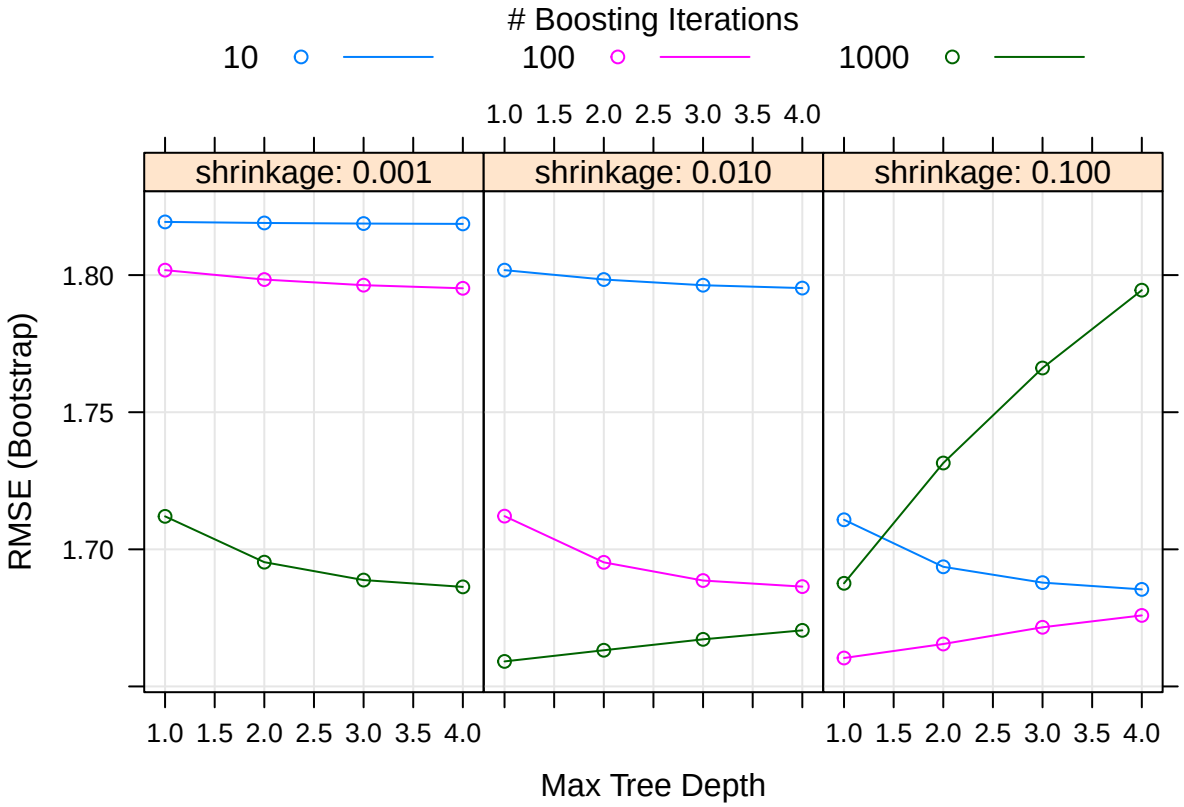
```
set.seed(3)
gbmFit <- train(D_TOT ~ . - D_DEPDYS, data = MASQ,
                method = "gbm", tuneGrid = grid,
                verbose = FALSE)
gbmFit
```

```
## Stochastic Gradient Boosting
##
## 3597 samples
##    8 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 3597, 3597, 3597, 3597, 3597, 3597, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.trees  RMSE      Rsquared   MAE
##   0.001      1                    10     1.819388  0.1237631  1.480786
##   0.001      1                   100     1.801805  0.1298996  1.471109
##   0.001      1                  1000     1.712028  0.1532300  1.421977
##   0.001      2                    10     1.819015  0.1443313  1.480537
##   0.001      2                   100     1.798380  0.1488688  1.468724
##   0.001      2                  1000     1.695321  0.1644198  1.406764
##   0.001      3                    10     1.818793  0.1533858  1.480384
##   0.001      3                   100     1.796329  0.1589185  1.467154
##   0.001      3                  1000     1.688793  0.1680476  1.399437
##   0.001      4                    10     1.818667  0.1597274  1.480273
##   0.001      4                   100     1.795190  0.1630719  1.466299
##   0.001      4                  1000     1.686308  0.1690735  1.396469
##   0.010      1                    10     1.801813  0.1269311  1.471041
##   0.010      1                   100     1.712098  0.1531036  1.422000
##   0.010      1                  1000     1.659126  0.1715115  1.348706
##   0.010      2                    10     1.798389  0.1475731  1.468591
##   0.010      2                   100     1.695246  0.1641575  1.406722
##   0.010      2                  1000     1.663203  0.1678094  1.350245
##   0.010      3                    10     1.796306  0.1563286  1.467081
##   0.010      3                   100     1.688618  0.1677319  1.399498
##   0.010      3                  1000     1.667177  0.1644662  1.351832
##   0.010      4                    10     1.795261  0.1606053  1.466394
##   0.010      4                   100     1.686398  0.1685419  1.396733
##   0.010      4                  1000     1.670454  0.1618898  1.353054
##   0.100      1                    10     1.710781  0.1504760  1.421132
##   0.100      1                   100     1.660370  0.1703624  1.349878
##   0.100      1                  1000     1.687611  0.1491066  1.363620
##   0.100      2                    10     1.693624  0.1622871  1.404900
```

```
##   0.100      2                  100     1.665473  0.1659455  1.351267
##   0.100      2                 1000     1.731470  0.1248946  1.390103
##   0.100      3                   10     1.687858  0.1646590  1.397783
##   0.100      3                  100     1.671569  0.1608475  1.354310
##   0.100      3                 1000     1.766119  0.1111083  1.413245
##   0.100      4                   10     1.685395  0.1658645  1.394813
##   0.100      4                  100     1.675898  0.1578167  1.356510
##   0.100      4                 1000     1.794492  0.1005477  1.432759
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 1000, interaction.depth =
##  1, shrinkage = 0.01 and n.minobsinnode = 10.
```

Note that these predictive accuracies are estimated through cross-validation, with a default value of k. We can also obtain a plot of the results:

```
plot(gbmFit)
```



Note that the y axis represents the square root of the MSE. Note that with higher values of the shrinkage parameter, lower values for the number of trees start to perform better.

The result indicates that an interaction depth of 1 would be optimal, indicating that there may be no interactions present in the data.

We can obtain the optimal parameter value set as follows:

```
gbmFit$bestTune
```

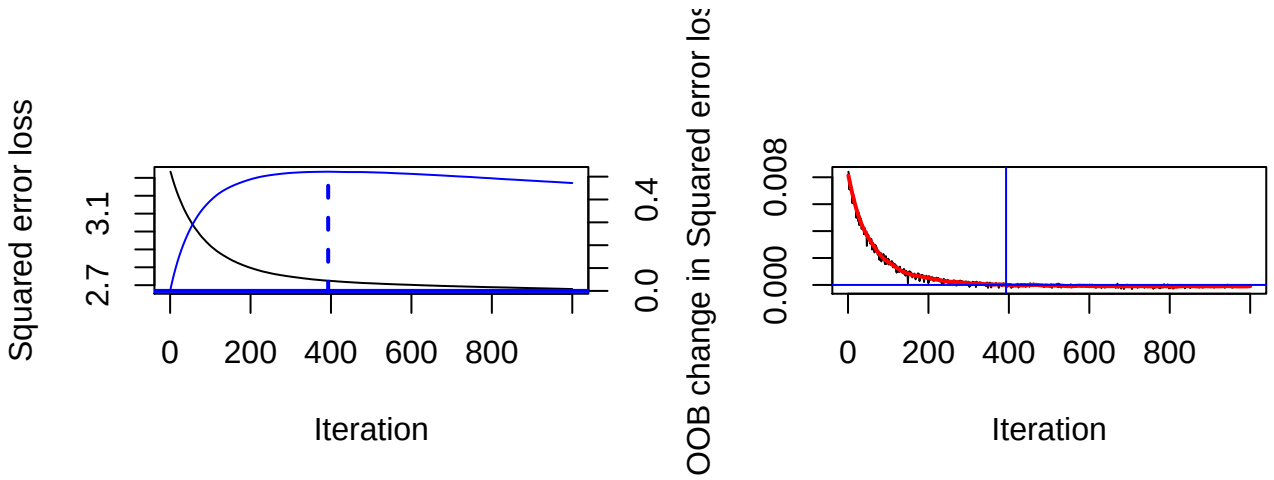
```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 15    1000                 1      0.01             10
```

These are not the same as the values we used for our two initially fitted ensembles. Thus, we refit the ensemble using the parameter values that can be expected to optimize predictive accuracy:

```
set.seed(1)
boost.ens3 <- gbm(D_TOT ~ . - D_DEPDYS, data = MASQ[train,], n.trees = 1000,
                  shrinkage = .01, interaction.depth = 1,
                  distribution = "gaussian")
```

```
gbm.perf(boost.ens3, oobag.curve = TRUE)
```

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




```
## [1] 393
## 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: 0.0001118
```

The OOB error estimate suggests optimal predictive accuracy for less than 1,000 trees. Let's compare the optimal value according to `caret` and according to `gbm`'s OOB error estimate:

```
mean((MASQ[-train, "D_TOT"] - predict(boost.ens3, newdata = MASQ[-train,], n.trees = 1000))^2)
```

```
## [1] 2.666956
```

```
mean((MASQ[-train, "D_TOT"] - predict(boost.ens3, newdata = MASQ[-train,], n.trees = 500))^2)
```

```
## [1] 2.668755
```

For the boosted ensemble, we get the best performance (lowest test MSE) using the parameter values obtained using bootstrapped cross validation.

For this data problem, boosting yielded the best predictive accuracy among the ensemble methods, but the ensemble methods did not outperform the single conditional inference tree. This may be due to the single train-test fold we used. Results for a different train-test on the same dataset may have yielded a different conclusion.