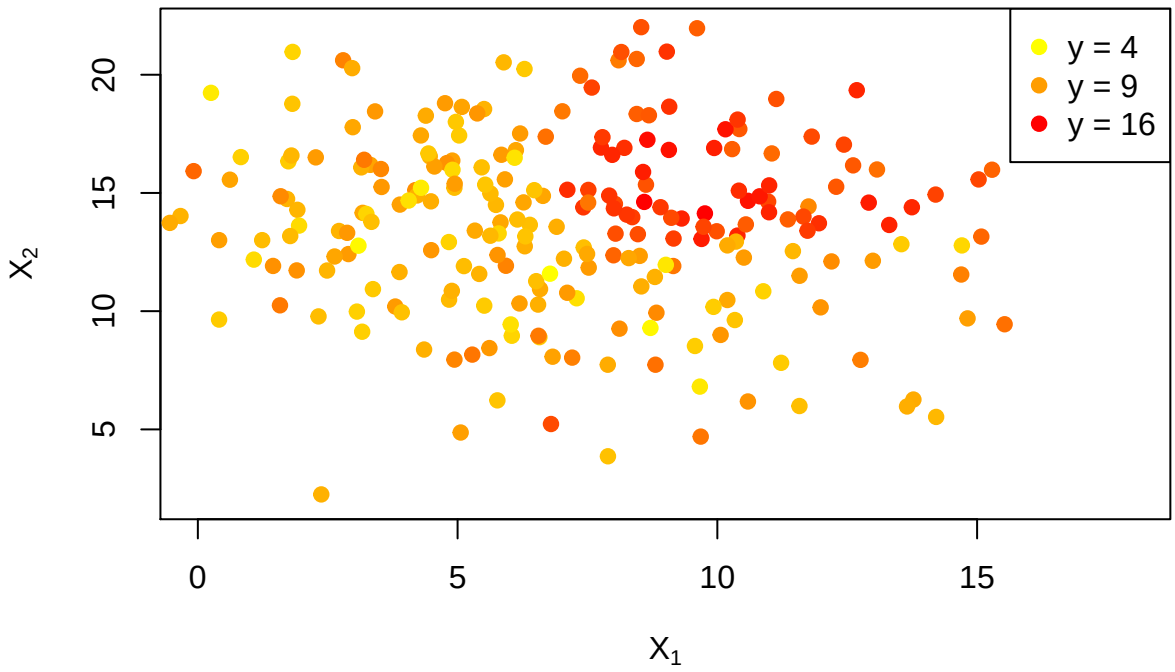
Code for recreating presentation examples

Examples involving X_1 and X_2

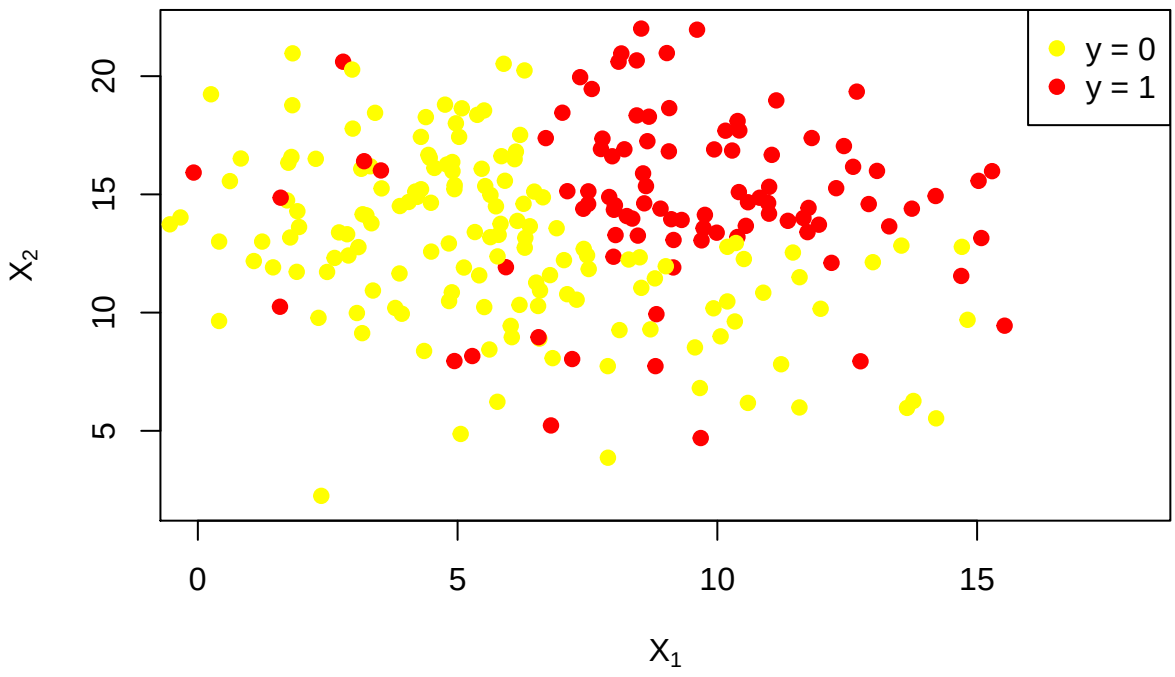
```
# generate data:
set.seed(12)
x1 <- rnorm(250, mean = 7, sd = 4)
x2 <- rnorm(250, mean = 14, sd = 4)
error <- rnorm(250, 0, 1.5)
y_cont <- 8 + 5*(x1 > 7 & x2 > 13) + error
y_bin <- as.factor(as.numeric(y_cont > mean(y_cont)))

# create plots:
plot(c(0, 18), c(2, 22), type = "n", xlab = expression(X[1]), ylab = expression(X[2]))
rbPal <- colorRampPalette(c('yellow','red'))
color <- rbPal(150)[as.numeric(cut(y_cont, breaks = 150))]
points(x1, x2, pch = 19, col = color)
cshort <- color[order(y_cont)][c(1, 125, 250)]
yshort <- round(y_cont[order(y_cont)][c(1, 125, 250)])
legend("topright", legend = paste("y =", yshort), col = cshort, pch = 19)
```

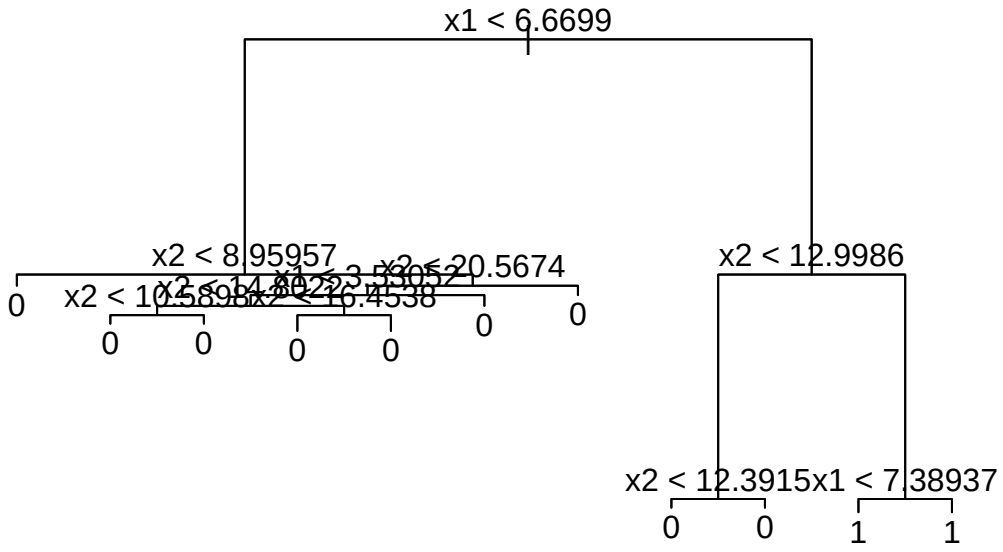



```
plot(c(0, 18), c(2, 22), type = "n", xlab = expression(X[1]), ylab = expression(X[2]))
color <- ifelse(y_bin == 0, "yellow", "red")
points(x1, x2, pch = 19, col = color)
```

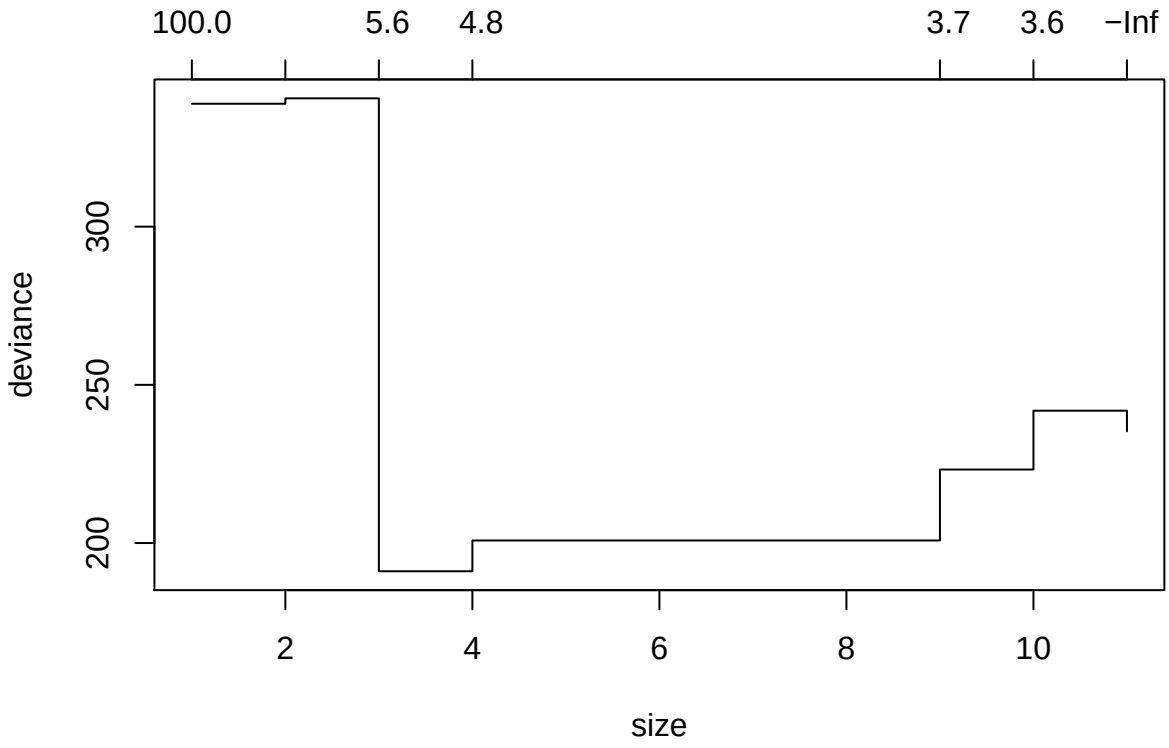
```
legend("topright", legend = paste("y =", 0:1), col = c("yellow", "red"), pch = 19)
```



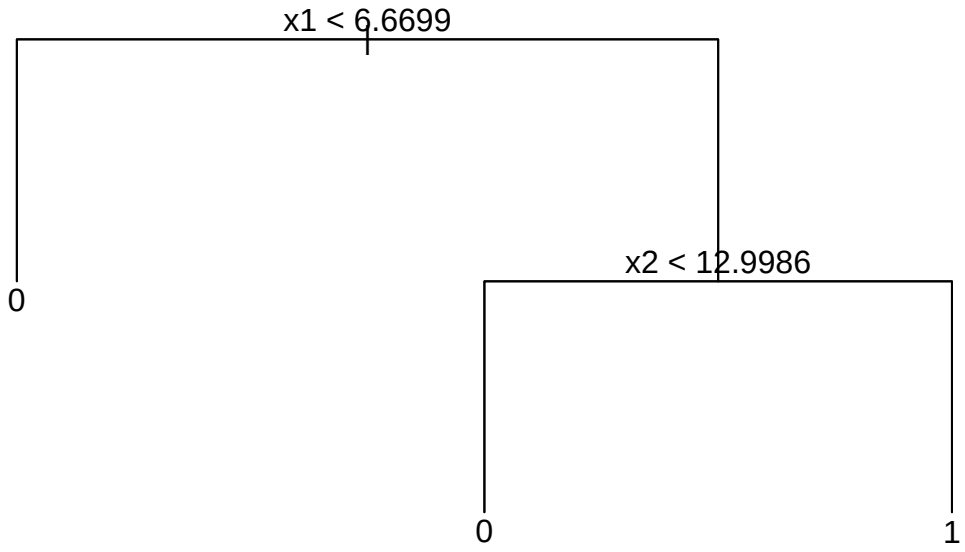

```
# fit and plot CART trees:
library("tree")
cart <- tree(y_bin ~ x1 + x2)
plot(cart)
text(cart)
```



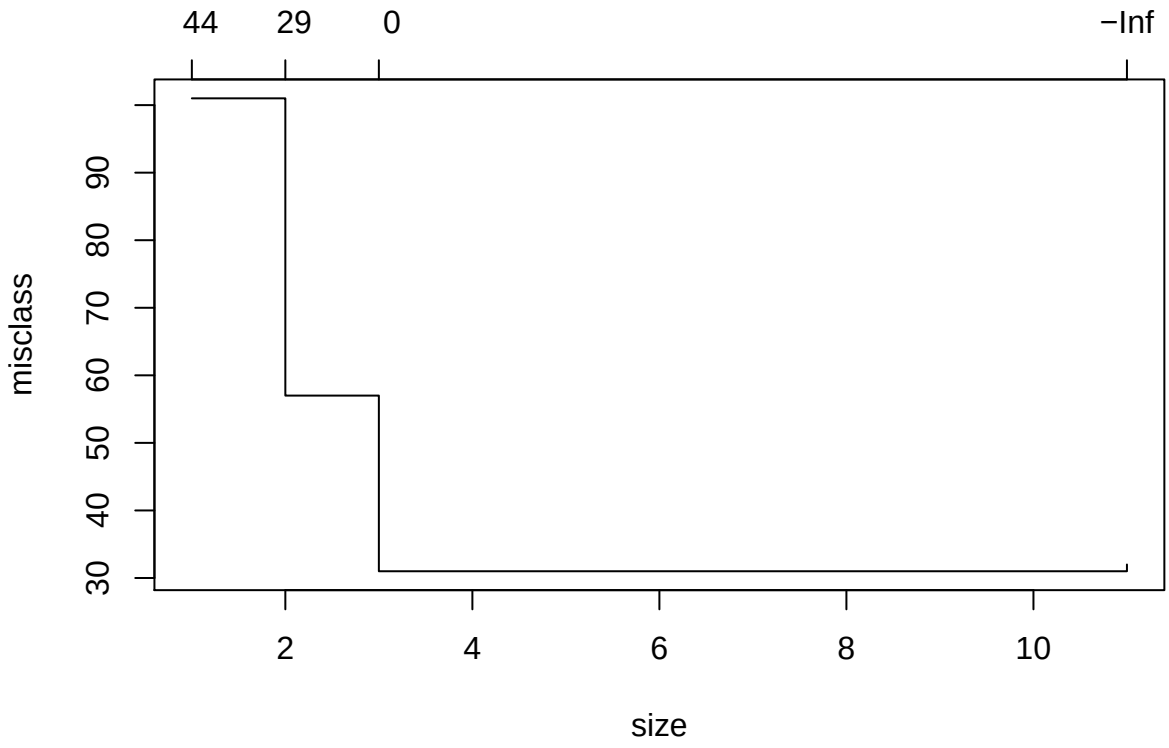

```
# Prune CART tree:
set.seed(12)
cv_cart <- cv.tree(cart)
plot(cv_cart)
```




```
pruned_cart <- prune.tree(cart, best = 3)
plot(pruned_cart)
text(pruned_cart)
```




```
# Alternatively, we could use misclassification error (instead of deviance) for pruning:
set.seed(12)
cv_cart2 <- cv.tree(cart, FUN = prune.misclass)
plot(cv_cart2)
```

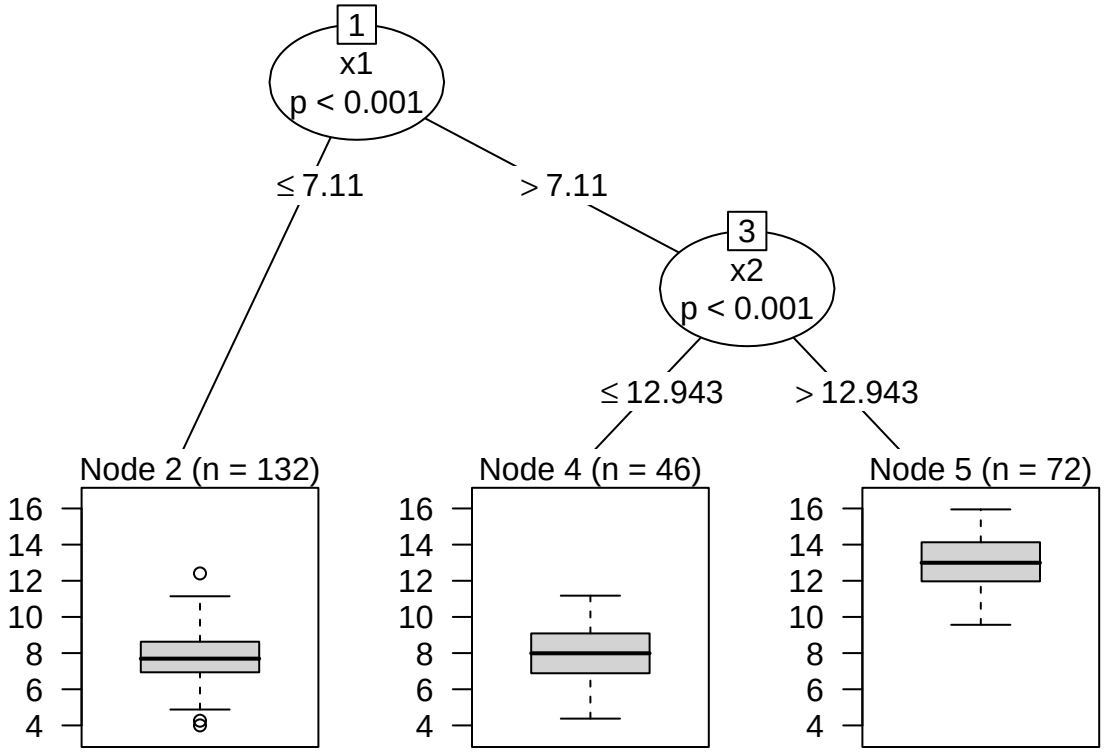



```
# In this case, deviance and misclassification error suggest trees of same size: 3 nodes
```

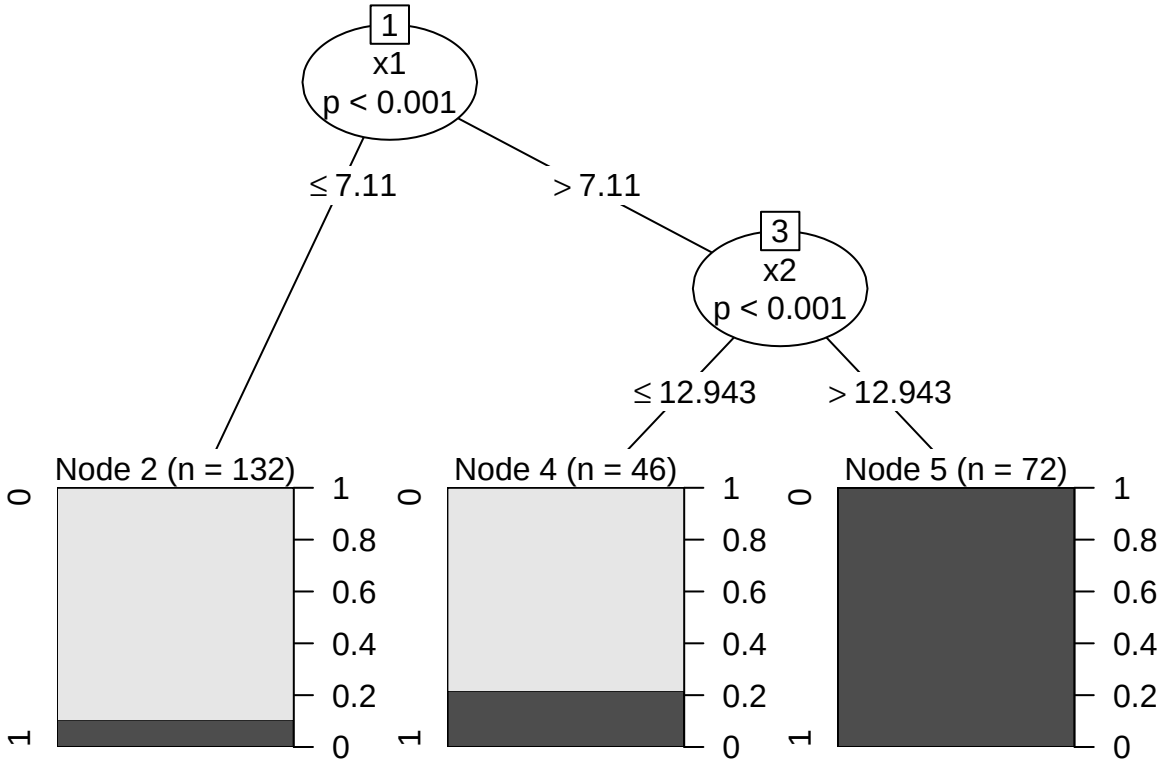
```
# fit and plot ctrees:
library("partykit")
```

```
## Loading required package: grid
```

```
ct_cont <- ctree(y_cont ~ x1 + x2)
plot(ct_cont)
```




```
ct_bin <- ctree(y_bin ~ x1 + x2)
plot(ct_bin)
```




```
# perform test for variable and split point selection:
library("coin")
```

```
## Loading required package: survival
```

```
library("strucchange")
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

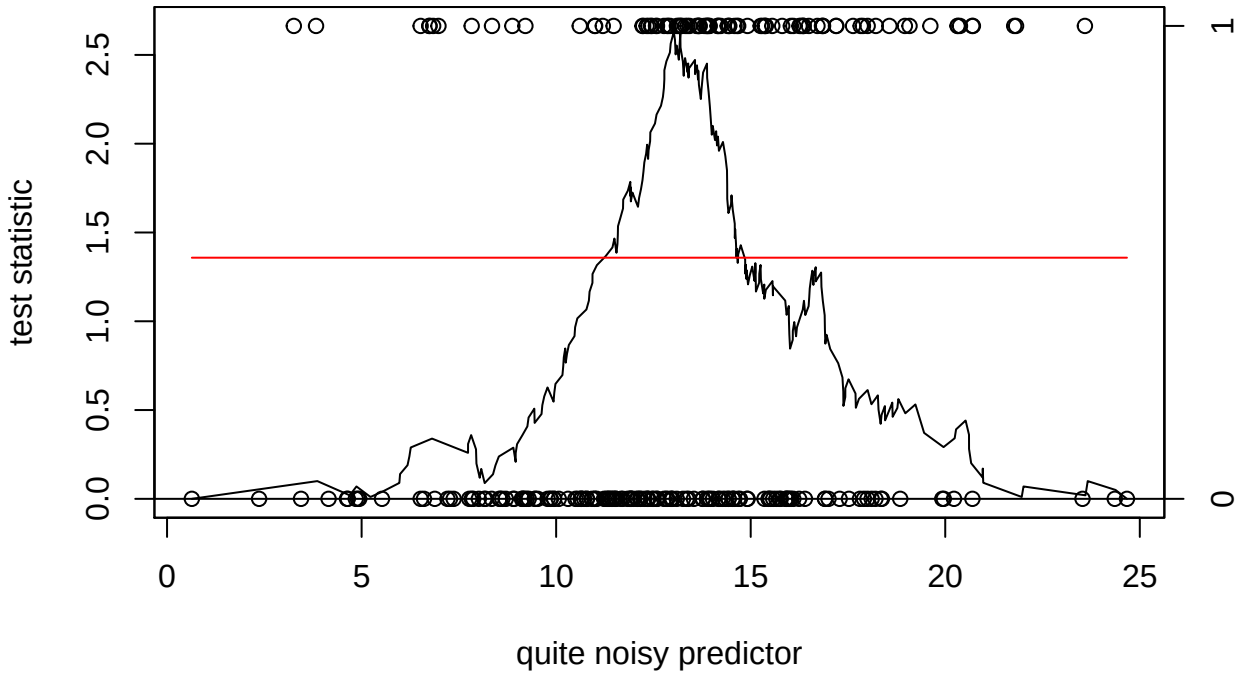
```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

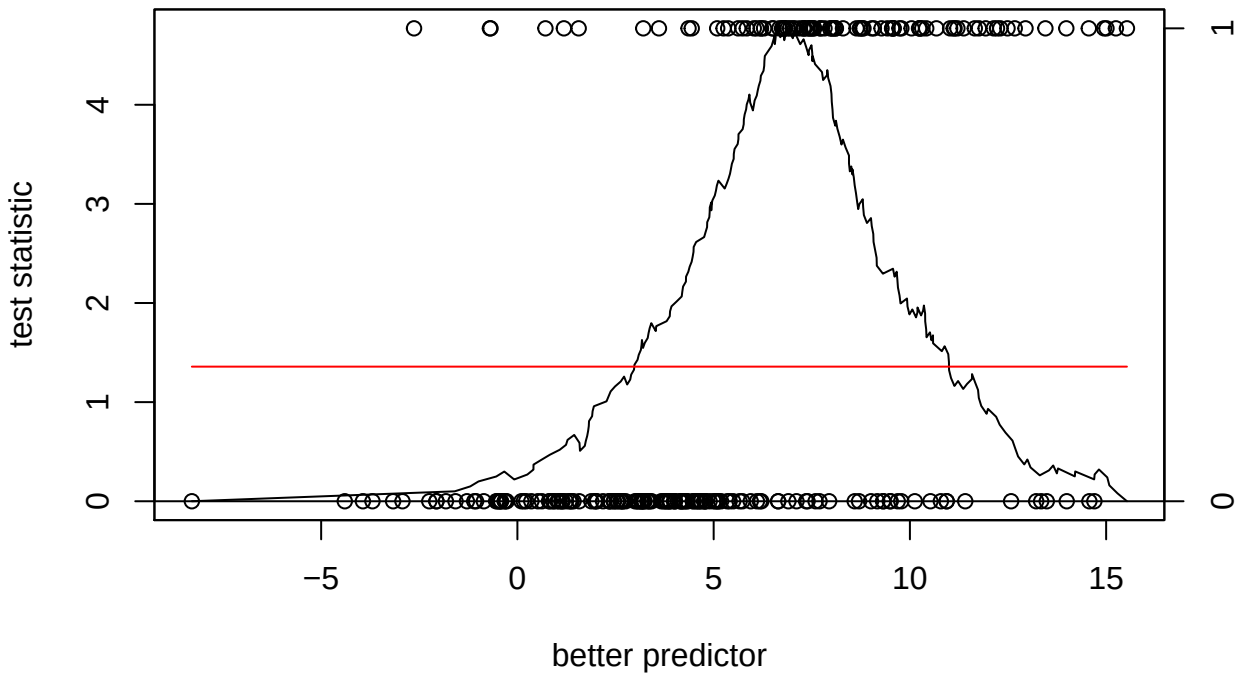
```
maxstat_test(y_bin ~ x1 + x2)
```

```
##
##  Asymptotic Generalized Maximally Selected Statistics
##
## data:  y_bin by x1, x2
## maxT = 9.5376, p-value < 2.2e-16
## alternative hypothesis: two.sided
## sample estimates:
##   "best" cutpoint: <= 7.109992
##         covariable: x1
```

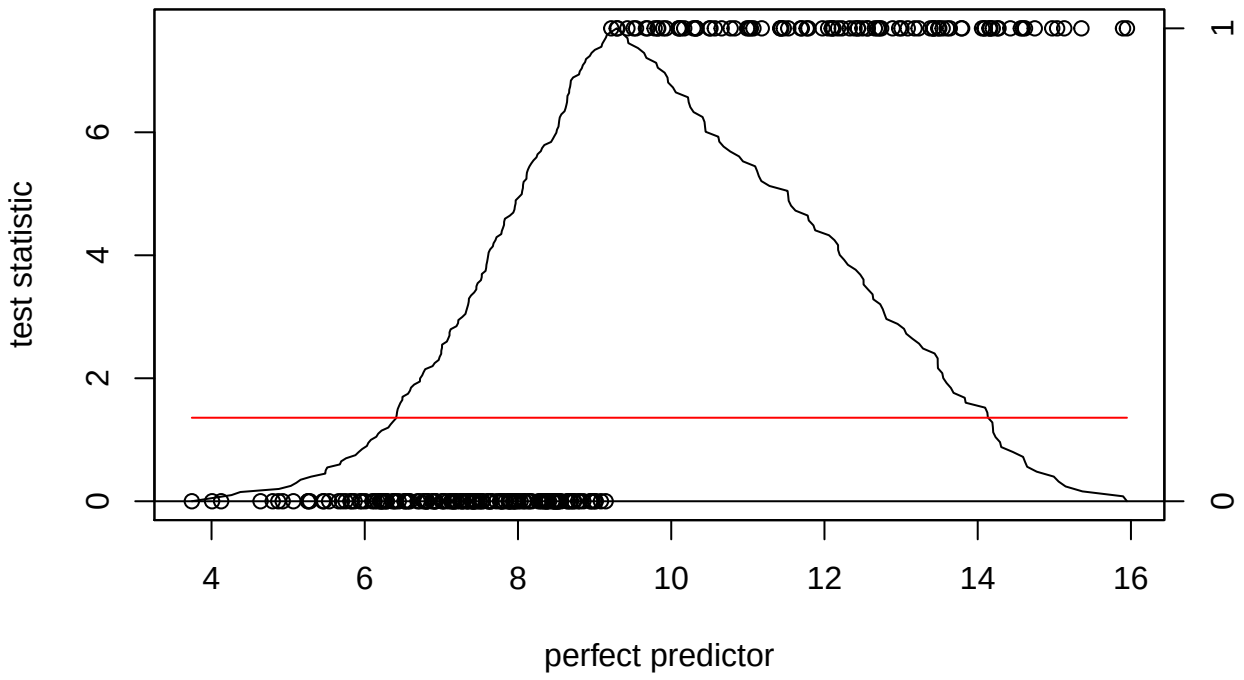
```
# Plot test statistics for X2 (noisy predictor):
gefp.x2.b <- gefp(y_bin ~ 1, family = binomial, order.by = ~ x2)
plot(gefp.x2.b, main = " ", ylab = "test statistic", xlab = "quite noisy predictor")
par(new = TRUE)
plot(x2, as.numeric(y_bin)-1, axes = FALSE, xlab = NA, ylab = NA)
mtext(side = 4, line = 3, 'class', cex = .7)
axis(side = 4, at = c(0,1))
```




```
# Plot test statistics for X1 (better predictor):
gefp.x1.b <- gefp(y_bin ~ 1, family = binomial, order.by = ~ x1)
plot(gefp.x1.b, main = " ", ylab = "test statistic", xlab = "better predictor")
par(new = TRUE)
plot(x1, as.numeric(y_bin)-1, axes=F, xlab=NA, ylab=NA)
mtext(side = 4, line = 3, 'class', cex = .7)
axis(side = 4, at = c(0,1))
```




```
# Plot test statistics for the underlying continuous y variables (perfect predictor):
gefp.y_cont.b <- gefp(y_bin ~ 1, family = binomial, order.by = ~ y_cont)
plot(gefp.y_cont.b, main = " ", ylab = "test statistic", xlab = "perfect predictor")
par(new = TRUE)
plot(y_cont, as.numeric(y_bin)-1, axes=F, xlab=NA, ylab=NA)
mtext(side = 4, line = 3, 'class', cex = .7)
axis(side = 4, at = c(0,1))
```

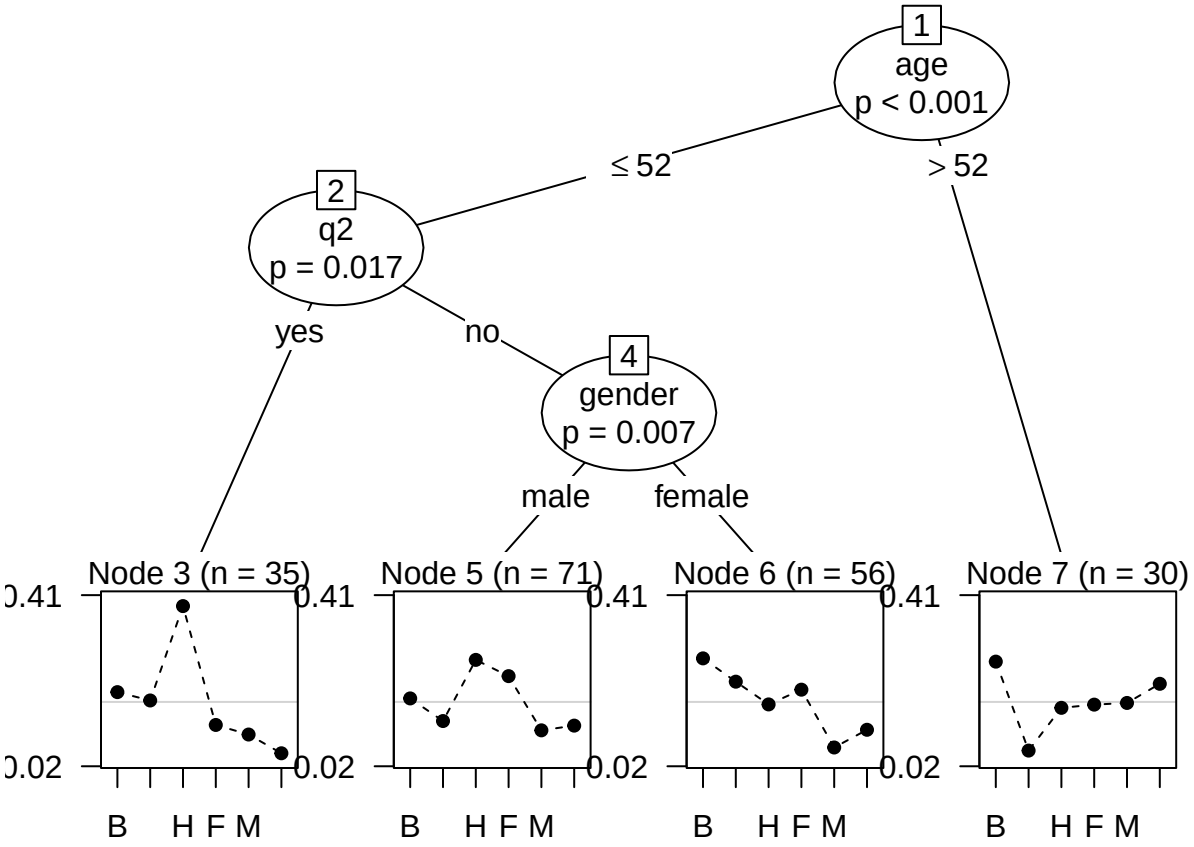


Bradley-Terry tree example

```
library("psychotree")
```

```
## Loading required package: psychotools
```

```
data("Topmodel2007", package = "psychotree")
tm_tree <- bttree(preference ~ ., data = Topmodel2007, minsplit = 5)
plot(tm_tree, abbreviate = 1)
```



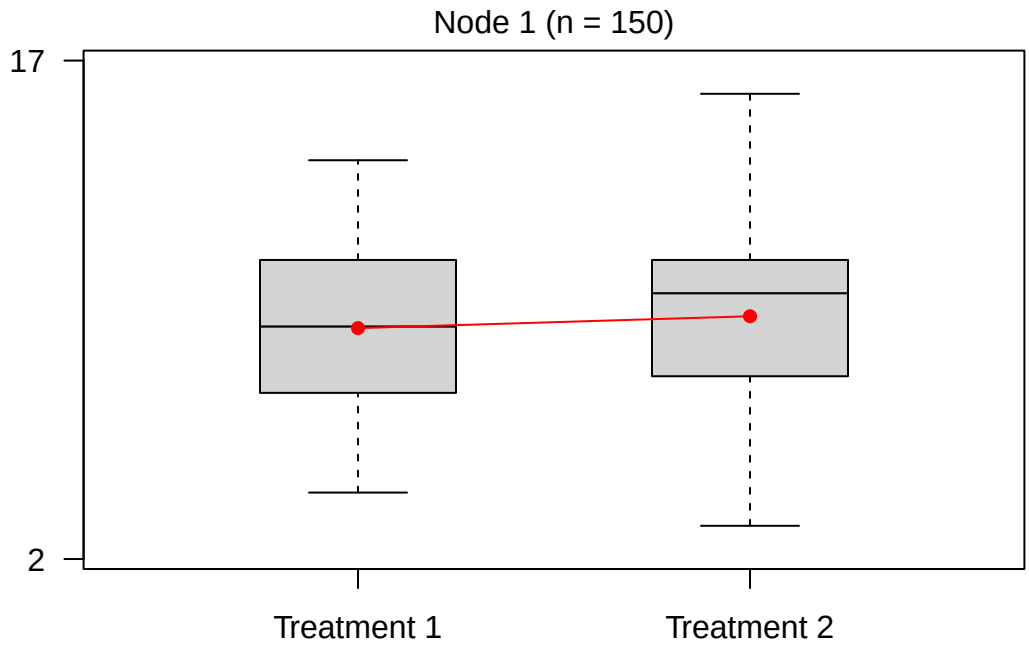
Linear (mixed-effects) model trees examples

```
library("glmertree")
```

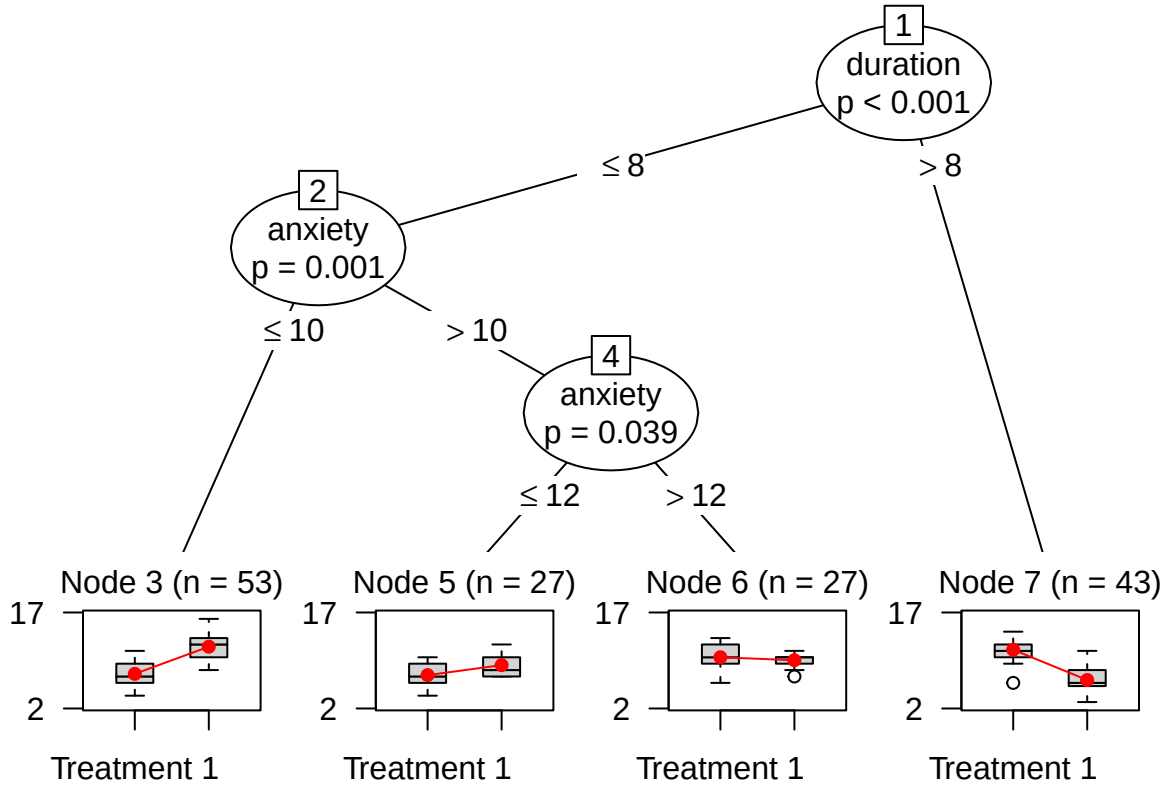
```
## Loading required package: lme4
```

```
## Loading required package: Matrix
```

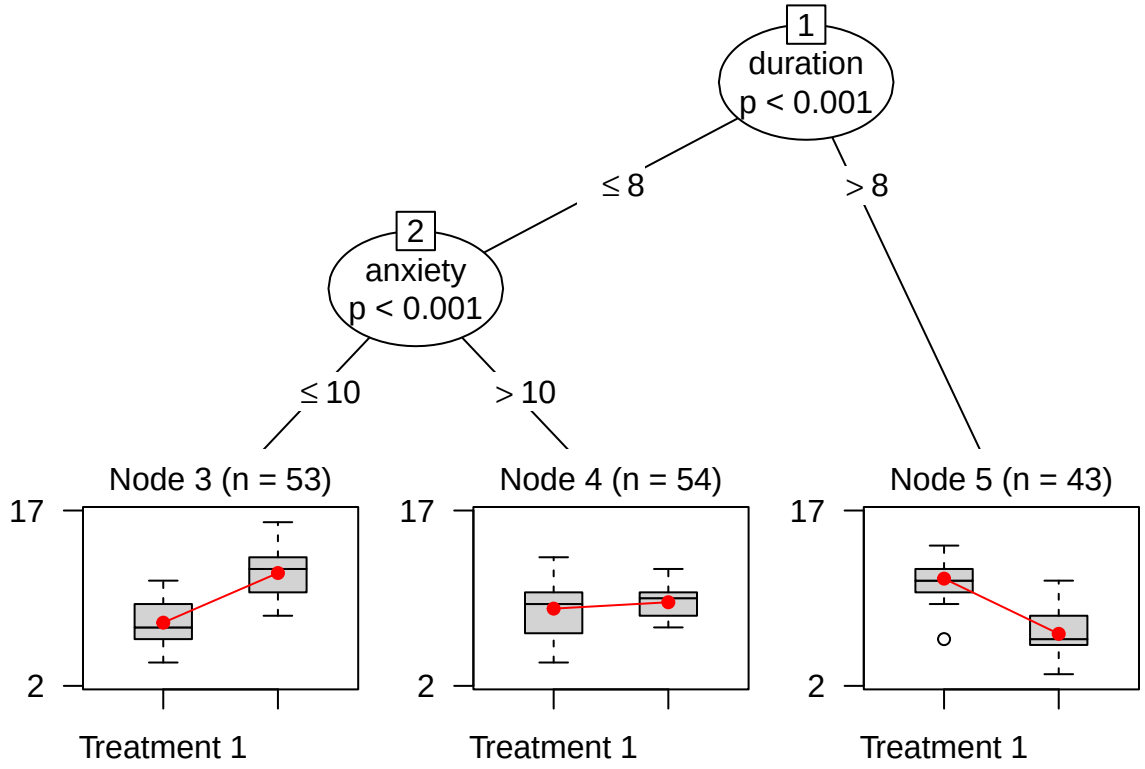
```
# fit and plot global lm:
global_lm <- lmtree(depression ~ treatment | age, data = DepressionDemo)
plot(global_lm)
```



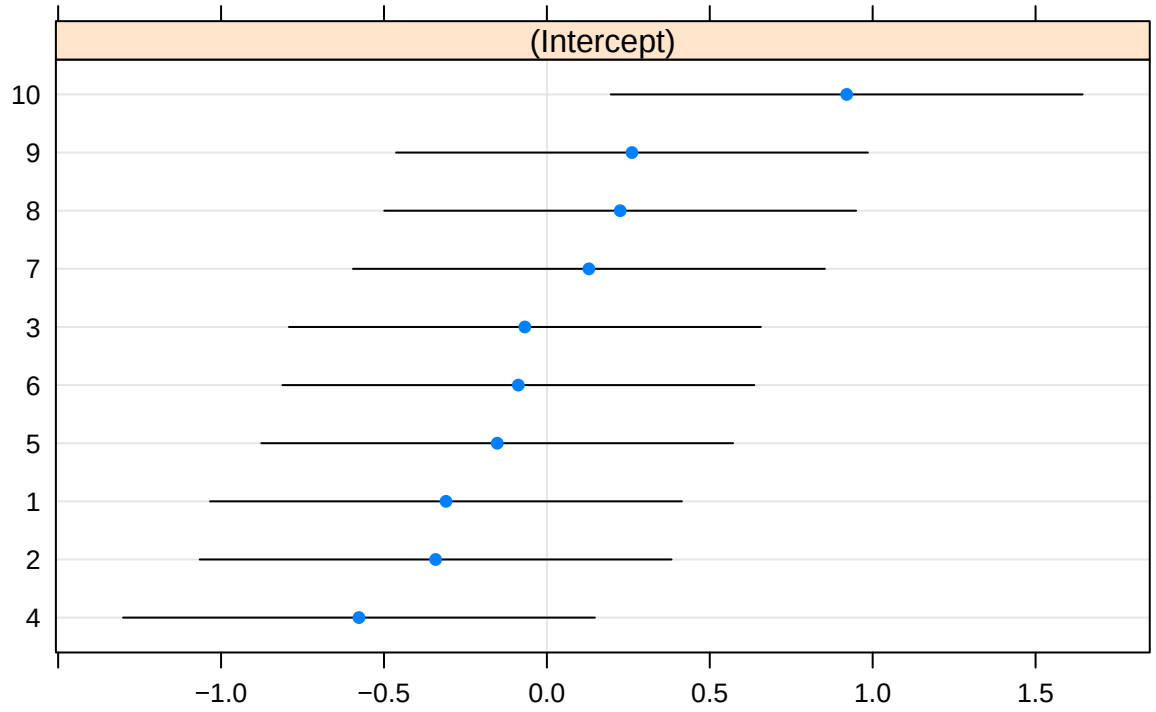

```
# fit and plot lmtree:
lm_tree <- lmtree(depression ~ treatment | age + duration + anxiety,
                  data = DepressionDemo)
plot(lm_tree)
```



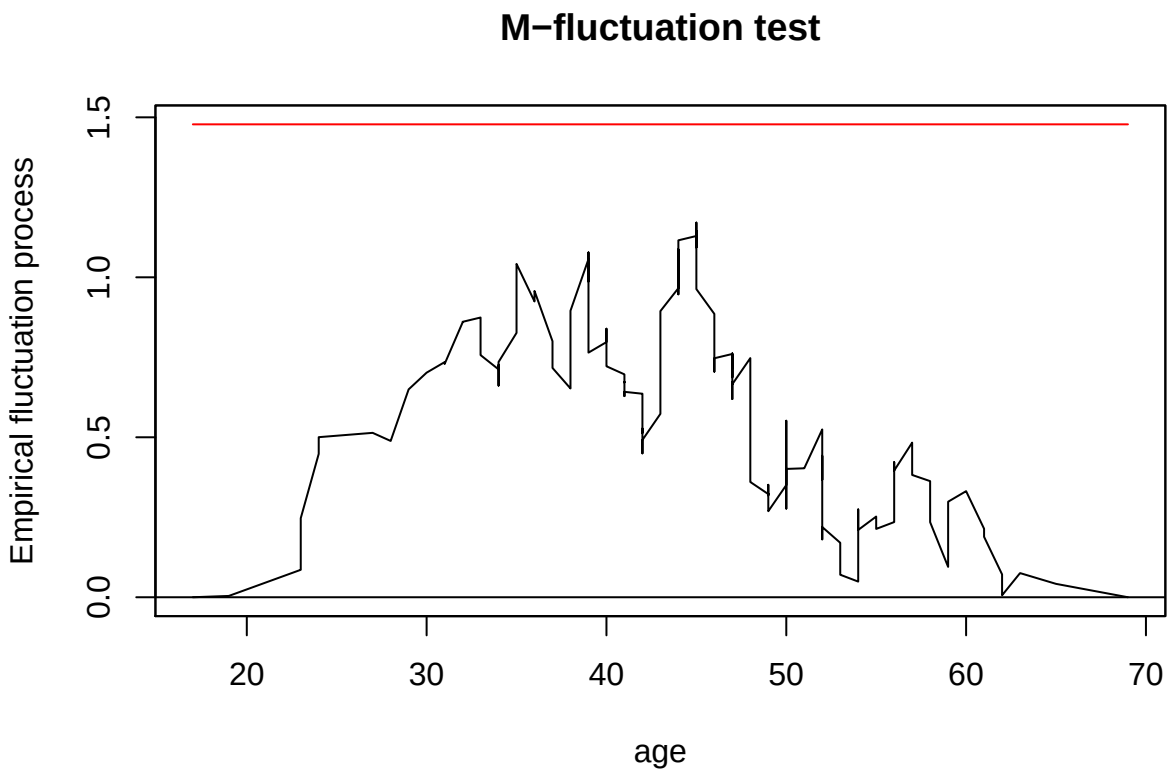

```
# fit and plot lmertree:
lmm_tree <- lmertree(depression ~ treatment | cluster | age + duration + anxiety,
                     data =DepressionDemo)
plot(lmm_tree)
```


1
duration
p < 0.001
≤ 8
> 8
2
anxiety
p < 0.001
≤ 10
> 10
Node 3 (n = 53)
Node 4 (n = 54)
Node 5 (n = 43)
17
2
17
2
17
2
Treatment 1
Treatment 1
Treatment 1

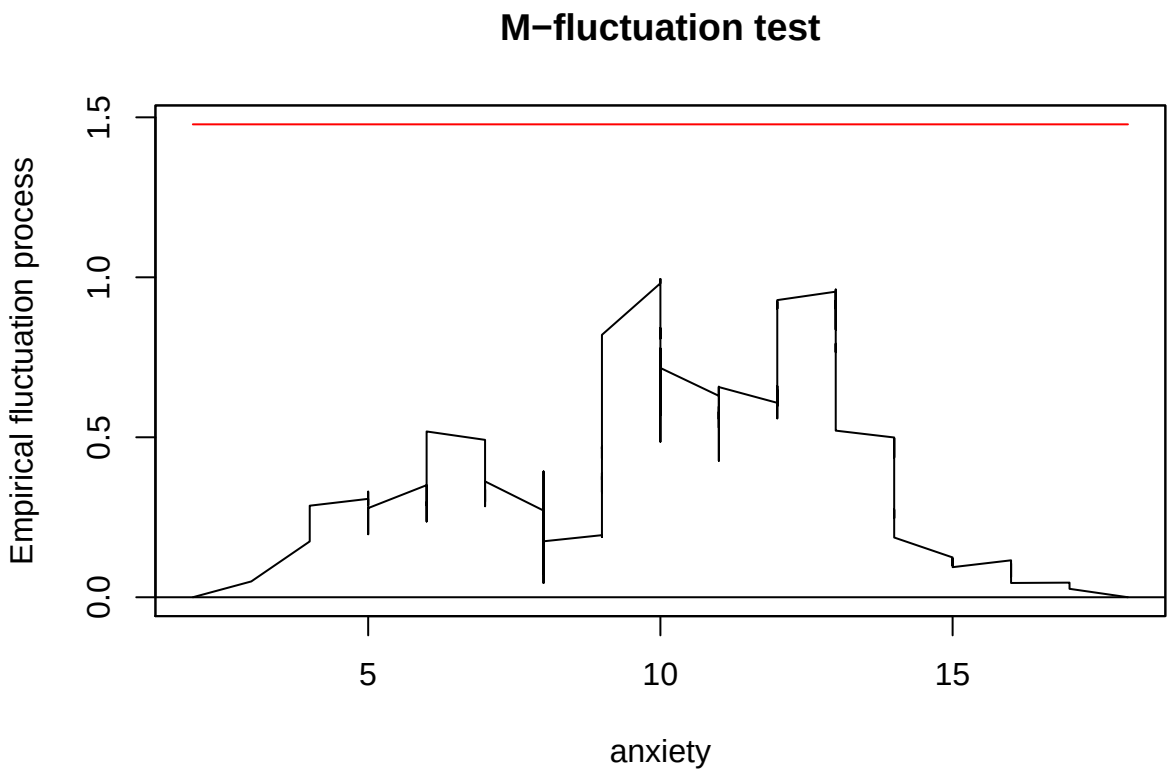
```
## $cluster
```



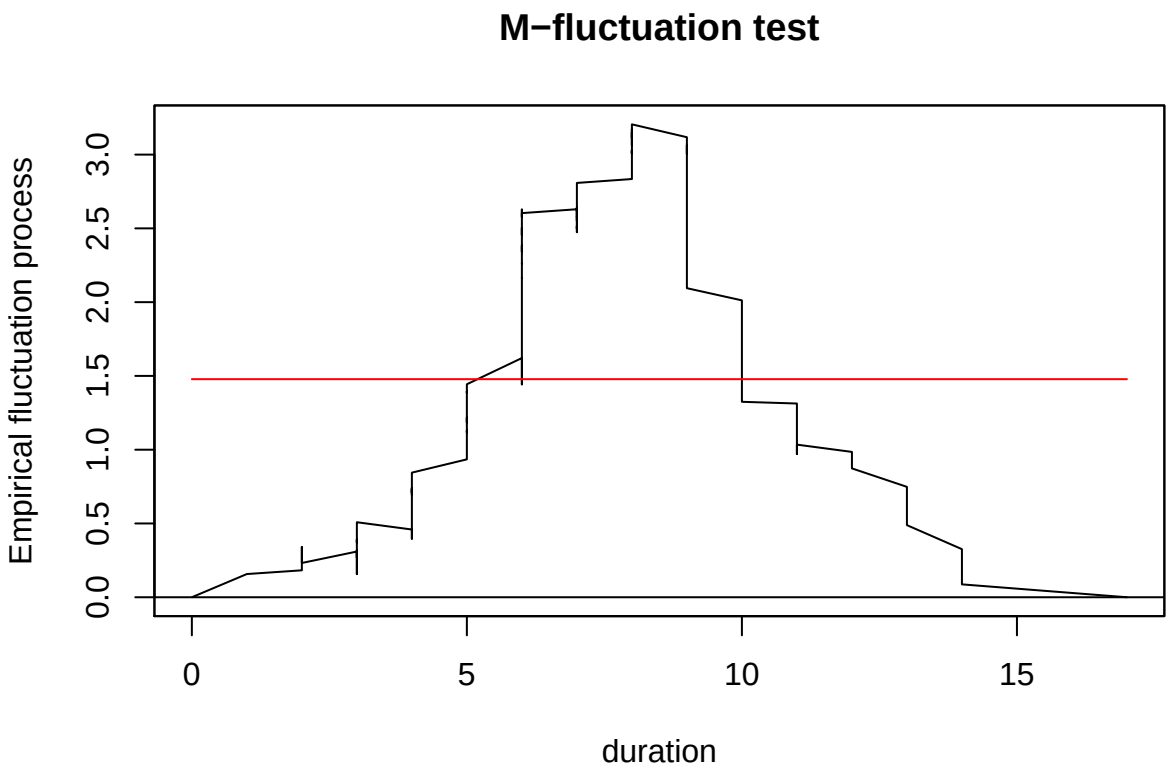

```
# inspect (in)stability statistics for the first split:
gefp.age <- gefp(lm.age <- lm(depression ~ treatment, data = DepressionDemo),
                 order.by = ~ age, data = DepressionDemo)
plot(gefp.age)
```




```
gefp.anx <- gefp(lm.anx <- lm(depression ~ treatment, data = DepressionDemo),
                 order.by = ~ anxiety, data = DepressionDemo)
plot(gefp.anx)
```

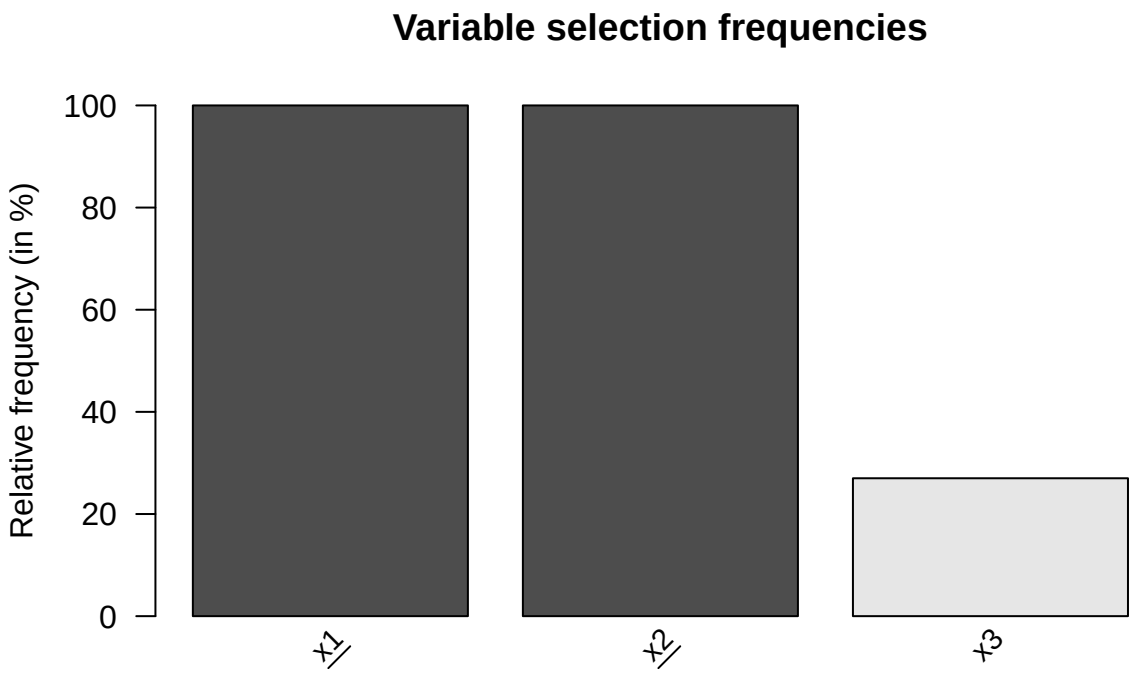



```
gefp.dur <- gefp(lm.dur <- lm(depression ~ treatment, data = DepressionDemo),
                 order.by = ~ duration, data = DepressionDemo)
plot(gefp.dur)
```



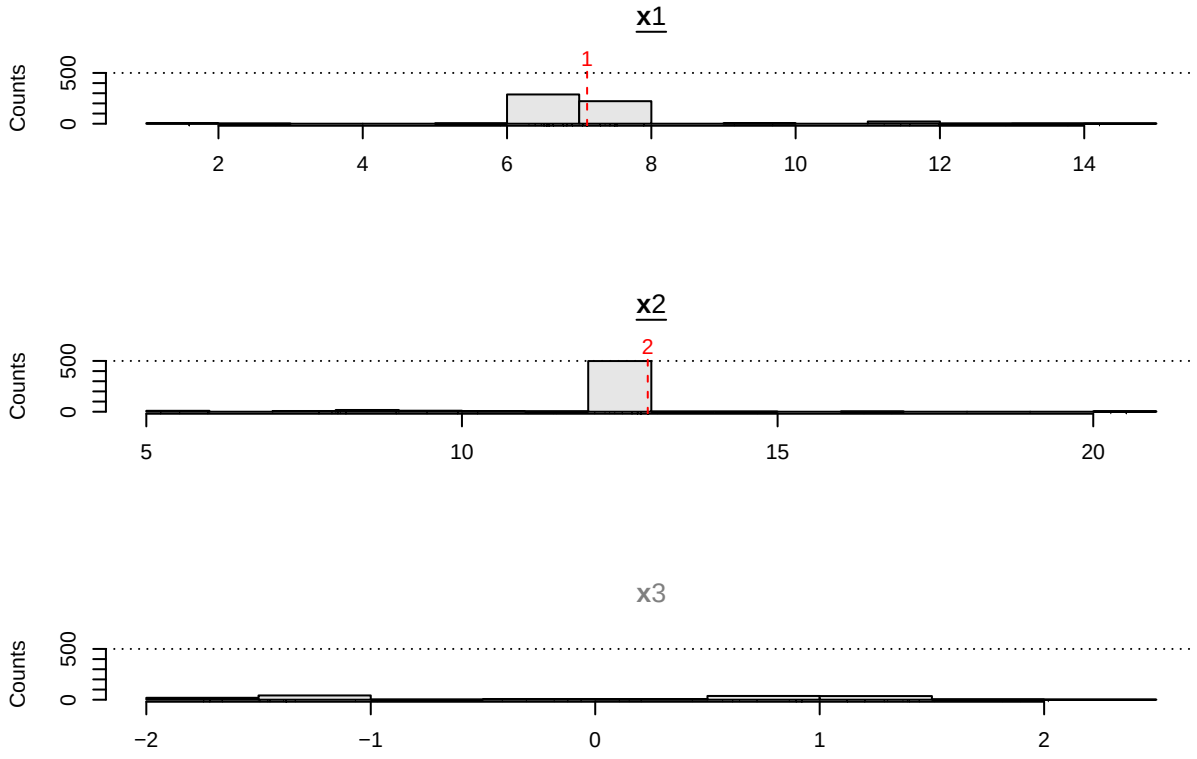
Assessing tree stability

```
library(stablelearner)
set.seed(42)
x3 <- rnorm(250)
set.seed(21)
ct_stab <- stabletree(ctree(y_bin ~ x1 + x2 + x3, data = data.frame(x1, x2, x3, y_bin)))
barplot(ct_stab)
```

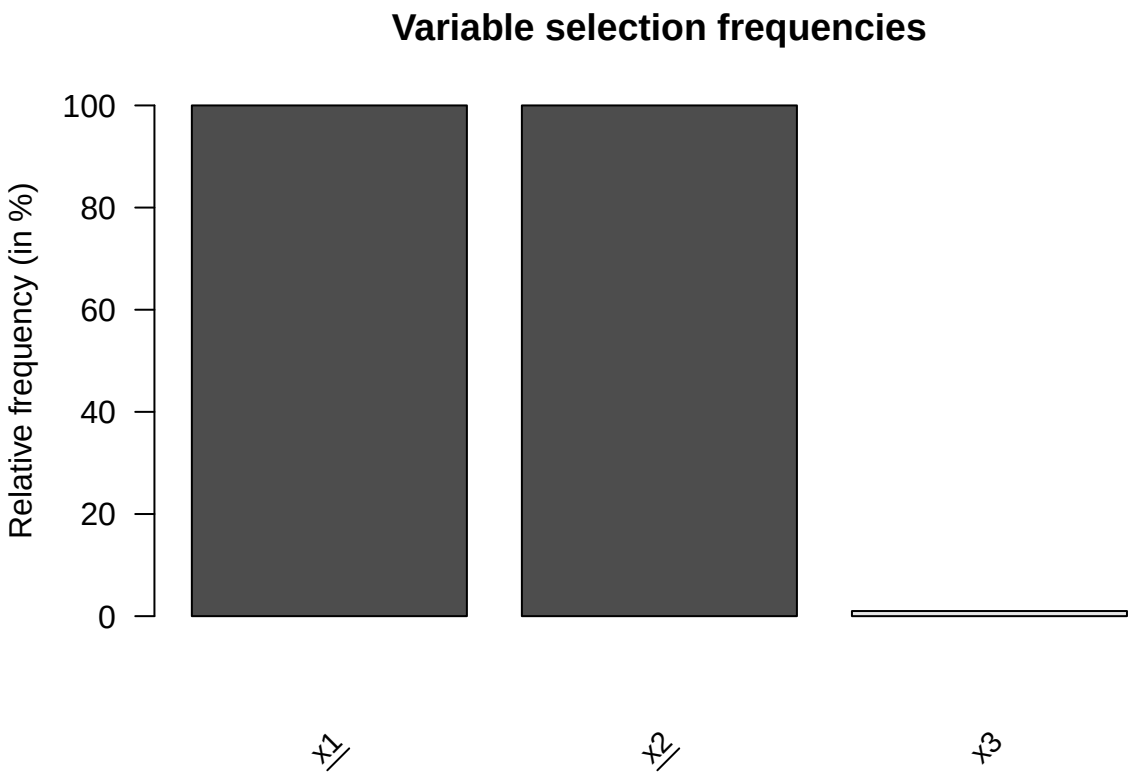



```
dev.new()
```

```
plot(ct_stab)
```




```
set.seed(21)
ct_stab2 <- stabletree(ctree(y_bin ~ x1 + x2 + x3, data = data.frame(x1, x2, x3, y_bin)),
                       sampler = subsampling)
barplot(ct_stab2)
```

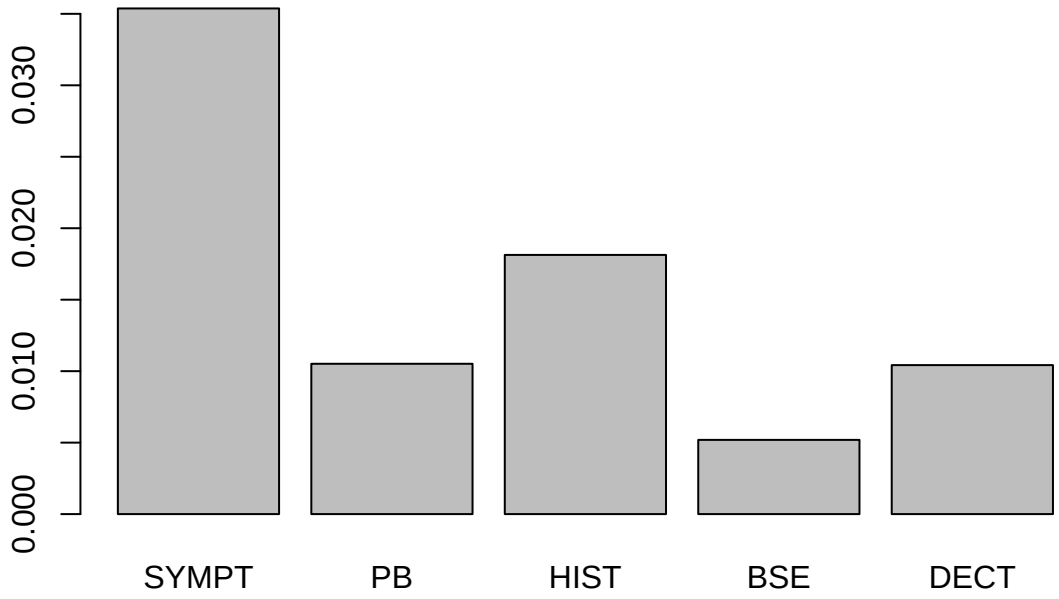


Variable importance example

```
# Note that cforest is implemented in partykit as well as party package.
# However, only party package has varimp() function, which is needed for
# calculating importances:
library("party")
```

```
## Loading required package: mvtnorm
## Loading required package: modeltools
## Loading required package: stats4
##
## Attaching package: 'modeltools'
## The following object is masked from 'package:lme4':
##
##     refit
##
## Attaching package: 'party'
## The following objects are masked from 'package:partykit':
##
##     cforest, ctree, ctree_control, edge_simple, mob, mob_control,
##     node_barplot, node_bivplot, node_boxplot, node_inner,
##     node_surv, node_terminal
```

```
data("mammoexp", package = "TH.data")
set.seed(42) # note: different seed was used for presentation
cf_me <- cforest(ME ~ SYMPT + PB + HIST + BSE + DECT, data = mammoexp)
barplot(varimp(cf_me))
```



Conditional permutation importance example

```
set.seed(12) # note: different seed was used in presentation
mycf <- cforest(score ~ ., data = readingSkills, control = cforest_unbiased(mtry = 2))
varimp(mycf)
```

```
## nativeSpeaker           age      shoeSize
##      12.98369      78.57741      17.04826
```

```
varimp(mycf, conditional = TRUE)
```

```
## nativeSpeaker           age      shoeSize
##      11.32044      43.95367       1.52258
```