# Recursive Partitioning

Semhar Michael and CPS

# Librarys

```
library("vcd")
library("lattice")
library("randomForest")
library("party")
library("partykit")
library("mboost")
library("TH.data")
library("ipred")
library("rpart")
```

Do not forget to install.packages(“”)

# Objectives

- Explain the concept of Recursive partitioning - for regression or classification problems
- Discuss different extensions (variation) of **Ensemble** modeling
- Construct a regression/classification tree model in R

# Body Fat Data

```
data("bodyfat")
head(bodyfat)
```

```
##    age DEXfat waistcirc hipcirc elbowbreadth kneebreadth anthro3a anthro3b
## 47  57  41.68     100.0   112.0          7.1          9.4     4.42     4.95
## 48  65  43.29      99.5   116.5          6.5          8.9     4.63     5.01
## 49  59  35.41      96.0   108.5          6.2          8.9     4.12     4.74
## 50  58  22.79      72.0    96.5          6.1          9.2     4.03     4.48
## 51  60  36.42      89.5   100.5          7.1         10.0     4.24     4.68
## 52  61  24.13      83.5    97.0          6.5          8.8     3.55     4.06
##    anthro3c anthro4
## 47     4.50    6.13
## 48     4.48    6.37
## 49     4.60    5.82
## 50     3.91    5.66
## 51     4.15    5.91
## 52     3.64    5.14
```

# Body Fat Data

- Question here is to predict bodyfat based on other predictor variables.
- This can be done using methods we learned in the previous sections
  - *e.g.* multiple linear regression model.
- In this section, we will see an alternative option: Recursive partitioning

# Recursive Partitioning

"There exist many algorithms for the construction of classification or regression trees but the majority of algorithms follow a simple general rule:

- First partition the observations by univariate splits in a recursive way
- Second fit a constant model in each cell of the resulting partition.
- An overview of this field of regression models is given by Murthy (1998)."

# Illustrative example

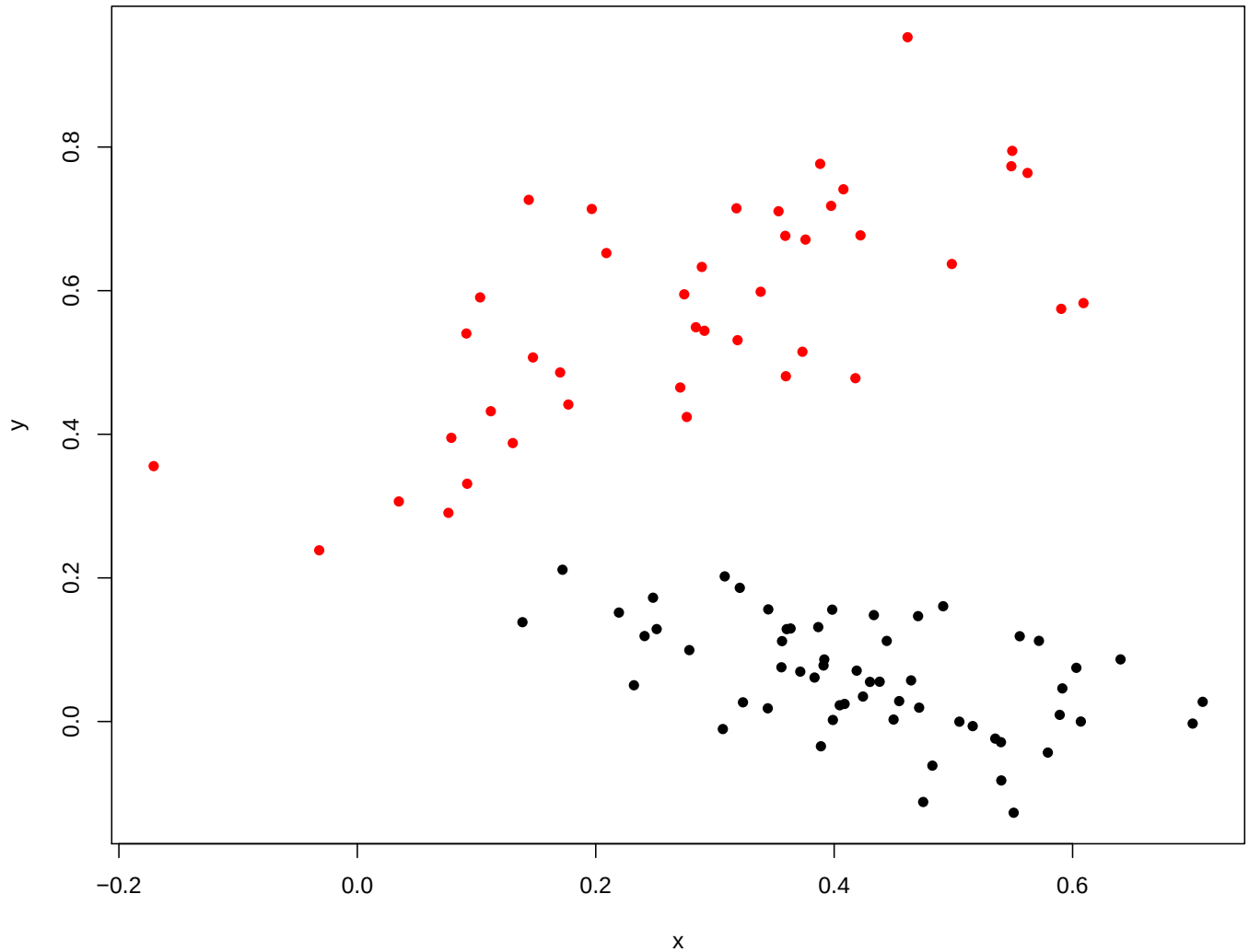



Figure 1: Classify

# Illustrative example

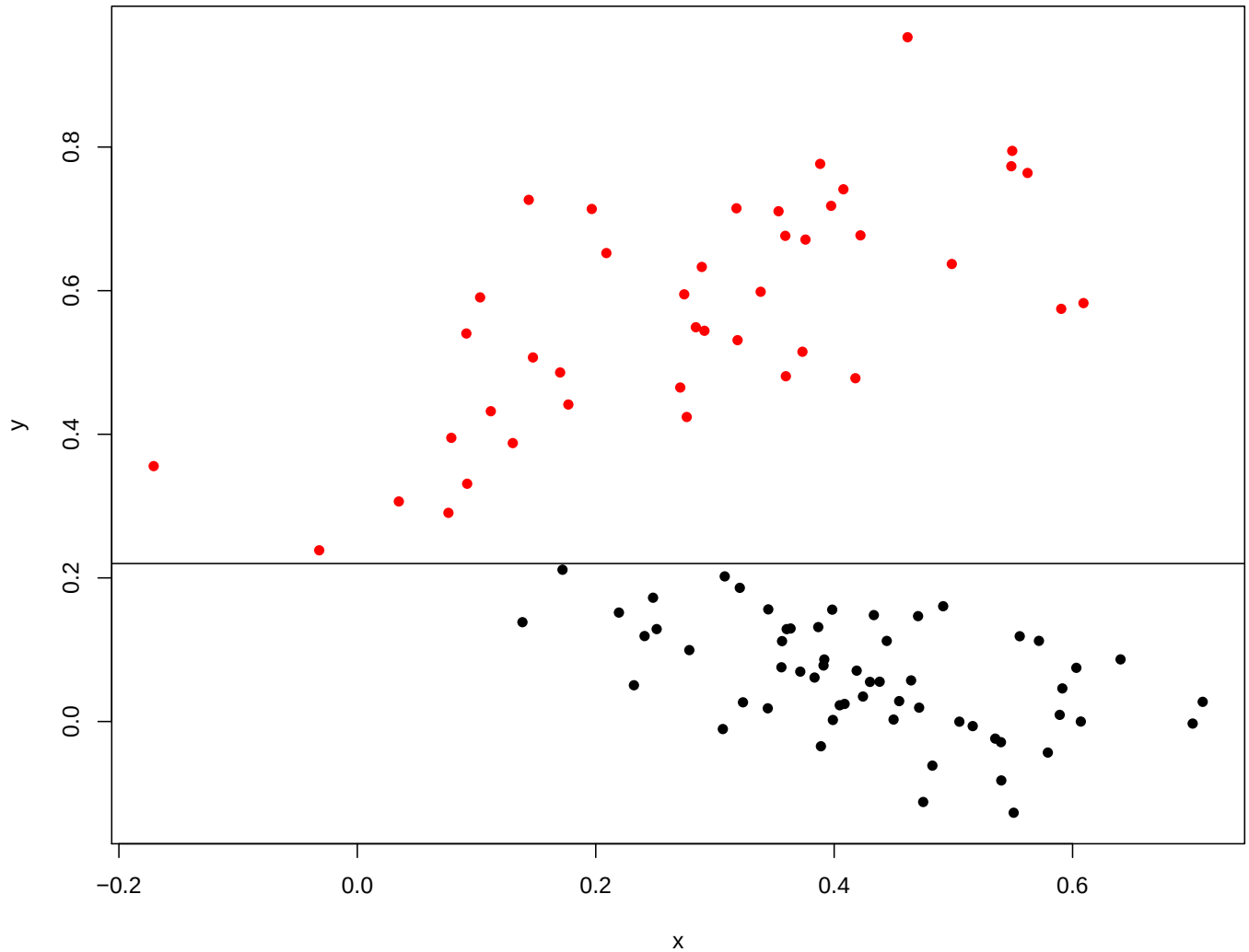



Figure 2: Classify

# Illustrative example

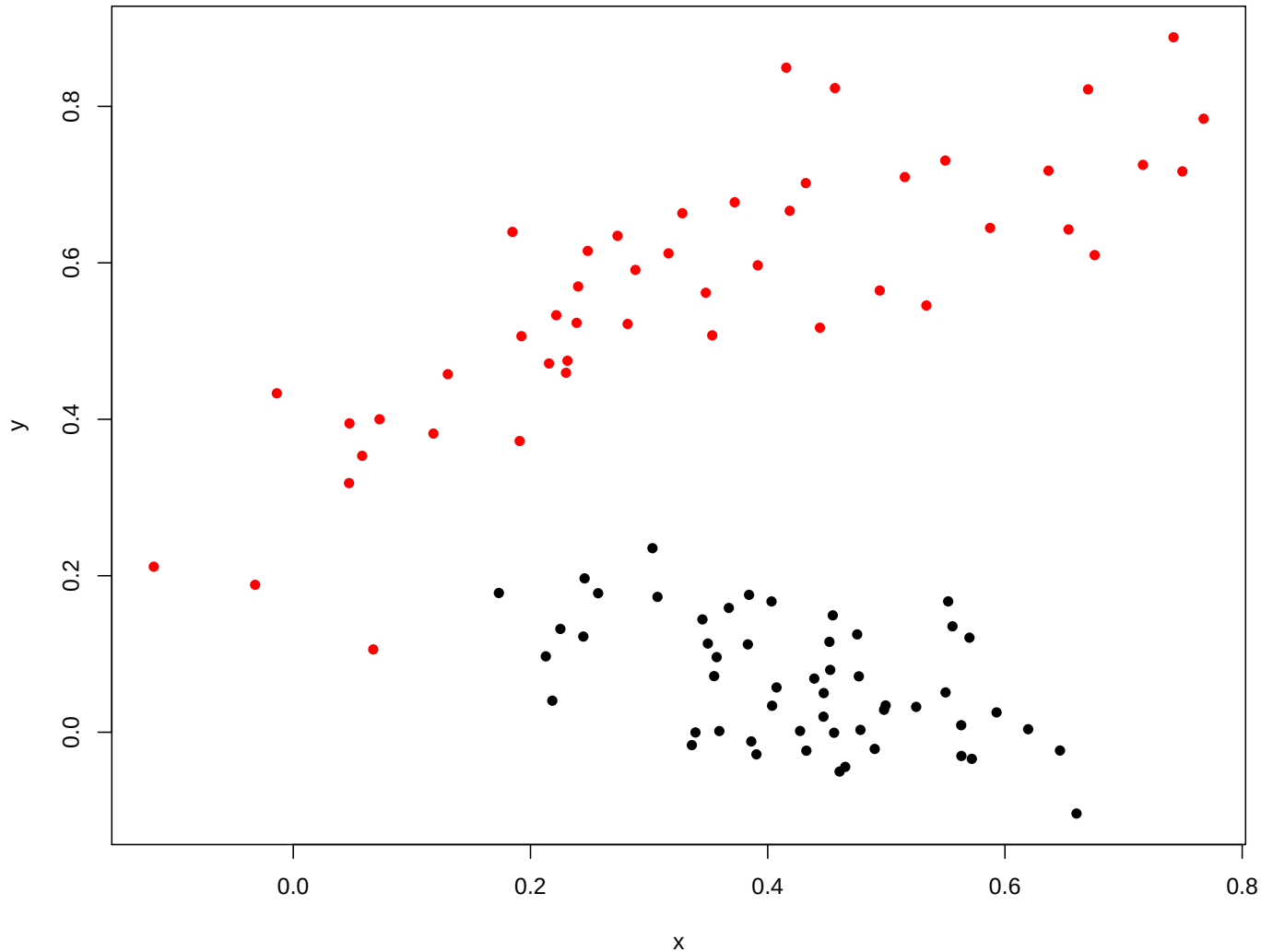



Figure 3: Classify

# Illustrative example

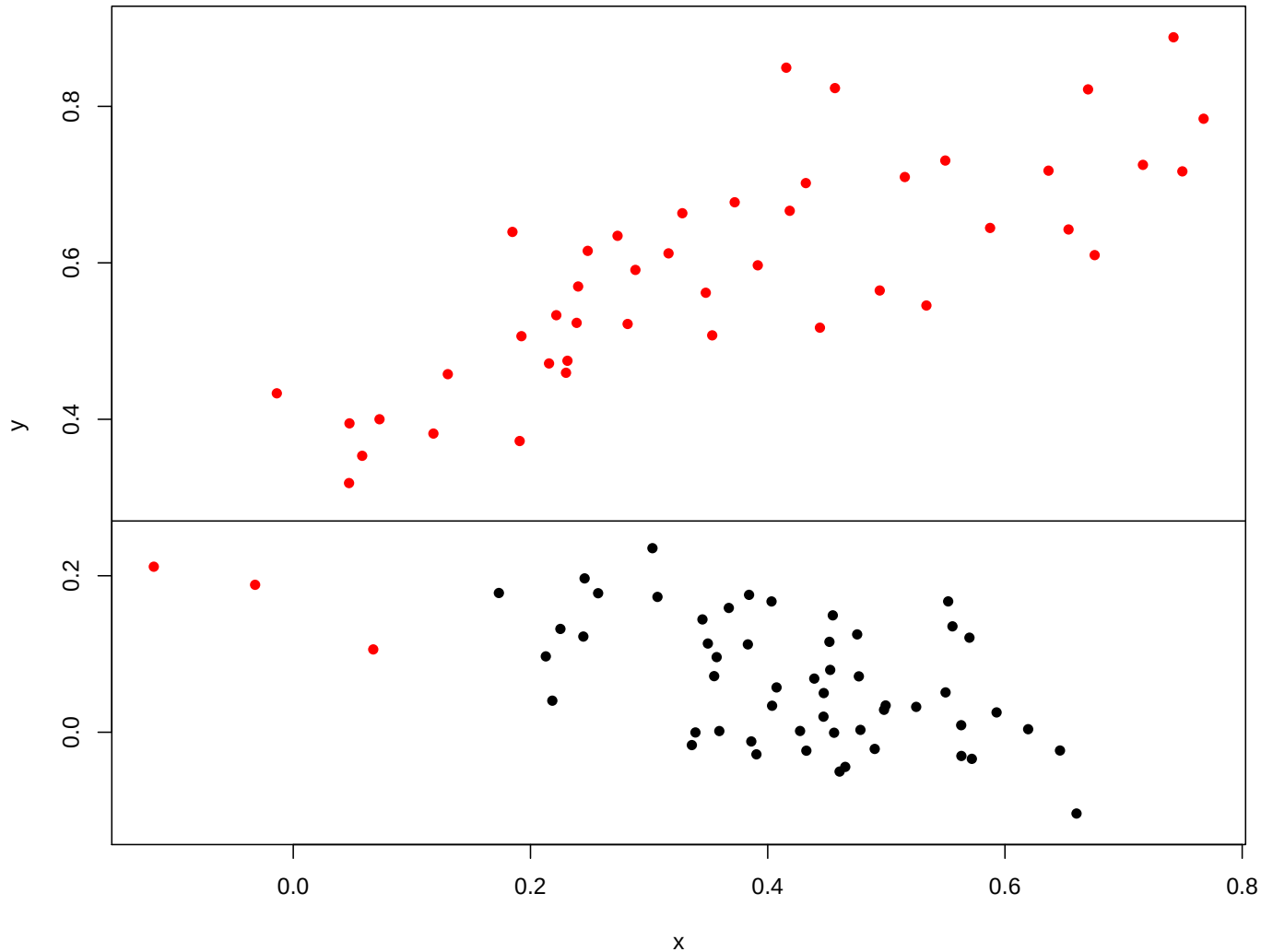



Figure 4: Classify

# Illustrative example

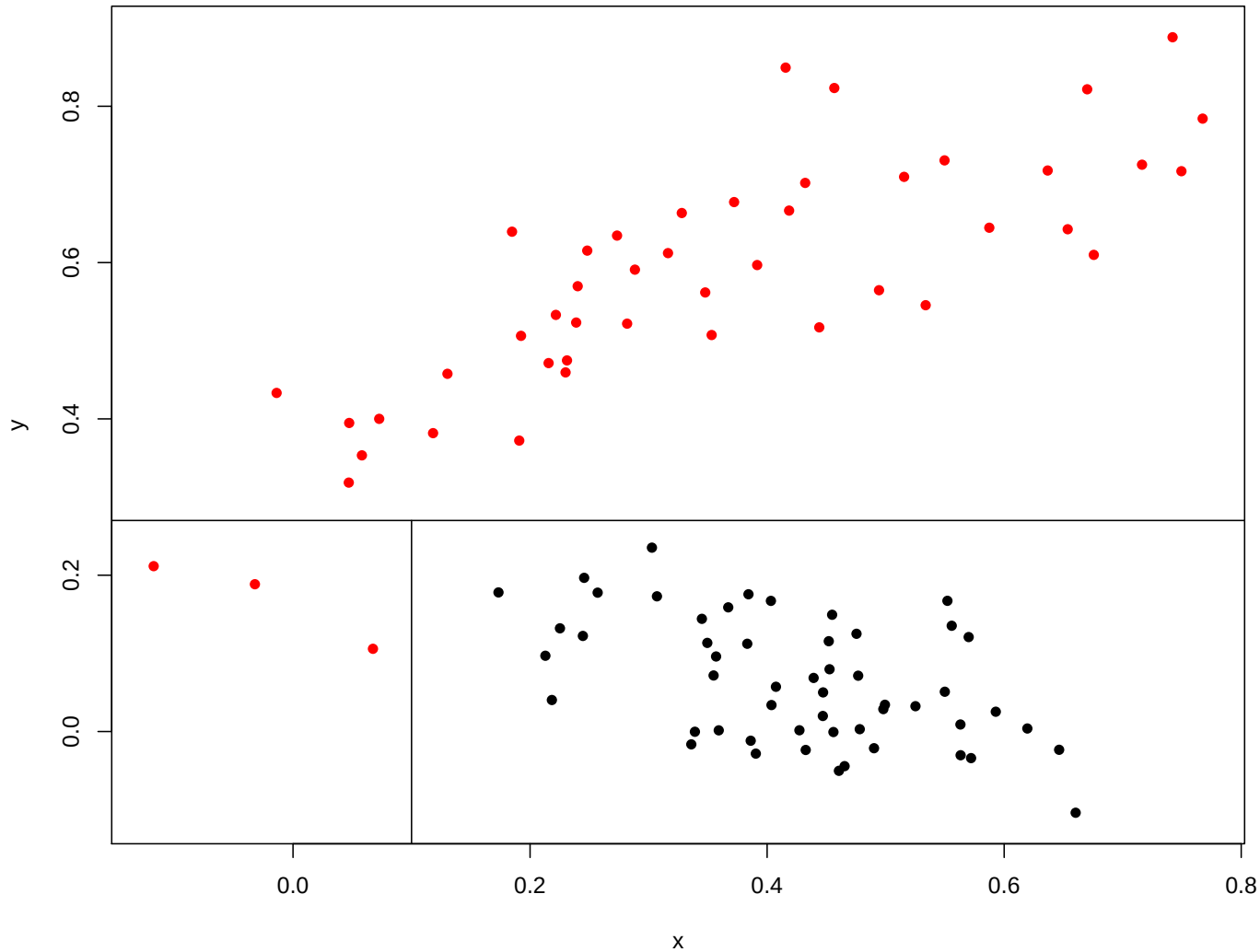



Figure 5: Classify

# Illustrative example

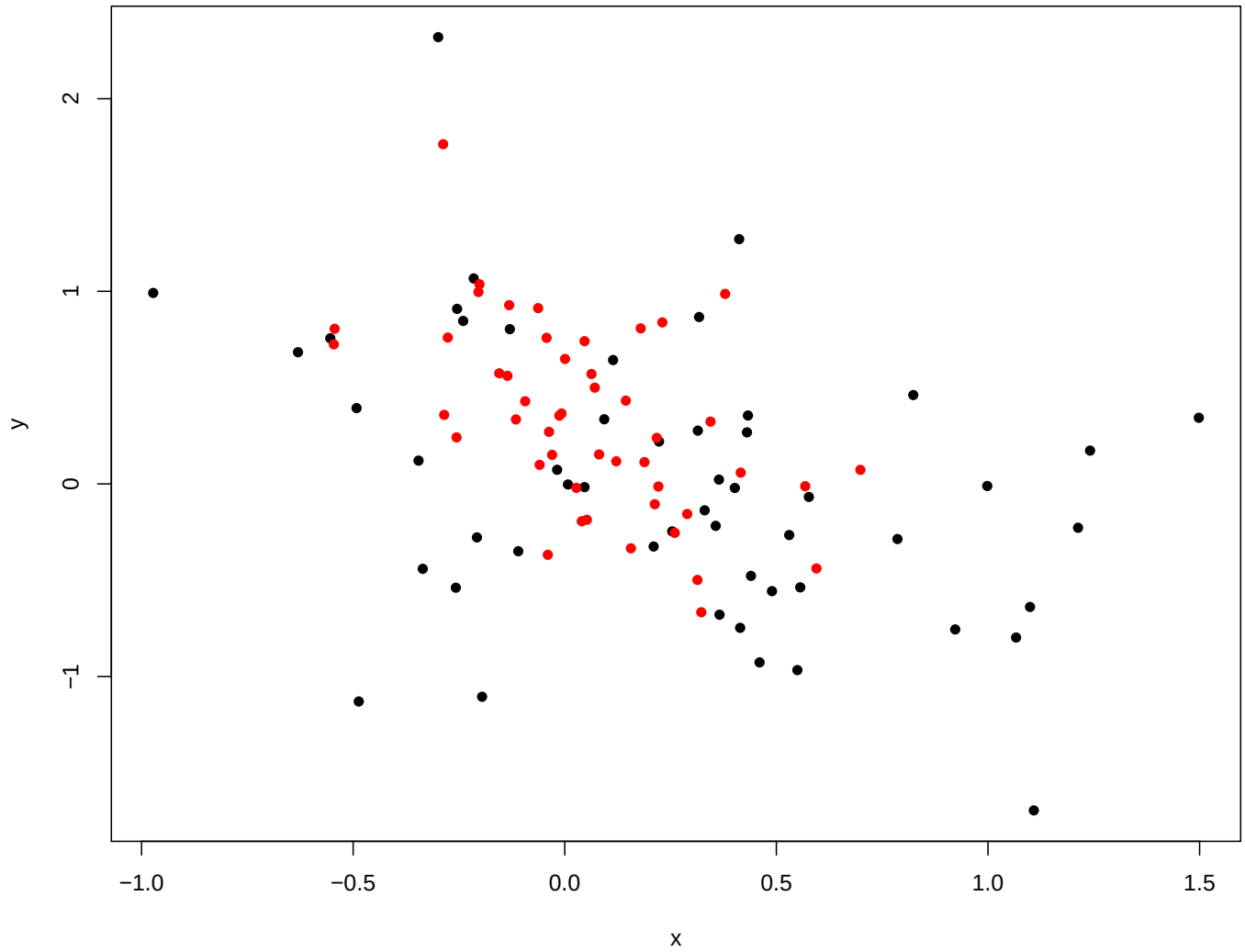



Figure 6: Classify

# Recursive Partitioning

Given the response variable $y$ and available covariates $x_1, x_2, \ldots, x_q$, these are the general steps

1. Select a covariate $x_j$ from the $q$ available covariates
2. Estimate a split point which divides the response $y_i$ into 'two' groups - for an ordered covariate the split point is a number $x_i$ - for a nominal covariate, two groups are defined by leves A, with $x_j \in A$ or not.
3. Repeat the steps 1 and 2 on the first and second group.
4. The recursion is stopped when a selected stopping criterion is met.

# Recursive Partitioning

"There exist many algorithms for the construction of classification or regression trees, they mainly differ in the following points:"

- How to select the covariate?
- How to choose the split point?
- What stopping criterion to use?

One popular algorithm is described in *classification and Regression Trees* book by Breiman et al. (1984)

```
- This is implemented in *rpart()* package in R
```

# rpart() - algorithm

## Steps

- Examine all possible splits for all covariates
- Choose the split which leads to two groups that are *purer* than the current group with respect to $y$
  - One measure of purity is *Gini* criterion for classification which is the default method in **rpart()**
- Stop the recursion - to avoid overfitting
  - Prunning - grow the tree until there is no more possible splits or using a lax convergence criterion then prune branches that are not necessary
- Get collection of nodes extending downward from root node, connected by branches until terminating leaf nodes
  - Each branch leads to decision node or leaf node
- Ones the tree is built and pruned, simple summary statistics is calculated as fitted value of $y$
  - Mean or median for continuous response and mode for the classification problems

# Ensemble methods

When the underlying relationship between covariate and response is smooth, such a split point estimate will be affected by high variability. This problem is addressed by so called ensemble methods. Here, multiple trees are grown on perturbed instances of the data set and their predictions are averaged. The simplest representative of such a procedure is called bagging (Breiman, 1996).

# Ensemble modeling

Steps

1. Draw $B$ bootstrap samples from the original data set, i.e., we draw $n$ out of $n$ observations with replacement from our $n$ original observations.
2. For the bootstrap sample grow a tree as follows: - At each node, randomly select $q$ of the $p$ predictors and restrict the splits based on the random subset of the $q$ variables ($q \ll p$).
3. Repeat that above two steps and generate a forest.

- When we are interested in the prediction for a new observation, we pass this observation through all $B$ trees and average their predictions or majority vote from all trees.

# Ensemble modeling

Note:

- Random forest: the above steps 1,2,3
- Bagging is the above steps without step 2.
- Bragging is the same as Bagging but take median instead of average.
- Bumping is the same as Bagging but choose the best instead of average.
- Model averaging is fit different models to original data and then take weighted average based on some criterion (BIC/AIC).

It has been shown that the goodness of the predictions for future cases can be improved dramatically by this or similar simple procedures.

# Boosting - Adaboost

Involves assigning weights to models (trees). It involves two sets of weights: the first $w$ is the weights associated with classification error for each observation and the second $a$ is associated with the model (tree). step:

- Boosting is an iterative procedure, and at each iteration, a model (tree) is built.
- It begins with an equal $w$-weight for all observations. Then, the $a$-weights are computed based on the $w$-weighted sum of error, and $w$-weights are updated with $a$-weights.
- With the updated weights, a new model is built and the process continues.

# Body Fat Data

In R, use rpart() function

```
set.seed(seed = 929270)
library("rpart")
bodyfat_rpart <- rpart(
  DEXfat ~ age + waistcirc + hipcirc +
    elbowbreadth + kneebreadth, data = bodyfat,
    control = rpart.control(minsplit = 10))
```

restricts the minimum number of observation required to establish a potential split is 10.

# RP-bodyfat-plot

Some useful functions for object rpart

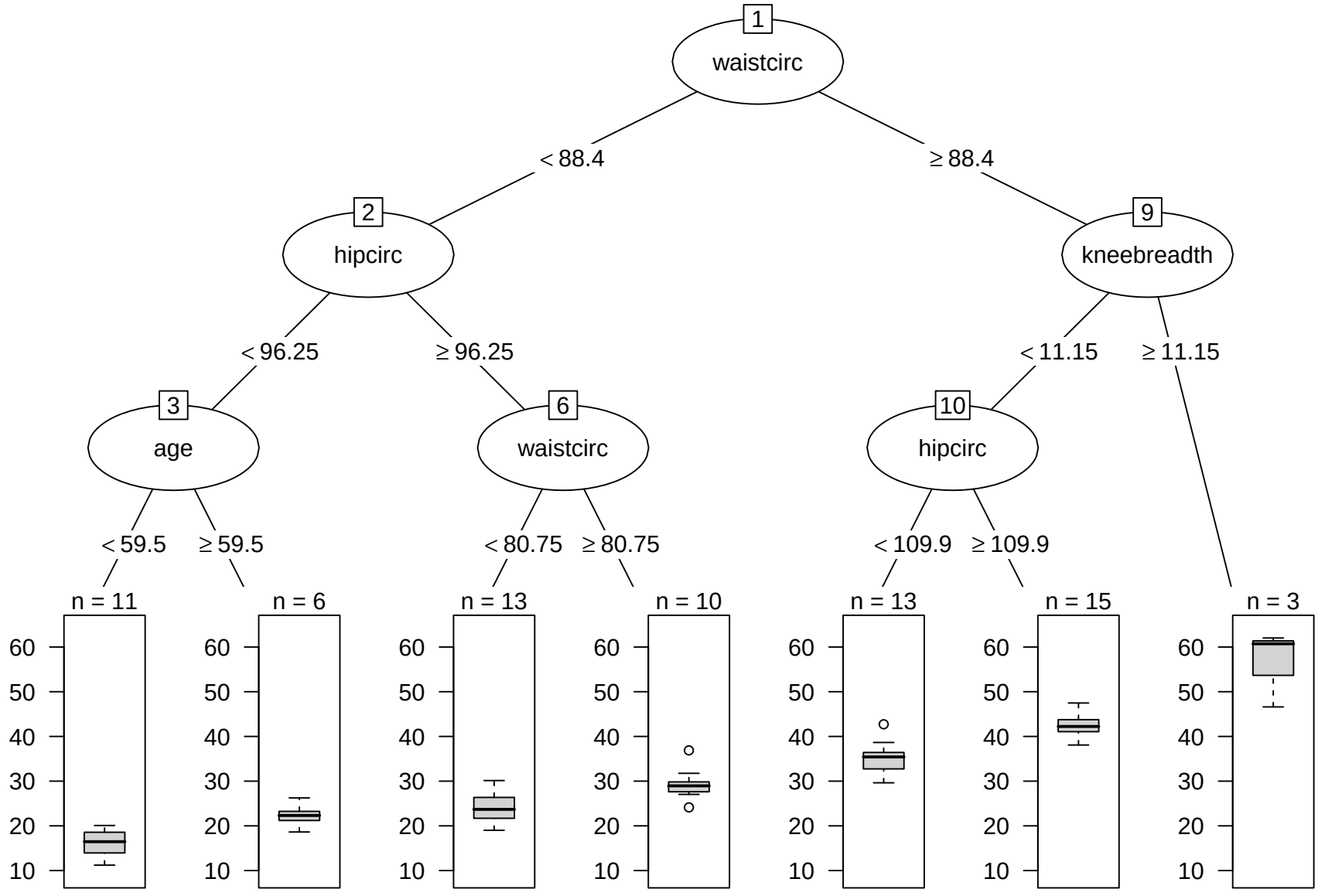
```
printcp(bodyfat_rpart)
plotcp(bodyfat_rpart)
print(bodyfat_rpart)
plot(bodyfat_rpart)
text(bodyfat_rpart)
```

# RP-bodyfat-plot

Better plot using ‘partykit’

```
library("partykit")
plot(as.party(bodyfat_rpart),
     tp_args = list(id = FALSE))
```

# RP-bodyfat-plot



# Rpart Function cp

- An advisory parameter specified according to the formula:

$$R_{cp}(T) = R(T) + cp|T|R(T_1)$$

1. $T_1$ is the tree with no splits,
2. $|T|$ is the number of splits for a tree
3. $R$ is the risk.

This scaled version is much more user friendly than the original CART formula since it is unit less.

A value of $cp = 1$ will always result in a tree with no splits.

# cp interpretation

For a regression model-

> *"If any split does not increase the overall $R^2$ of the model by at least cp (where R^2 is the usual linear-models definition) then that split is decreed to be, a priori, not worth pursuing."*

R-help for Rpart.

# RP-bodyfat-cp

Print the cptable object to determine if the tree needs to be pruned

```
print(bodyfat_rpart$cptable)
```

```
##           CP nsplit  rel error    xerror       xstd
## 1 0.66289544      0 1.00000000 1.0317304 0.17139727
## 2 0.09376252      1 0.33710456 0.4336503 0.09551925
## 3 0.07703606      2 0.24334204 0.4179093 0.09003978
## 4 0.04507506      3 0.16630598 0.3130661 0.06812183
## 5 0.01844561      4 0.12123092 0.2489333 0.05826671
## 6 0.01818982      5 0.10278532 0.2557582 0.05775228
## 7 0.01000000      6 0.08459549 0.2496436 0.05815841
```

```
opt <- which.min(bodyfat_rpart$cptable[,"xerror"])
opt
```

```
## 5
## 5
```

# RP-bodyfat-prune

## Prune back the large tree

```
cp <- bodyfat_rpart$cptable[opt, "CP"]
bodyfat_prune <- prune(bodyfat_rpart, cp = cp)
bodyfat_prune
```

```
## n= 71
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 71 8535.98400 30.78282
##    2) waistcirc< 88.4 40 1315.35800 22.92375
##      4) hipcirc< 96.25 17  285.91370 18.20765 *
##      5) hipcirc>=96.25 23  371.86530 26.40957 *
##    3) waistcirc>=88.4 31 1562.16200 40.92355
##      6) kneebreadth< 11.15 28  615.52590 39.26036
##       12) hipcirc< 109.9 13  136.29600 35.27846 *
##       13) hipcirc>=109.9 15   94.46997 42.71133 *
##      7) kneebreadth>=11.15 3  146.28030 56.44667 *
```

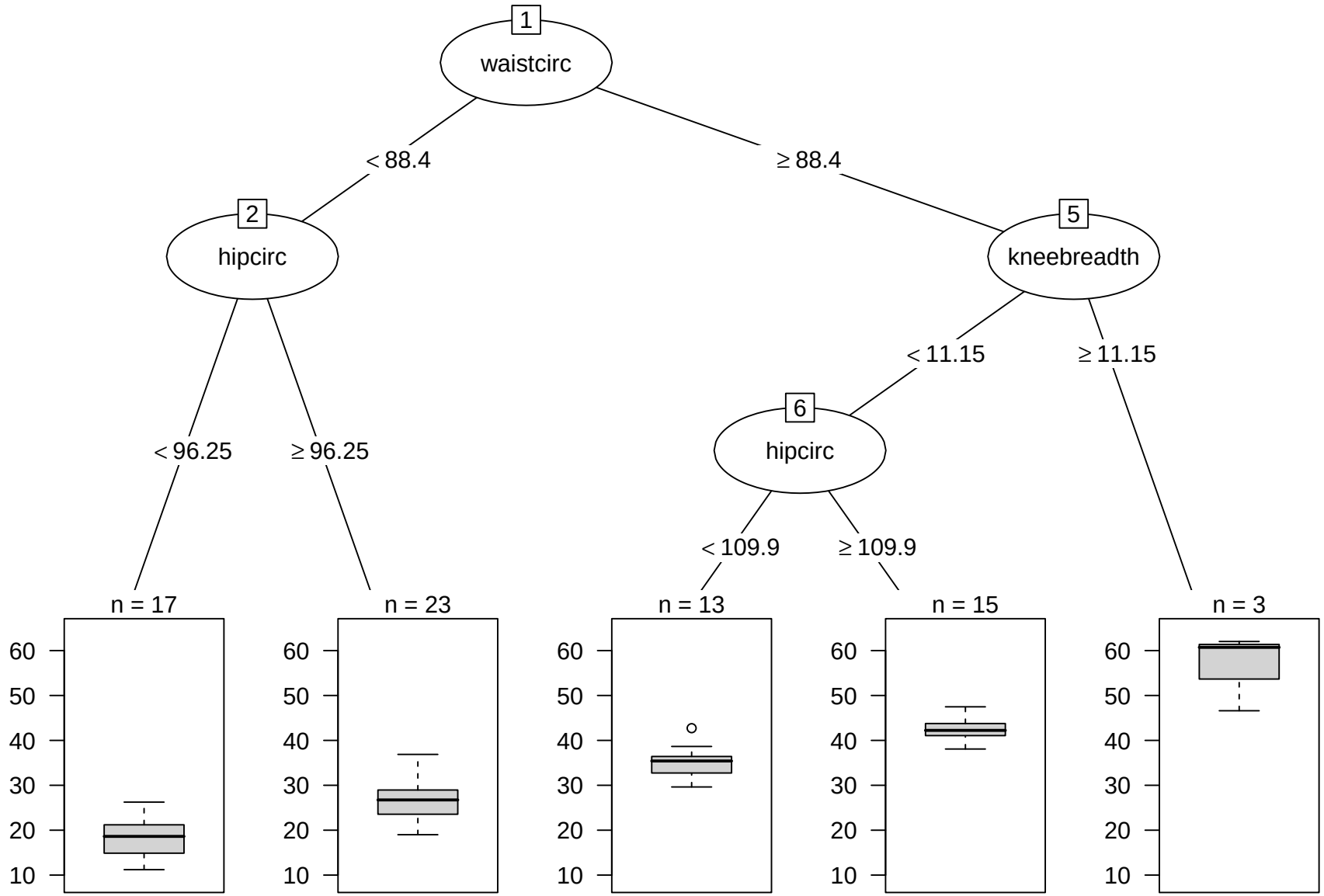
# RP-bodyfat-pruneplot

```
plot(as.party(bodyfat_prune), tp_args = list(id = FALSE))
```

Note that the inner nodes 3 and 6 have been removed.

# RP-bodyfat-pruneplot

```
plot(as.party(bodyfat_prune), tp_args = list(id = FALSE))
```



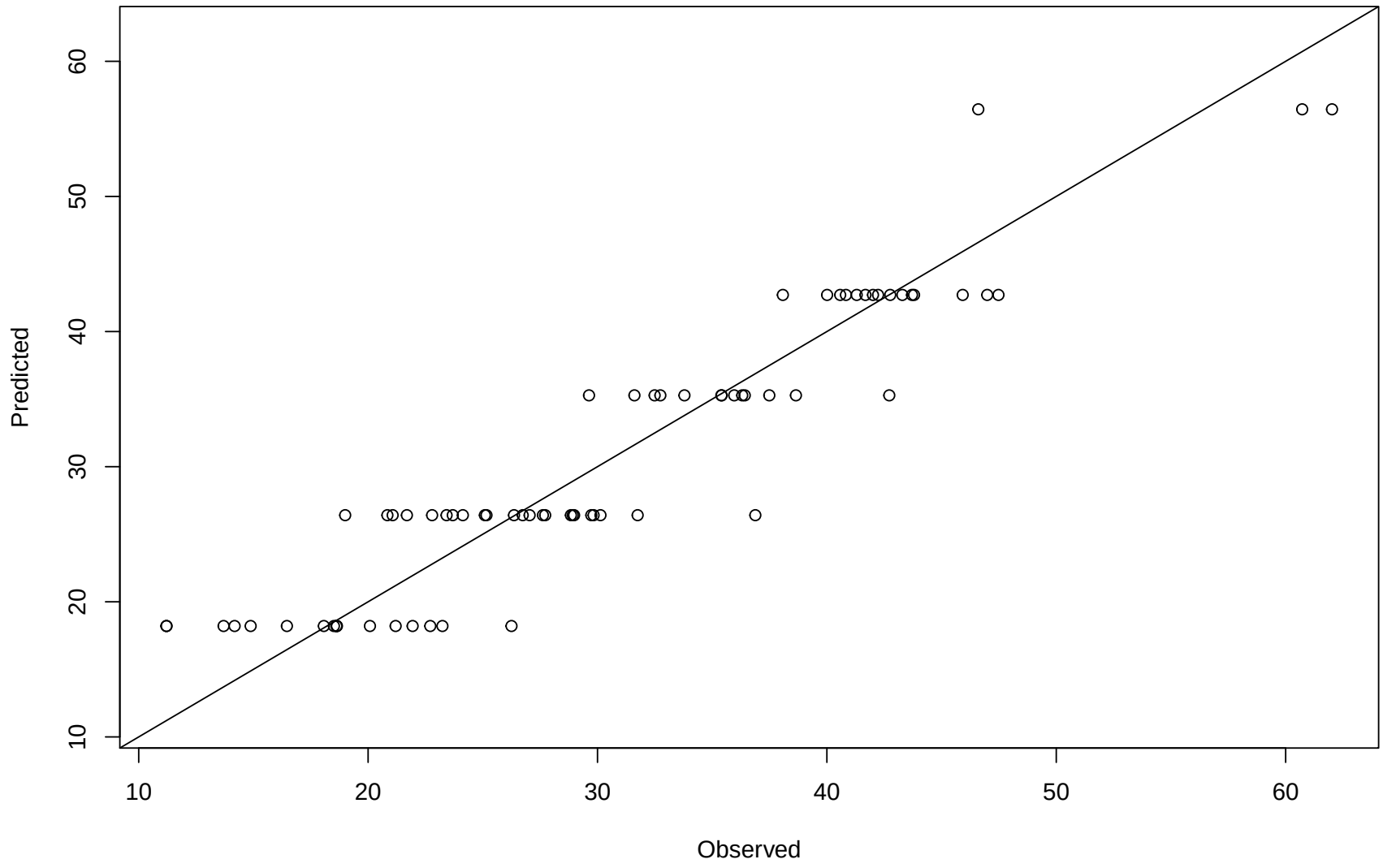
# RP-bodyfat-predict

Given a model we can predict bodyfat for new dataset given the measurement of the covariate variables used in the model.

Here we use the original data

```
DEXfat_pred <- predict(bodyfat_prune, newdata = bodyfat)
xlim <- range(bodyfat$DEXfat)
plot(DEXfat_pred ~ DEXfat, data = bodyfat, xlab = "Observed",
     ylab = "Predicted", ylim = xlim, xlim = xlim)
abline(a = 0, b = 1)
```

# RP-bodyfat-predict



# Calculate mean square error (MSE)

```
mean((bodyfat$DEXfat - DEXfat_pred)^2)
```

```
## [1] 14.575
```

Exercise

- change the seed number (slide 12) and build a tree (go through pruning as well). Calculate the MSE after predicting using the new tree built. Is it the same as above?

# Set Seed

```
set.seed(290875)
```

# RP- glaucoma data

- Glaucoma is a neuro-degenerative disease of the optic nerve and is one of the major reasons for blindness in elderly people.
- For 196 people, 98 patients suffering glaucoma and 98 controls which have been matched by age and sex, 62 numeric variables derived from the laser scanning images are available.
- The data are available as GlaucomaM from package **TH.data**.
- The variables describe the morphology of the optic nerve head, i.e., measures of volumes and areas in certain regions of the eye background. Those regions have been manually outlined by a physician.
- Our aim is to construct a prediction model which is able to decide whether an eye is affected by glaucomateous changes based on the laser image data.

# RP-glaucoma-rpart

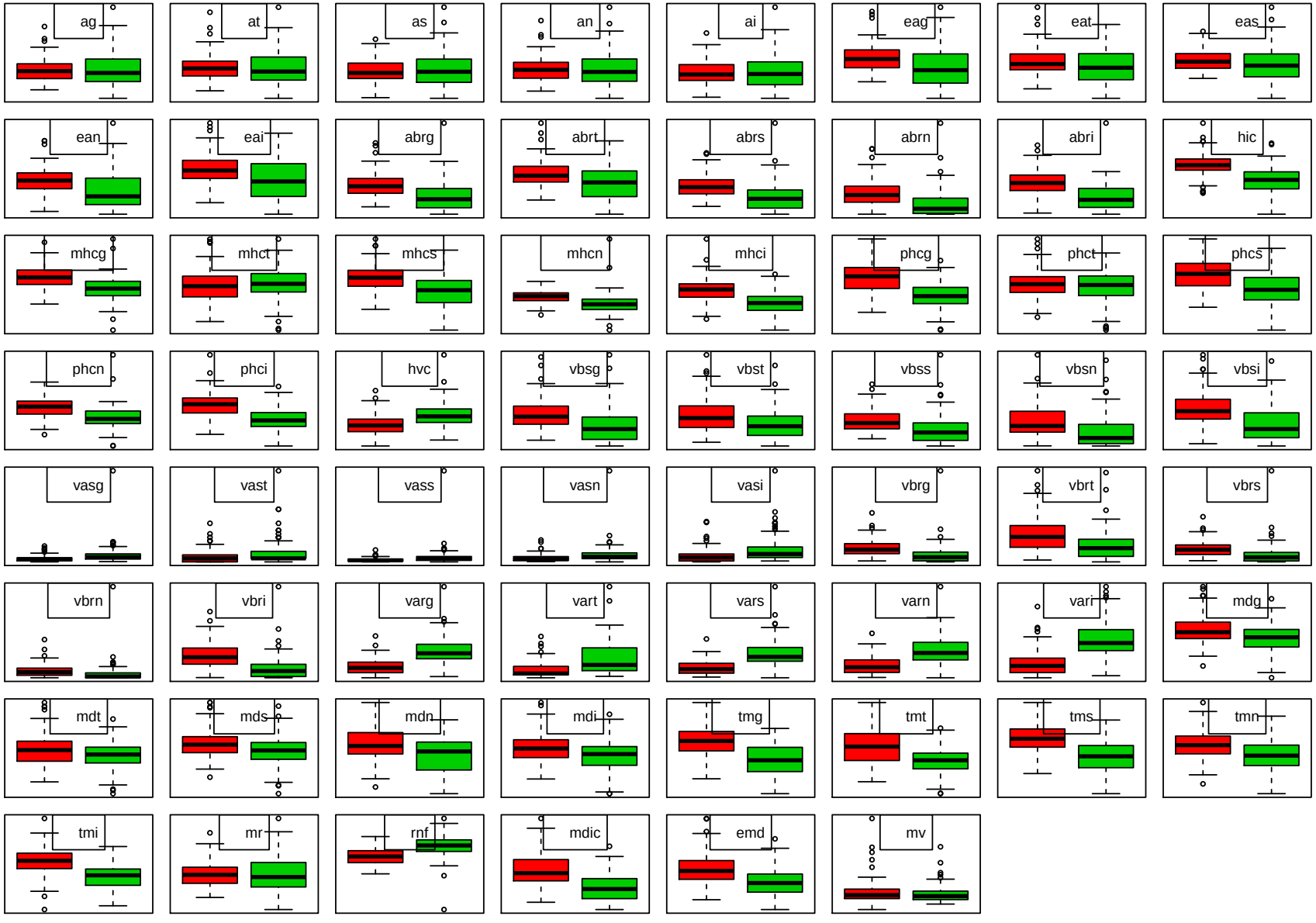
```
dim((GlaucomaM))
```

```
## [1] 196  63
```

```
head(GlaucomaM, n = 2)
```

```
##       ag    at    as    an    ai   eag   eat   eas   ean   eai  abrg  abrt
## 2  2.220 0.354 0.580 0.686 0.601 1.267 0.336 0.346 0.255 0.331 0.479 0.260
## 43 2.681 0.475 0.672 0.868 0.667 2.053 0.440 0.520 0.639 0.454 1.090 0.377
##      abrs  abrn  abri   hic  mhcg  mhct  mhcs  mhcn   mhci   phcg  phct
## 2   0.107 0.014 0.098 0.214 0.111 0.412 0.036 0.105 -0.022 -0.139 0.242
## 43  0.257 0.212 0.245 0.382 0.140 0.338 0.104 0.080  0.109 -0.015 0.296
##       phcs   phcn   phci   hvc  vbsg  vbst  vbss  vbsn  vbsi  vasg  vast
## 2   -0.053  0.010 -0.139 0.613 0.303 0.103 0.088 0.022 0.090 0.062 0.000
## 43 -0.015 -0.015  0.036 0.382 0.676 0.181 0.186 0.141 0.169 0.029 0.001
##      vass  vasn  vasi  vbrg  vbrt  vbrs  vbrn  vbri  varg  vart  vars  varn
## 2   0.011 0.032 0.018 0.075 0.039 0.021 0.002 0.014 0.756 0.009 0.209 0.298
## 43  0.007 0.011 0.010 0.370 0.127 0.099 0.050 0.093 0.410 0.006 0.105 0.181
##      vari   mdg   mdt   mds   mdn   mdi    tmg    tmt    tms    tmn    tmi
## 2   0.240 0.705 0.637 0.738 0.596 0.691 -0.236 -0.018 -0.230 -0.510 -0.158
## 43  0.117 0.898 0.850 0.907 0.771 0.940 -0.211 -0.014 -0.165 -0.317 -0.192
##       mr   rnf  mdic   emd    mv  Class
## 2   0.841 0.410 0.137 0.239 0.035 normal
## 43  0.924 0.256 0.252 0.329 0.022 normal
```

# RP-glaucome-rpart



# RP-glaucoma-rpart

```
data("GlaucomaM", package = "TH.data")
#xval = 100 means 100 runs of 10-fold CV
glaucoma_rpart <- rpart(Class ~ ., data = GlaucomaM,
    control = rpart.control(xval = 100))
glaucoma_rpart$cptable
opt <- which.min(glaucoma_rpart$cptable[,"xerror"])
cp <- glaucoma_rpart$cptable[opt, "CP"]
glaucoma_prune <- prune(glaucoma_rpart, cp = cp)
```

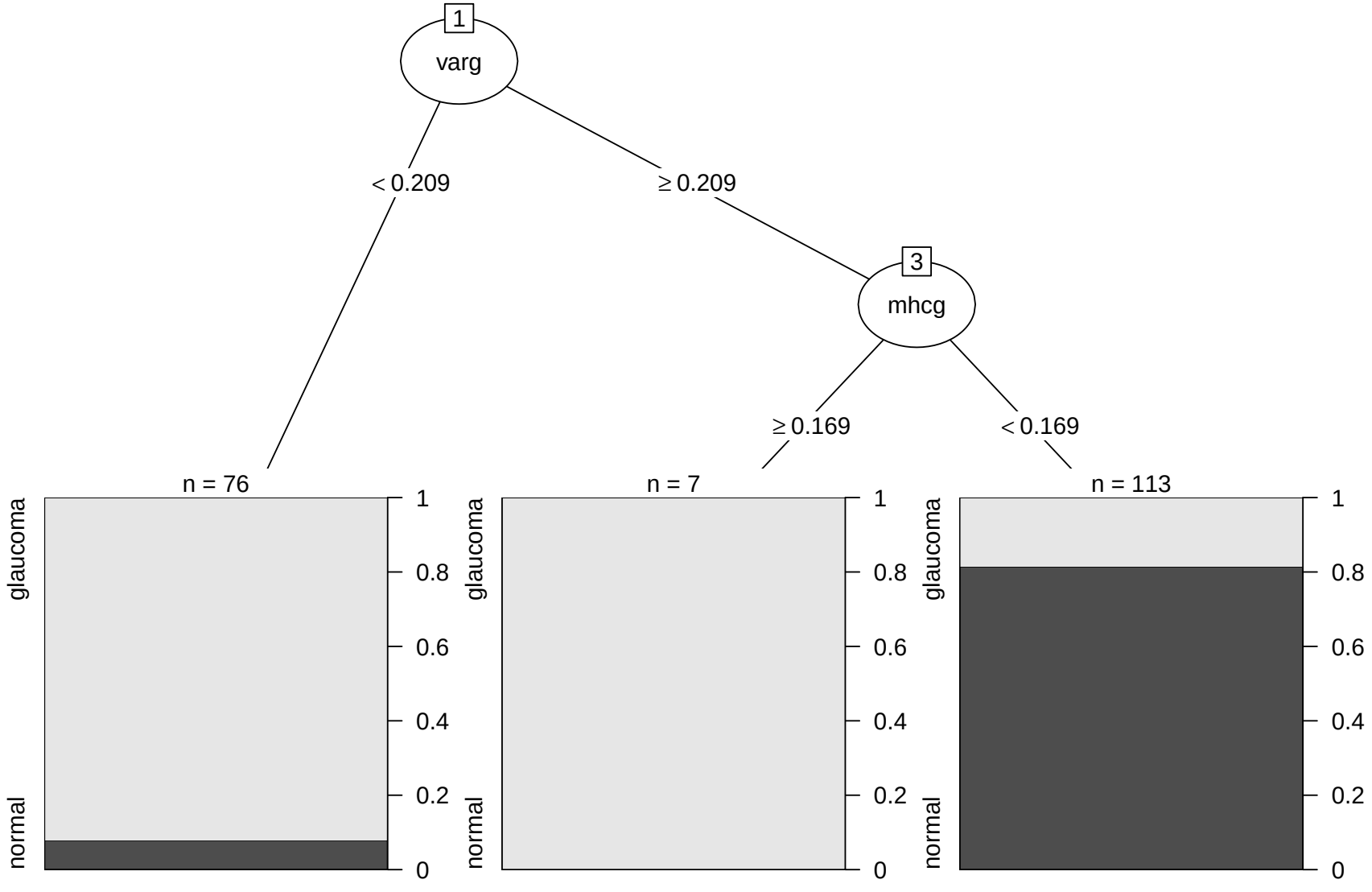
# RP-glaucoma-rpart

```
##           CP nsplit rel error    xerror       xstd
## 1 0.65306122      0 1.0000000 1.5306122 0.06054391
## 2 0.07142857      1 0.3469388 0.3877551 0.05647630
## 3 0.01360544      2 0.2755102 0.3775510 0.05590431
## 4 0.01000000      5 0.2346939 0.4489796 0.05960655
```

# RP-glaucoma-plot

```
plot(as.party(glaucoma_prune), tp_args = list(id = FALSE))
```

# RP-glaucoma-plot



# RP-glaucoma-cp

Unstable result

```
nsplitopt <- vector(mode = "integer", length = 25)
for (i in 1:length(nsplitopt)) {
    cp <- rpart(Class ~ ., data = GlaucomaM)$cptable
    nsplitopt[i] <- cp[which.min(cp[,"xerror"]), "nsplit"]
}
table(nsplitopt)
```

```
## nsplitopt
##  1  2  5
## 14  7  4
```

# RP-glaucoma-bagg

Bagging for GlaucomaM data

```
trees <- vector(mode = "list", length = 25)
n <- nrow(GlaucomaM)
bootsamples <- rmultinom(length(trees), n, rep(1, n)/n)
mod <- rpart(Class ~ ., data = GlaucomaM,
             control = rpart.control(xval = 0))
for (i in 1:length(trees))
    trees[[i]] <- update(mod, weights = bootsamples[,i])
```

# RP-glacoma - bagg

Closer look at the trees

```
table(sapply(trees, function(x)
  as.character(x$frame$var[1])))
```

```
##
## phcg varg vari vars
##    1   14    9    1
```

Note the change in the choice of variables for the root node.

# RP-glaucoma-splits

Predictions for the out-of-bag data (observations not used to build the tree).

```
classprob <- matrix(0, nrow = n, ncol = length(trees))
for (i in 1:length(trees)) {
    classprob[,i] <- predict(trees[[i]],
                             newdata = GlaucomaM)[,1]
    classprob[bootsamples[,i] > 0,i] <- NA
}
```

# RP-glaucoma-avg

Take average to find final predictions

```
avg <- rowMeans(classprob, na.rm = TRUE)
predictions <- factor(ifelse(avg > 0.5, "glaucoma",
                                        "normal"))
predtab <- table(predictions, GlaucomaM$Class)
predtab
```

```
##
## predictions glaucoma normal
##    glaucoma       77     12
##    normal         21     86
```

# RP-glaucoma-sensivity

- Sensitivity= true positives/(true positive + false negative)
- If a person has glaucoma what is the probability of predicting glaucoma?

```
round(predtab[1,1] / colSums(predtab)[1] * 100)
```

```
## glaucoma
##       79
```

# RP-glaucoma-specificity

- Specificity=true negatives/(true negative + false positives)
- If a person does not have glaucoma what is the probability of predicting normal?

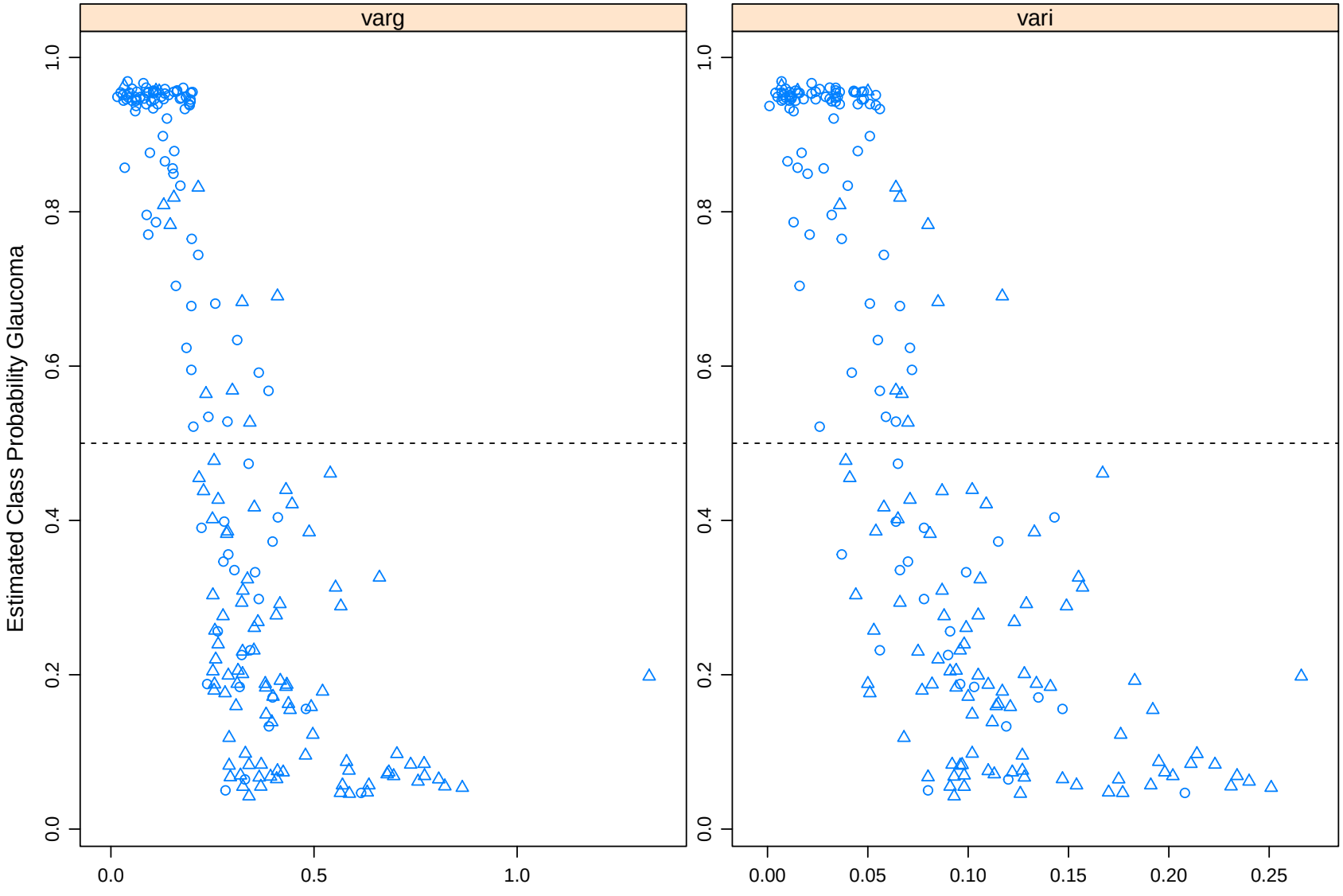
```
round(predtab[2,2] / colSums(predtab)[2] * 100)
```

```
## normal
##     88
```

# RP-glaucoma-baggplot

```
library("lattice")
gdata <- data.frame(avg = rep(avg, 2),
    class = rep(as.numeric(GlaucomaM$Class), 2),
    obs = c(GlaucomaM[["varg"]], GlaucomaM[["vari"]]),
    var = factor(c(rep("varg", nrow(GlaucomaM)),
                   rep("vari", nrow(GlaucomaM)))))
panelf <- function(x, y) {
           panel.xyplot(x, y, pch = gdata$class)
           panel.abline(h = 0.5, lty = 2)
       }
print(xyplot(avg ~ obs | var, data = gdata,
       panel = panelf,
       scales = "free", xlab = "",
       ylab = "Estimated Class Probability Glaucoma"))
```

# RP-glaucoma-baggplot



# Random Forest

randomForest function in r

```
library("randomForest")
rf <- randomForest(Class ~ ., data = GlaucomaM)
table(predict(rf), GlaucomaM$Class)
```

```
##
##              glaucoma normal
##    glaucoma        80     11
##    normal          18     87
```

```
#importance(rf) #importance of the covariates
```