Regression Trees and Classification Trees

Yifei Sun

Contents

```
library(ISLR)
library(mlbench)
library(caret)
library(tidymodels)
library(rpart)
library(rpart.plot)
library(party)
library(partykit)
library(pROC)
```

Regression Trees

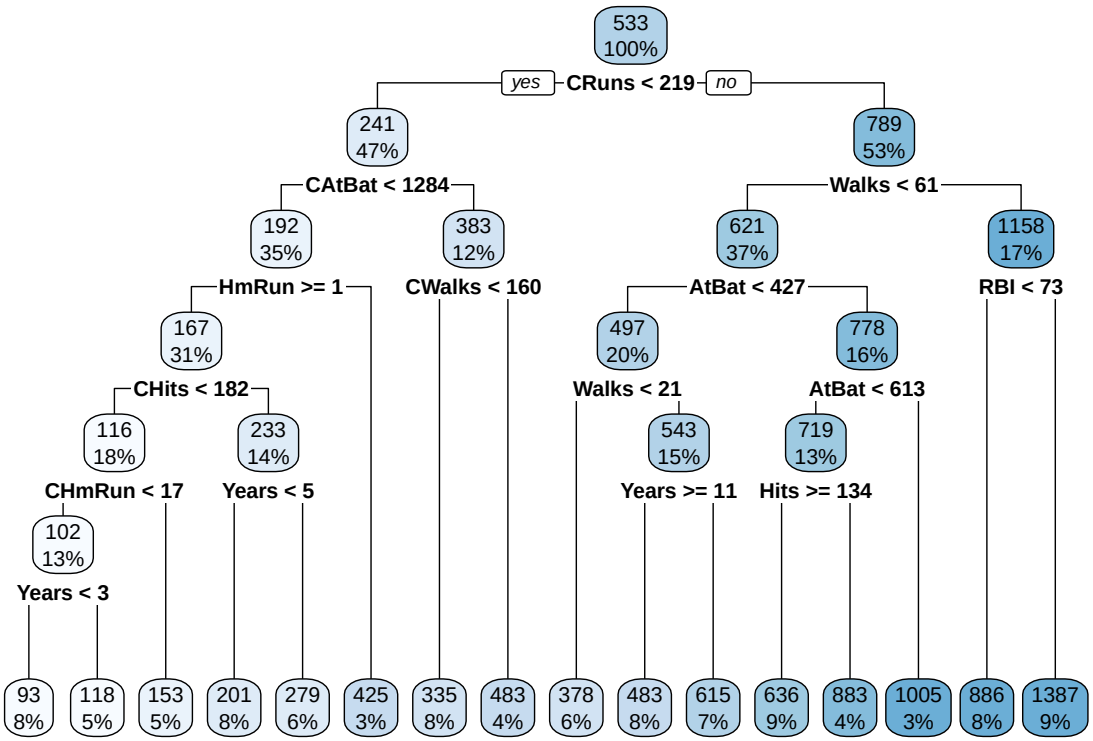
Predict a baseball player's salary on the basis of various statistics associated with performance in the previous year. Use `?Hitters` for more details.

```
data(Hitters)
Hitters <- na.omit(Hitters)

set.seed(2)
data_split <- initial_split(Hitters, prop = 0.8)

# Extract the training and test data
training_data <- training(data_split)
testing_data <- testing(data_split)
```

The CART approach

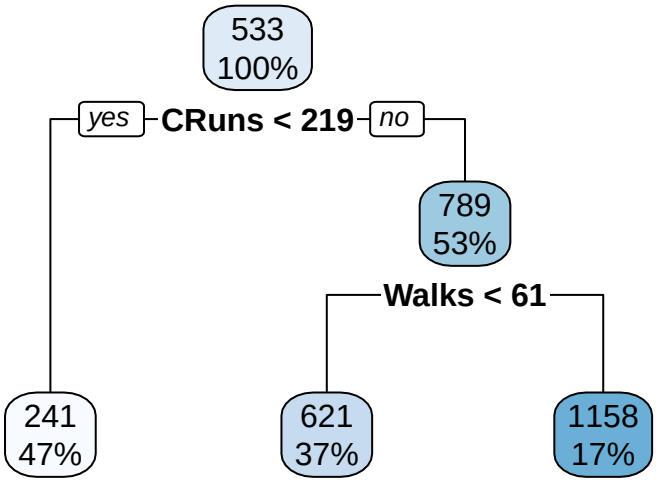
We first apply the regression tree method to the Hitters data. `cp` is the complexity parameter. The default value for `cp` is 0.01. Sometimes the default value may over prune the tree.

```
set.seed(1)
tree1 <- rpart(formula = Salary ~ . ,
               data = training_data,
               control = rpart.control(cp = 0))

## plot.rpart
# plot(tree1)
# text(tree1)

rpart.plot(tree1)
```



We get a smaller tree by increasing the complexity parameter.

```
set.seed(1)
tree2 <- rpart(Salary ~ . ,
               data = training_data,
               control = rpart.control(cp = 0.1))
rpart.plot(tree2)
```



We next apply cost complexity pruning to obtain a tree with the right size. The functions `printcp()` and `plotcp()` give the set of possible cost-complexity prunings of a tree from a nested set. For the geometric means of the intervals of values of `cp` for which a pruning is optimal, a cross-validation has been done in the initial construction by `rpart()`.

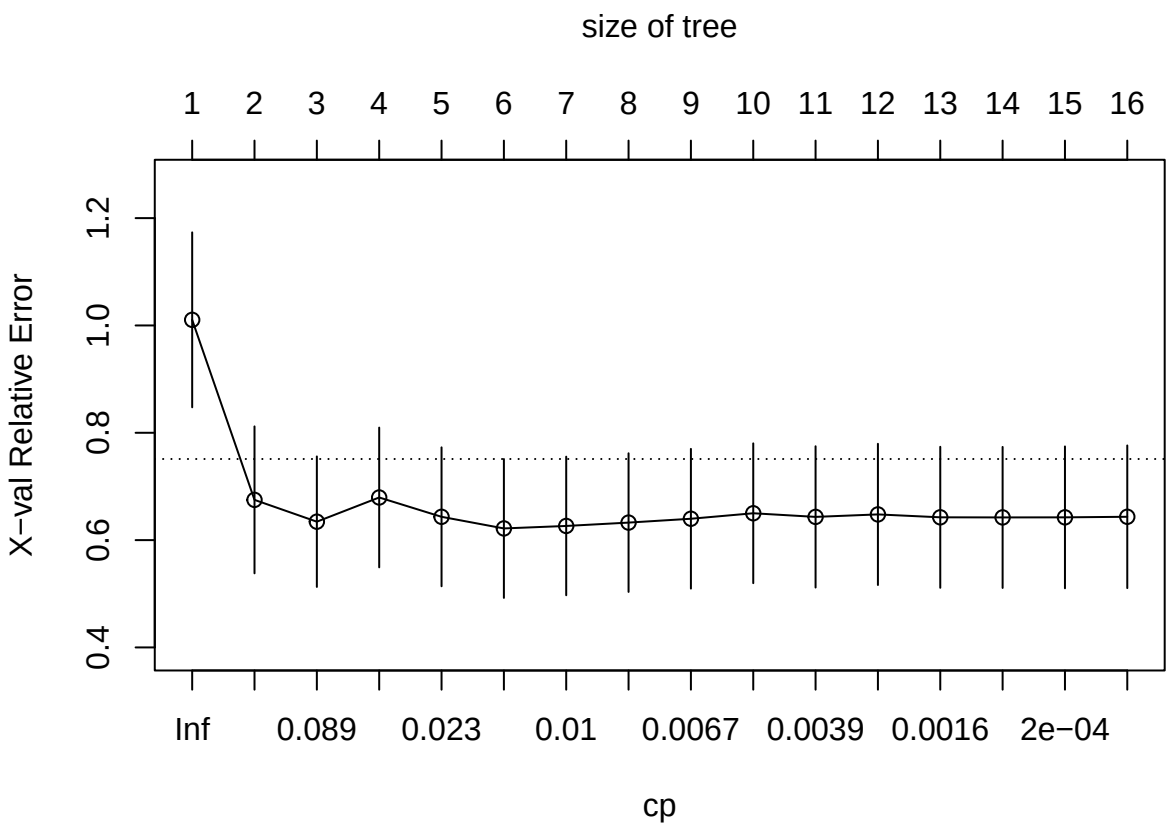
The `cptable` in the fit contains the mean and standard deviation of the errors in the cross-validated prediction against each of the geometric means, and these are plotted by `plotcp()`. `Rel error` (relative error) is `\(1 - R^2\)`. The x-error is the cross-validation error generated by built-in cross validation. A good choice of `cp`

for pruning is often the leftmost value for which the mean lies below the horizontal line.

```
printcp(tree1)
```

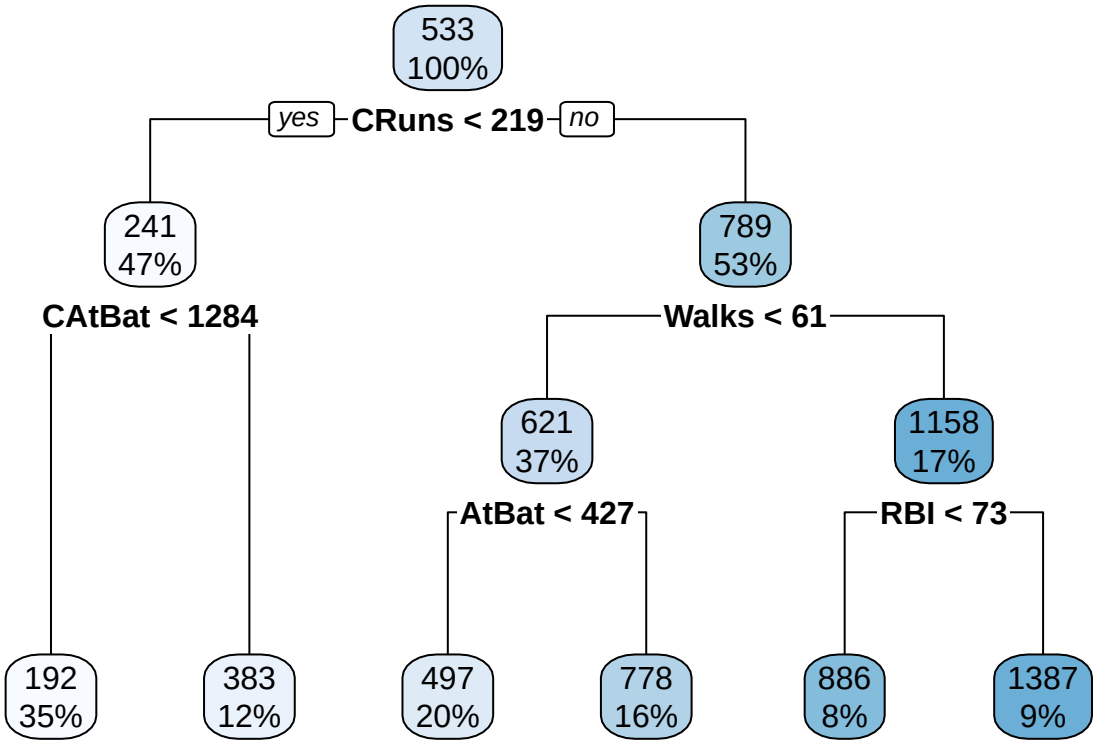
```
##
## Regression tree:
## rpart(formula = Salary ~ ., data = training_data, control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
##  [1] AtBat  CAtBat CHits  CHmRun CRuns  CWalks Hits   HmRun  RBI    Walks
## [11] Years
##
## Root node error: 43935232/210 = 209215
##
## n= 210
##
##            CP nsplit rel error  xerror    xstd
## 1  3.5743e-01      0   1.00000 1.01046 0.16303
## 2  1.5798e-01      1   0.64257 0.67497 0.13692
## 3  4.9700e-02      2   0.48460 0.63434 0.12162
## 4  3.4048e-02      3   0.43490 0.67942 0.13024
## 5  1.5396e-02      4   0.40085 0.64339 0.12945
## 6  1.0392e-02      5   0.38545 0.62172 0.12944
## 7  9.5374e-03      6   0.37506 0.62637 0.12914
## 8  8.2968e-03      7   0.36552 0.63264 0.12919
## 9  5.3687e-03      8   0.35723 0.63980 0.13022
## 10 5.0983e-03      9   0.35186 0.65000 0.13029
## 11 3.0441e-03     10   0.34676 0.64324 0.13165
## 12 2.7021e-03     11   0.34372 0.64787 0.13159
## 13 9.6231e-04     12   0.34101 0.64248 0.13140
## 14 4.3094e-04     13   0.34005 0.64224 0.13141
## 15 9.0096e-05     14   0.33962 0.64240 0.13221
## 16 0.0000e+00     15   0.33953 0.64345 0.13293
```

```
cpTable <- tree1$cptable
plotcp(tree1)
```

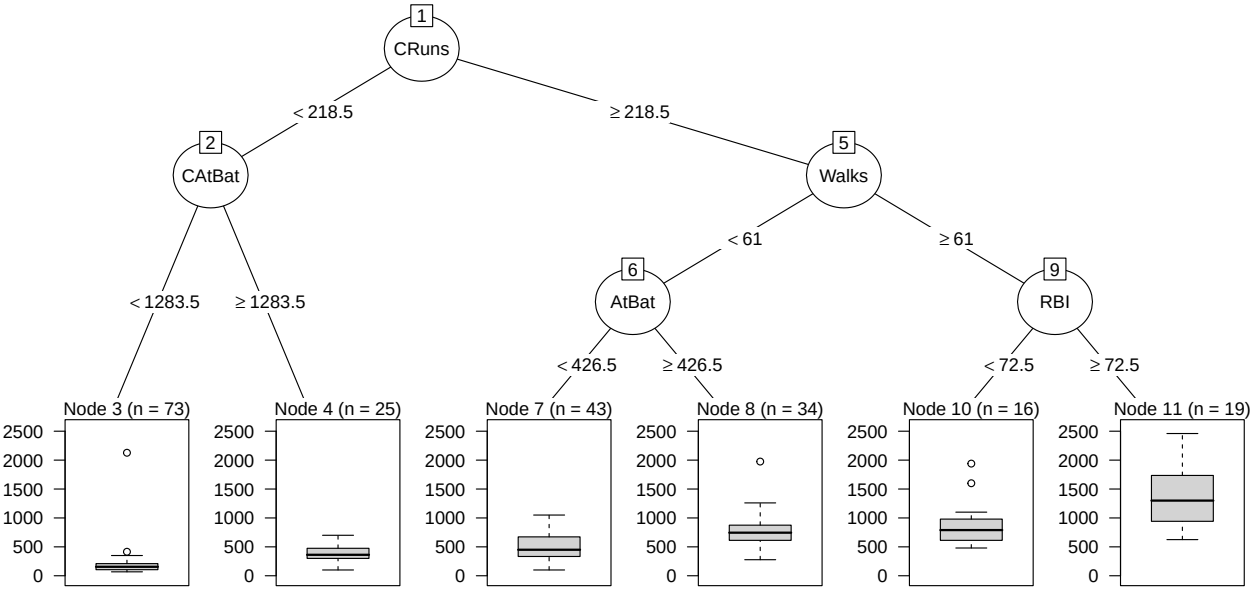


Prune the tree based on the `cp` table.

```
# minimum cross-validation error
minErr <- which.min(cpTable[,4])
tree3 <- rpart::prune(tree1, cp = cpTable[minErr,1])
rpart.plot(tree3)
```




```
plot(as.party(tree3))
```



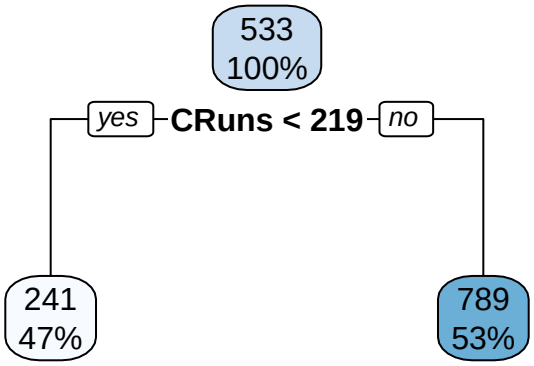

```
summary(tree3)
```

```
## Call:
## rpart(formula = Salary ~ ., data = training_data, control = rpart.control(cp = 0))
##   n= 210
##
##           CP nsplit rel error    xerror      xstd
## 1 0.35742526      0 1.0000000 1.0104572 0.1630347
## 2 0.15797588      1 0.6425747 0.6749728 0.1369186
## 3 0.04970014      2 0.4845989 0.6343359 0.1216197
```

```
## 4 0.03404822      3 0.4348987 0.6794184 0.1302364
## 5 0.01539613      4 0.4008505 0.6433932 0.1294489
## 6 0.01039234      5 0.3854544 0.6217198 0.1294425
##
## Variable importance
##   CRuns   CHits  CAtBat    CRBI  CWalks  CHmRun   Walks     RBI    Runs   HmRun
##      14      14      14      14      12      12       6       4       4       3
##   AtBat PutOuts    Hits
##       2       2       1
##
## Node number 1: 210 observations,    complexity param=0.3574253
##   mean=532.9367, MSE=209215.4
##   left son=2 (98 obs) right son=3 (112 obs)
##   Primary splits:
##       CRuns  < 218.5  to the left,  improve=0.3574253, (0 missing)
##       CHits  < 450    to the left,  improve=0.3534071, (0 missing)
##       CAtBat < 1762.5 to the left,  improve=0.3474006, (0 missing)
##       CRBI   < 307.5  to the left,  improve=0.3398106, (0 missing)
##       CWalks < 222.5  to the left,  improve=0.2990640, (0 missing)
##   Surrogate splits:
##       CHits  < 450    to the left,  agree=0.971, adj=0.939, (0 split)
##       CAtBat < 1762.5 to the left,  agree=0.967, adj=0.929, (0 split)
##       CRBI   < 224    to the left,  agree=0.943, adj=0.878, (0 split)
##       CWalks < 164    to the left,  agree=0.910, adj=0.806, (0 split)
##       CHmRun < 31.5   to the left,  agree=0.848, adj=0.673, (0 split)
##
## Node number 2: 98 observations,    complexity param=0.01539613
##   mean=240.5986, MSE=54395.64
##   left son=4 (73 obs) right son=5 (25 obs)
##   Primary splits:
##       CAtBat < 1283.5 to the left,  improve=0.1268920, (0 missing)
##       CWalks < 117.5  to the left,  improve=0.1253894, (0 missing)
##       CHits  < 285.5  to the left,  improve=0.1138070, (0 missing)
##       CRBI   < 113.5  to the left,  improve=0.1123852, (0 missing)
##       CRuns  < 153    to the left,  improve=0.1054127, (0 missing)
##   Surrogate splits:
##       CHits  < 324    to the left,  agree=0.959, adj=0.84, (0 split)
##       CRuns  < 158    to the left,  agree=0.918, adj=0.68, (0 split)
##       CRBI   < 119.5  to the left,  agree=0.888, adj=0.56, (0 split)
##       CWalks < 120    to the left,  agree=0.878, adj=0.52, (0 split)
##       Years  < 6.5    to the left,  agree=0.786, adj=0.16, (0 split)
##
## Node number 3: 112 observations,    complexity param=0.1579759
##   mean=788.7326, MSE=204472.3
##   left son=6 (77 obs) right son=7 (35 obs)
##   Primary splits:
##       Walks < 61     to the left,  improve=0.3030758, (0 missing)
##       AtBat < 426.5  to the left,  improve=0.2635160, (0 missing)
##       RBI   < 80.5   to the left,  improve=0.2575344, (0 missing)
##       Hits  < 117    to the left,  improve=0.2476573, (0 missing)
##       Runs  < 60.5   to the left,  improve=0.2386230, (0 missing)
##   Surrogate splits:
##       Runs    < 80.5   to the left,  agree=0.777, adj=0.286, (0 split)
##       RBI     < 80.5   to the left,  agree=0.741, adj=0.171, (0 split)
```

```
##       HmRun   < 23.5   to the left,  agree=0.732, adj=0.143, (0 split)
##       CHmRun  < 250    to the left,  agree=0.732, adj=0.143, (0 split)
##       PutOuts < 857.5  to the left,  agree=0.732, adj=0.143, (0 split)
##
## Node number 4: 73 observations
##   mean=191.9795, MSE=57597.49
##
## Node number 5: 25 observations
##   mean=382.5667, MSE=17988.94
##
## Node number 6: 77 observations,    complexity param=0.03404822
##   mean=620.898, MSE=82928.03
##   left son=12 (43 obs) right son=13 (34 obs)
##   Primary splits:
##       AtBat < 426.5  to the left,  improve=0.2342693, (0 missing)
##       Hits  < 103.5  to the left,  improve=0.1946345, (0 missing)
##       Runs  < 60     to the left,  improve=0.1755517, (0 missing)
##       RBI   < 52     to the left,  improve=0.1597126, (0 missing)
##       HmRun < 20.5   to the left,  improve=0.1551018, (0 missing)
##   Surrogate splits:
##       Hits    < 114.5  to the left,  agree=0.948, adj=0.882, (0 split)
##       Runs    < 50.5   to the left,  agree=0.870, adj=0.706, (0 split)
##       RBI     < 58.5   to the left,  agree=0.870, adj=0.706, (0 split)
##       PutOuts < 229.5  to the left,  agree=0.766, adj=0.471, (0 split)
##       HmRun   < 13.5   to the left,  agree=0.753, adj=0.441, (0 split)
##
## Node number 7: 35 observations,    complexity param=0.04970014
##   mean=1157.969, MSE=273563.8
##   left son=14 (16 obs) right son=15 (19 obs)
##   Primary splits:
##       RBI      < 72.5   to the left,  improve=0.2280573, (0 missing)
##       Division splits as  RL,         improve=0.2187075, (0 missing)
##       CHits    < 974    to the left,  improve=0.2033136, (0 missing)
##       CWalks   < 415.5  to the left,  improve=0.2033136, (0 missing)
##       CRBI     < 365.5  to the left,  improve=0.2023335, (0 missing)
##   Surrogate splits:
##       HmRun  < 18.5   to the left,  agree=0.886, adj=0.750, (0 split)
##       CHmRun < 99.5   to the left,  agree=0.829, adj=0.625, (0 split)
##       Runs   < 68.5   to the left,  agree=0.771, adj=0.500, (0 split)
##       CRBI   < 365.5  to the left,  agree=0.743, adj=0.438, (0 split)
##       AtBat  < 527.5  to the left,  agree=0.686, adj=0.313, (0 split)
##
## Node number 12: 43 observations
##   mean=496.9574, MSE=44253.21
##
## Node number 13: 34 observations
##   mean=777.6464, MSE=87842.74
##
## Node number 14: 16 observations
##   mean=885.7812, MSE=146893.5
##
## Node number 15: 19 observations
##   mean=1387.179, MSE=265307.8
```

```
# 1SE rule
tree4 <- rpart::prune(tree1,
                      cp = cpTable[cpTable[,4]<cpTable[minErr,4]+cpTable[minErr,5],1][1])
rpart.plot(tree4)
```



Finally, the function `predict()` can be used for prediction from a fitted `rpart` object.

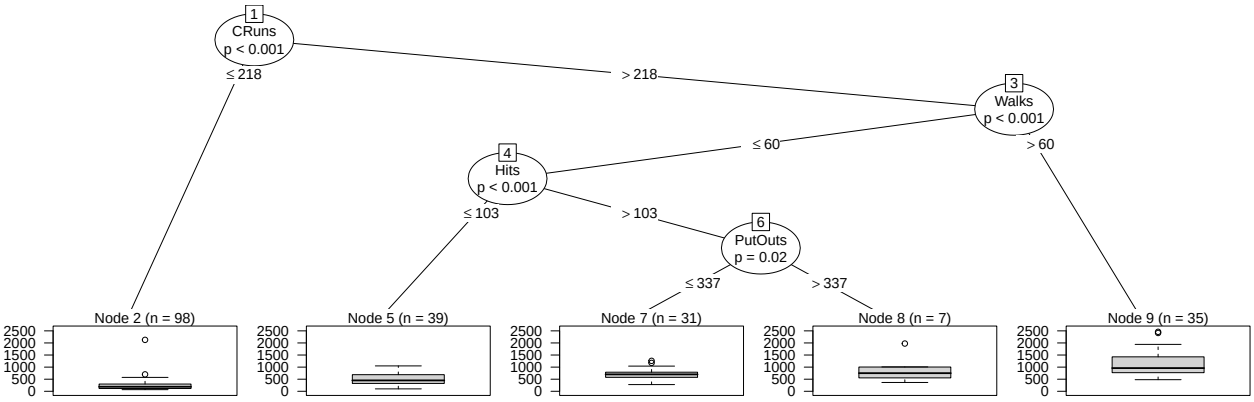
```
head(predict(tree3, newdata = testing_data))
```

```
## -Andres Galarraga  -Alfredo Griffin     -Bill Buckner          -Bo Diaz
##          191.9795          777.6464          777.6464          777.6464
##     -Bill Madlock       -Bob Melvin
##          496.9574          191.9795
```

Conditional inference trees

The implementation utilizes a unified framework for conditional inference, or permutation tests. Unlike CART, the stopping criterion is based on p-values. A split is implemented when (1 - p-value) exceeds the value given by `mincriterion` as specified in `ctree_control()`. This approach ensures that the right-sized tree is grown without additional pruning or cross-validation, but can stop early. At each step, the splitting variable is selected as the input variable with strongest association to the response (measured by a p-value corresponding to a test for the partial null hypothesis of a single input variable and the response). Such a splitting procedure can avoid a variable selection bias towards predictors with many possible cutpoints.

```
tree5 <- ctree(Salary ~ . ,
               training_data)
plot(tree5)
```



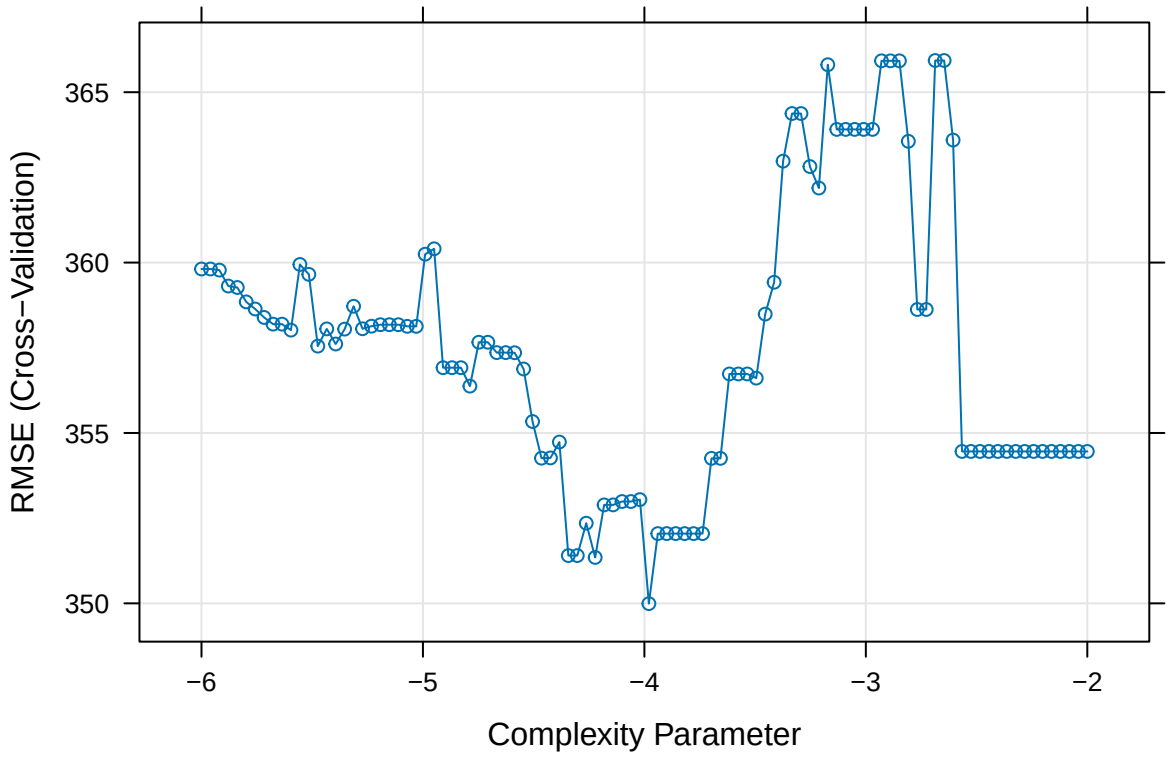
Note that `tree5` is a `party` object. The function `predict()` can be used for prediction from a fitted `party` object.

```
head(predict(tree5, newdata = testing_data))
```

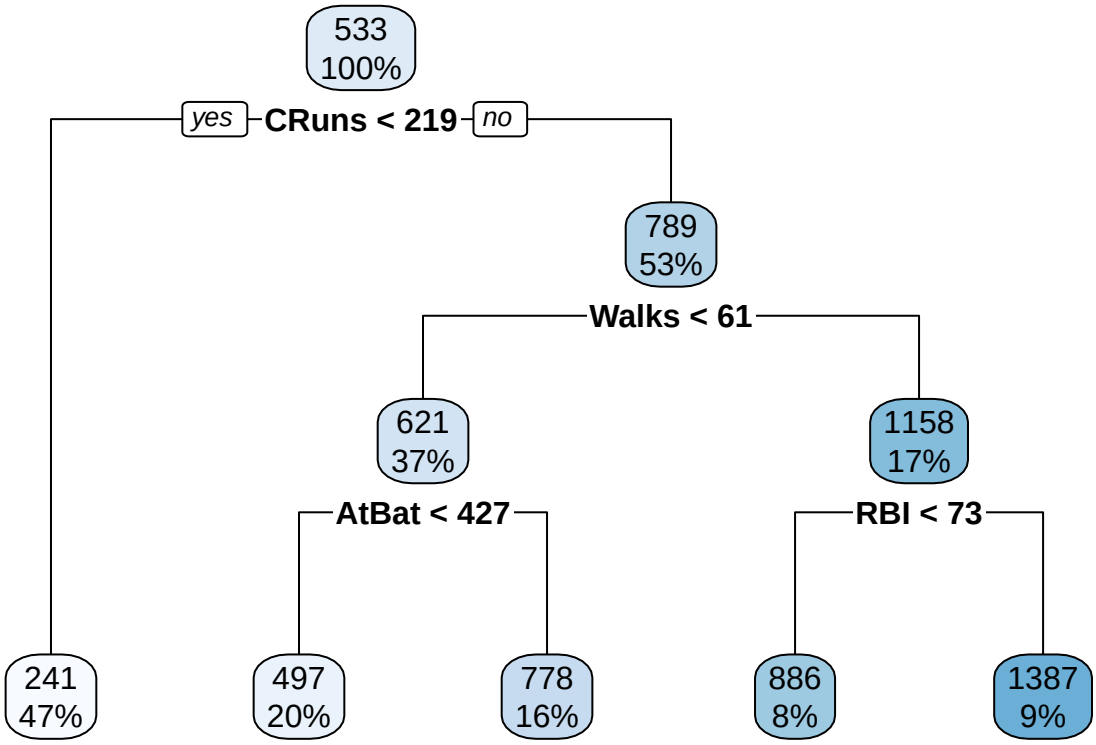
```
## -Andres Galarraga  -Alfredo Griffin     -Bill Buckner          -Bo Diaz
##          240.5986          718.6014          886.9047          886.9047
##     -Bill Madlock       -Bob Melvin
##          718.6014          240.5986
```

caret

```
ctrl <- trainControl(method = "cv")

set.seed(1)
rpart.fit <- train(Salary ~ . ,
                   training_data,
                   method = "rpart",
                   tuneGrid = data.frame(cp = exp(seq(-6,-2, length = 100))),
                   trControl = ctrl)
plot(rpart.fit, xTrans = log)
```

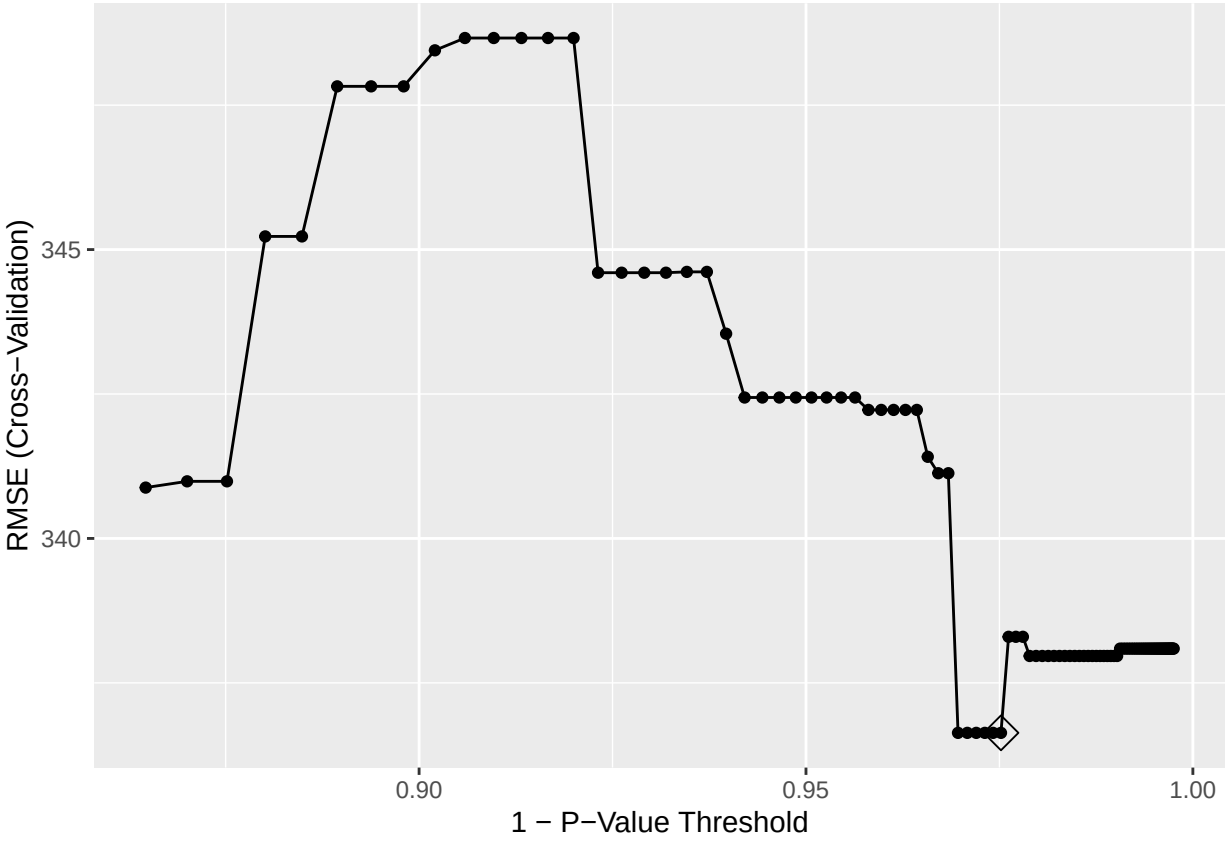



```
rpart.plot(rpart.fit$finalModel)
```

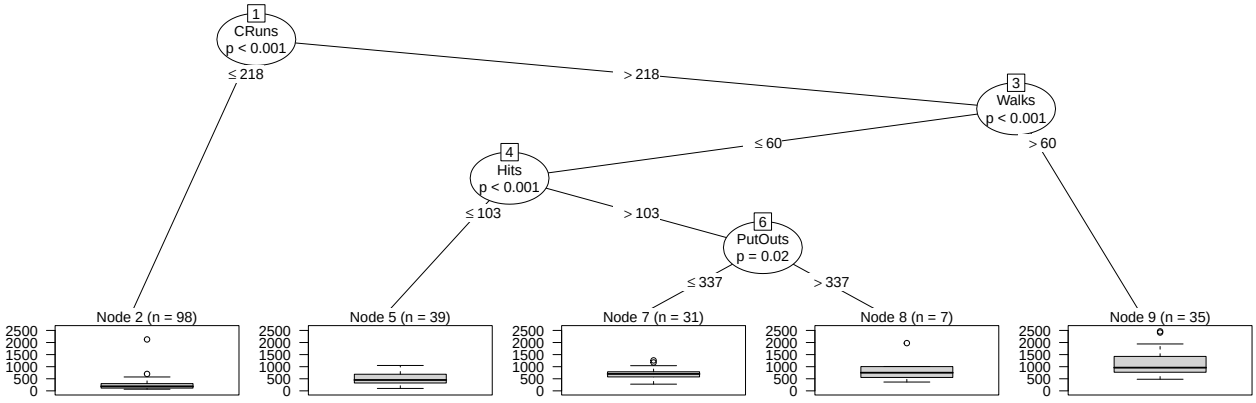


We can also fit a conditional inference tree model. The tuning parameter is `mincriterion`.

```
set.seed(1)
ctree.fit <- train(Salary ~ . ,
                   training_data,
                   method = "ctree",
                   tuneGrid = data.frame(mincriterion = 1-exp(seq(-6, -2, length = 100))),
                   trControl = ctrl)
ggplot(ctree.fit, highlight = TRUE)
```




```
plot(ctree.fit$finalModel)
```




```
summary(resamples(list(rpart.fit, ctree.fit)))
```

```
##
## Call:
## summary.resamples(object = resamples(list(rpart.fit, ctree.fit)))
##
## Models: Model1, Model2
## Number of resamples: 10
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## Model1 173.1019 195.1238 225.3180 233.7376 251.4178 364.8438    0
## Model2 154.0550 186.0376 216.0313 221.5436 244.1977 343.5896    0
```

```
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## Model1 227.4213 288.7912 338.4415 349.9922 360.9544 573.8640    0
## Model2 204.0065 264.6927 326.1890 336.6350 355.2773 567.4219    0
##
## Rsquared
##              Min.   1st Qu.   Median      Mean   3rd Qu.      Max. NA's
## Model1 0.08907984 0.3716446 0.458554 0.4507650 0.6062769 0.7054867    0
## Model2 0.06025527 0.3946665 0.561469 0.4850654 0.6411694 0.6527808    0
```

```
RMSE(predict(rpart.fit, newdata = testing_data), testing_data$Salary)
```

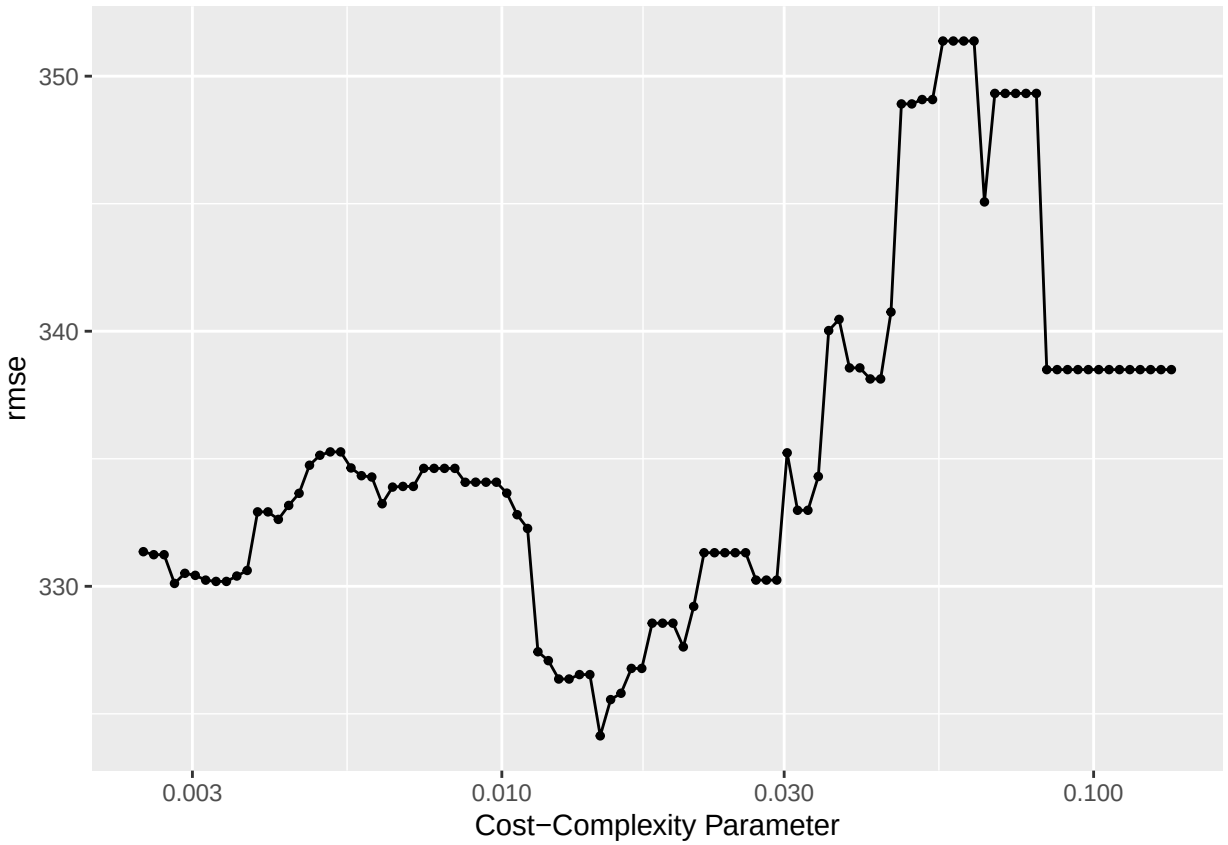
```
## [1] 340.4501
```

```
RMSE(predict(ctree.fit, newdata = testing_data), testing_data$Salary)
```

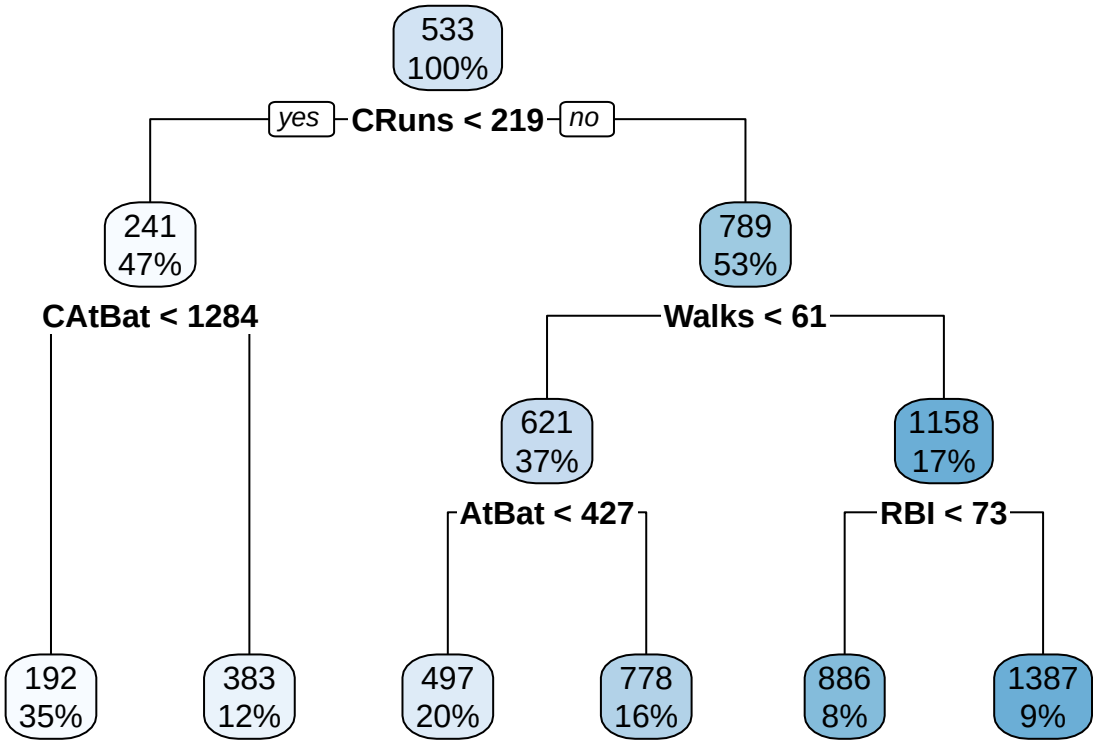
```
## [1] 345.2822
```

tidymodels

```
set.seed(2)
cv_folds <- vfold_cv(training_data, v = 10)

# Model specification
rpart_spec <- decision_tree(cost_complexity = tune(), tree_depth = 30, min_n = 20) %>%
  set_engine("rpart") %>%
  set_mode("regression")

# Tuning grid
rpart_grid_set <- dials::parameters(cost_complexity(range = c(-6, -2), trans = log_trans()))
rpart_grid <- grid_regular(rpart_grid_set, levels = c(100))

# Set up the workflow
rpart_workflow <- workflow() %>%
  add_model(rpart_spec) %>%
  add_formula(Salary ~ .)

rpart_tune <- rpart_workflow %>%
  tune_grid(resamples = cv_folds,
            grid = rpart_grid)

autoplot(rpart_tune, metric = "rmse")
```




```
rpart_best <- select_best(rpart_tune, metric = "rmse")

# Update the model spec
final_rpart_spec <- rpart_spec %>%
  update(cost_complexity = rpart_best$cost_complexity)

rpart_fit <- parsnip::fit(final_rpart_spec, formula = Salary ~ ., data = training_data)
rpart.plot(rpart_fit$fit, roundint = FALSE)
```



Classification trees

We use the Pima Indians Diabetes Database for illustration. The data contain 768 observations and 9 variables. The outcome is a binary variable `diabetes`.

```
data(PimaIndiansDiabetes2)
dat <- PimaIndiansDiabetes2
# dat$diabetes <- factor(dat$diabetes, c("pos", "neg"))

dat <- na.omit(PimaIndiansDiabetes2)

set.seed(1)
data_split <- initial_split(dat, prop = 0.7)

# Extract the training and test data
training_data <- training(data_split)
testing_data <- testing(data_split)
```

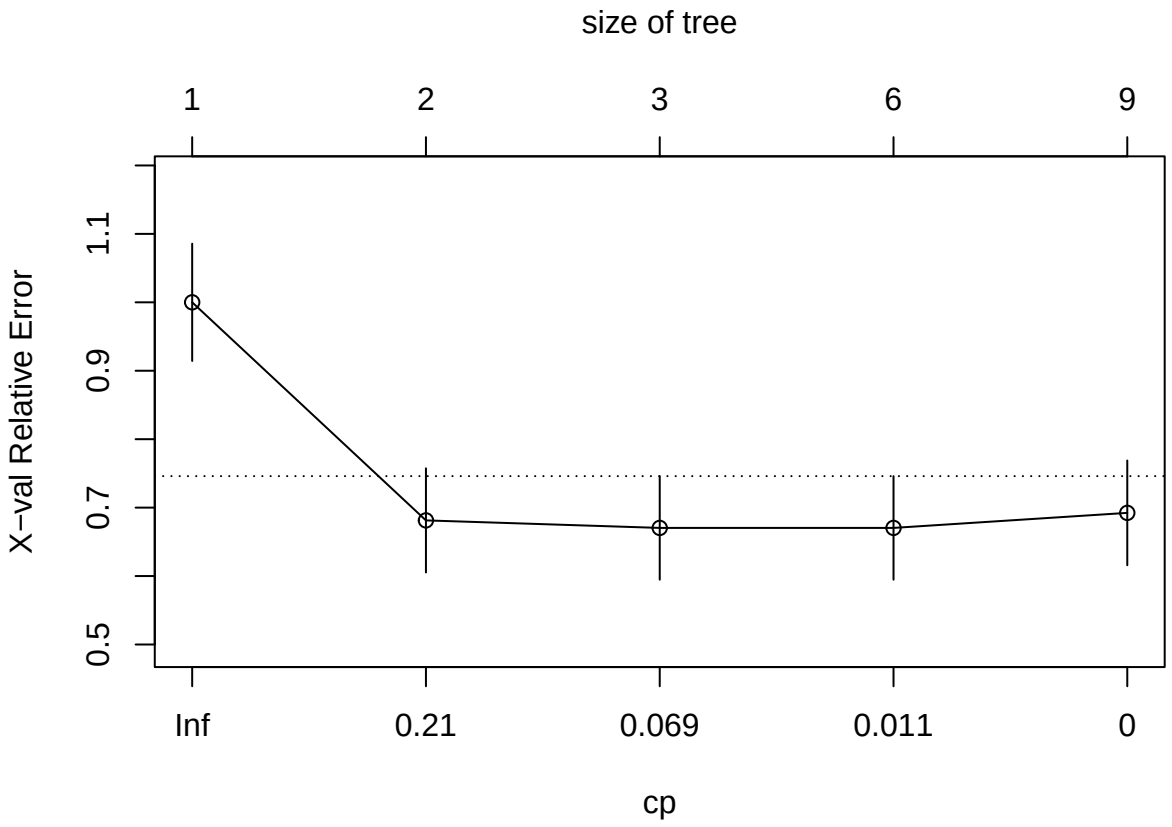
rpart

```
set.seed(1)
tree1 <- rpart(formula = diabetes ~ . ,
               data = training_data,
               control = rpart.control(cp = 0))

cpTable <- printcp(tree1)
```

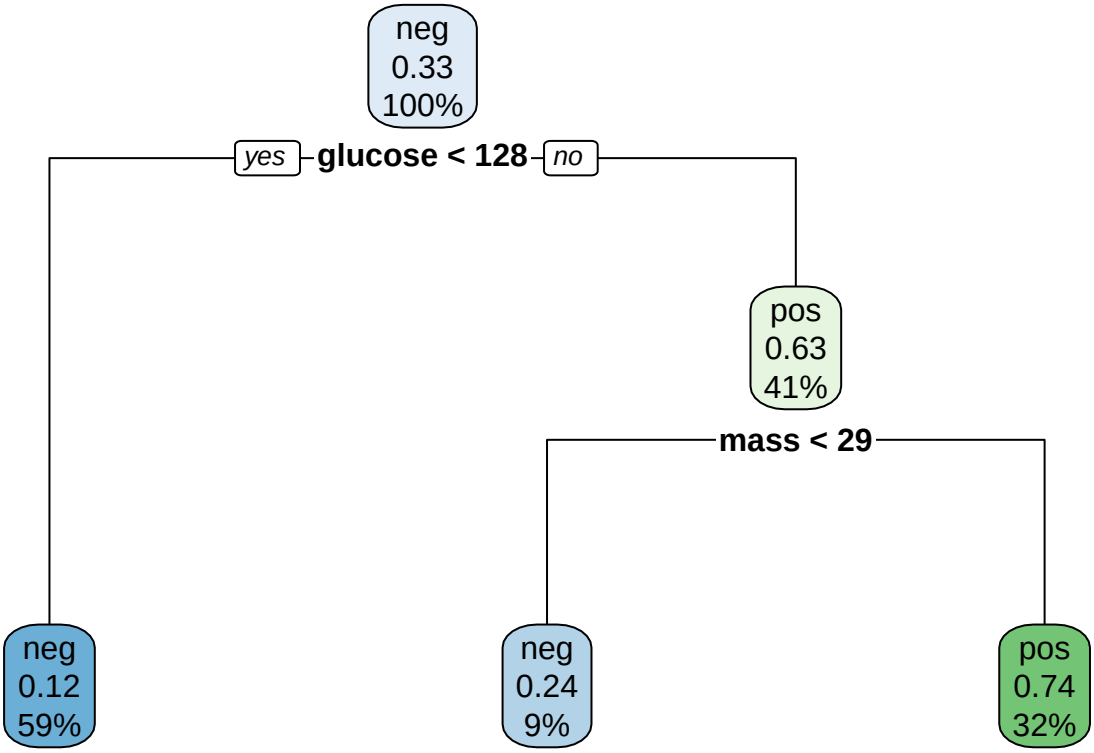
```
##
## Classification tree:
```

```
## rpart(formula = diabetes ~ ., data = training_data, control = rpart.control(cp = 0))
##
## Variables actually used in tree construction:
## [1] age      glucose  mass     pedigree
##
## Root node error: 91/274 = 0.33212
##
## n= 274
##
##         CP nsplit rel error  xerror     xstd
## 1 0.318681      0   1.00000 1.00000 0.085670
## 2 0.142857      1   0.68132 0.68132 0.076111
## 3 0.032967      2   0.53846 0.67033 0.075672
## 4 0.003663      5   0.43956 0.67033 0.075672
## 5 0.000000      8   0.42857 0.69231 0.076541
```

```
plotcp(tree1)
```



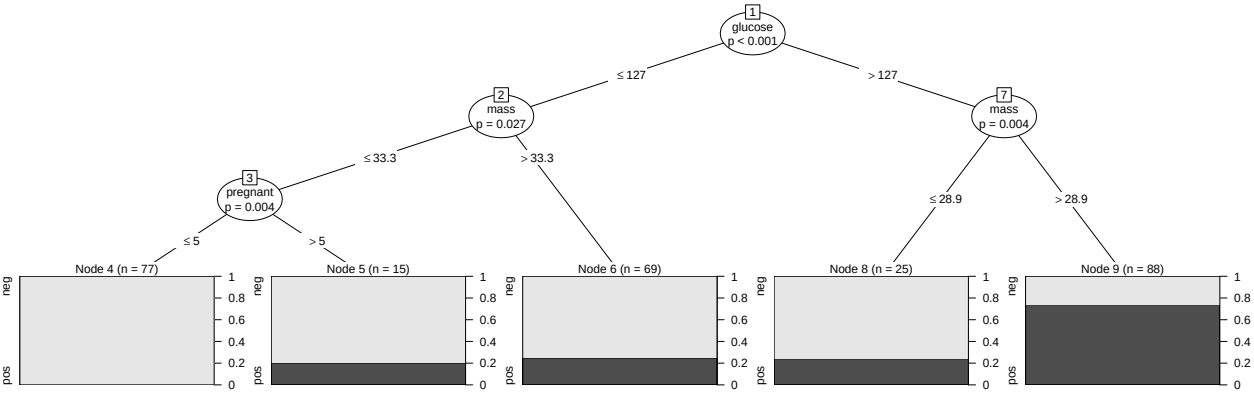

```
# minimum cross-validation error; may also use the 1SE rule
minErr <- which.min(cpTable[,4])
tree2 <- rpart::prune(tree1, cp = cpTable[minErr,1])
rpart.plot(tree2)
```




```
# summary(tree2)
```

ctree

```
tree2 <- ctree(formula = diabetes ~ . ,
               data = training_data)
plot(tree2)
```

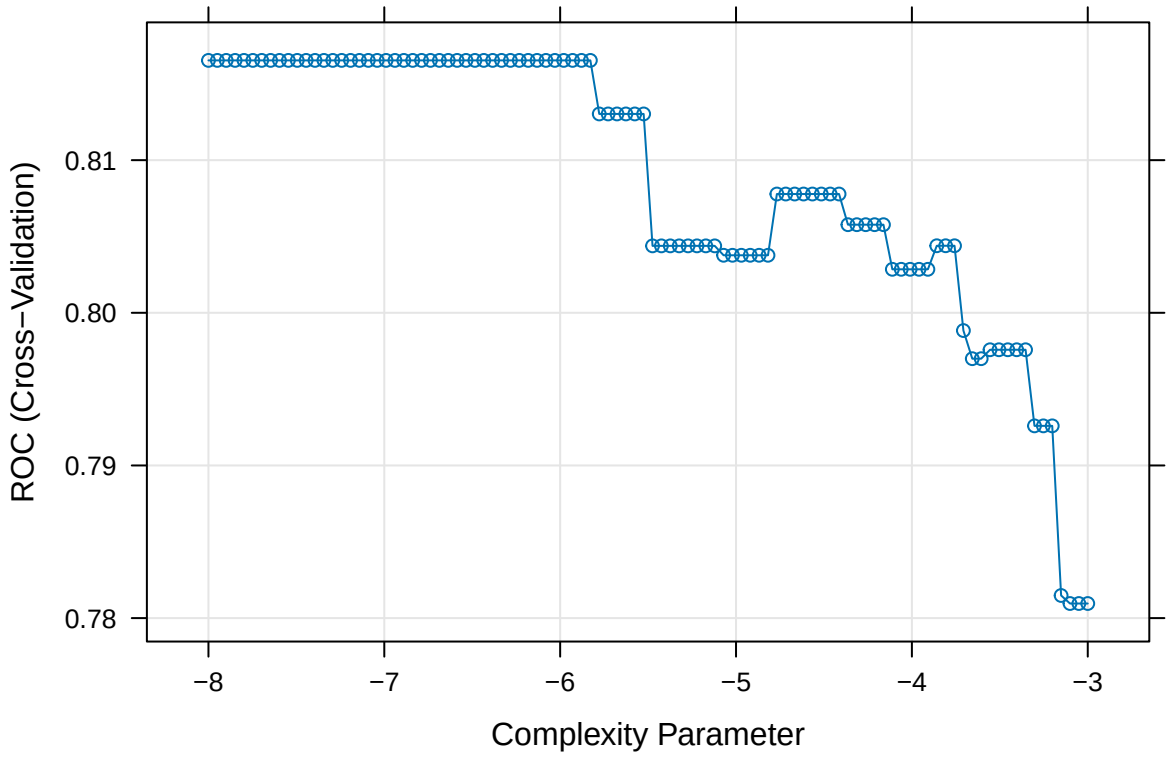


caret

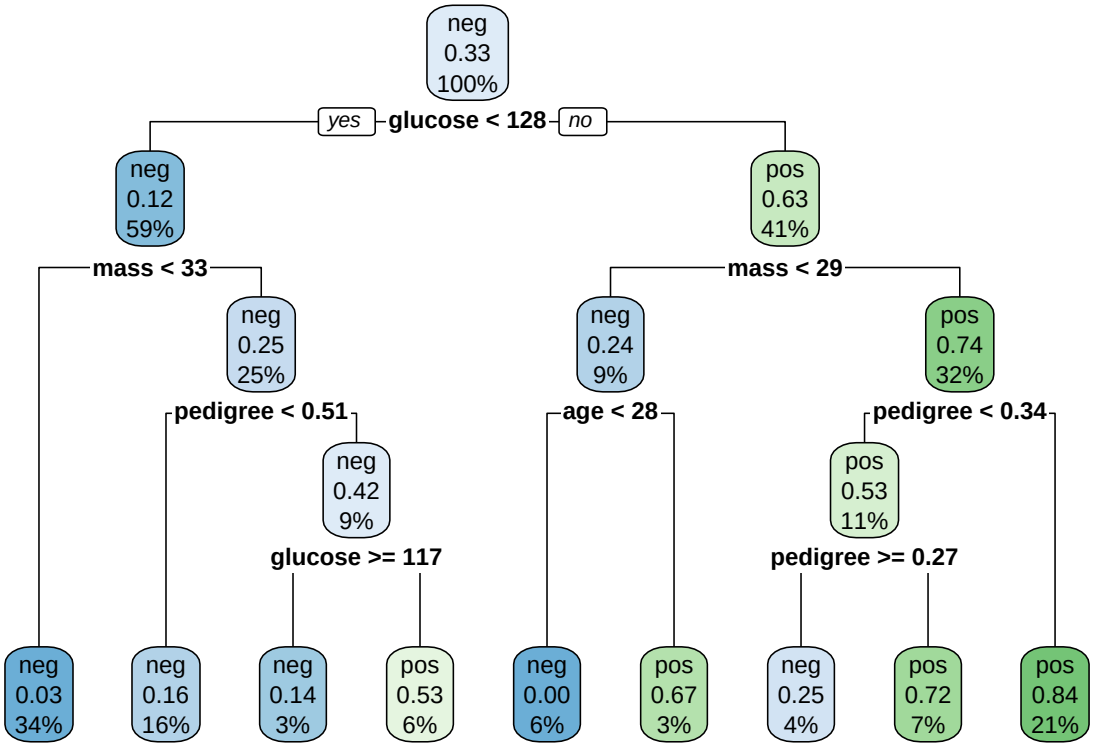
CART

```
ctrl <- trainControl(method = "cv",
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)
```

```
set.seed(1)
rpart.fit <- train(diabetes ~ . ,
                   training_data,
                   method = "rpart",
                   tuneGrid = data.frame(cp = exp(seq(-8,-3, len = 100))),
                   trControl = ctrl,
                   metric = "ROC")
plot(rpart.fit, xTrans = log)
```

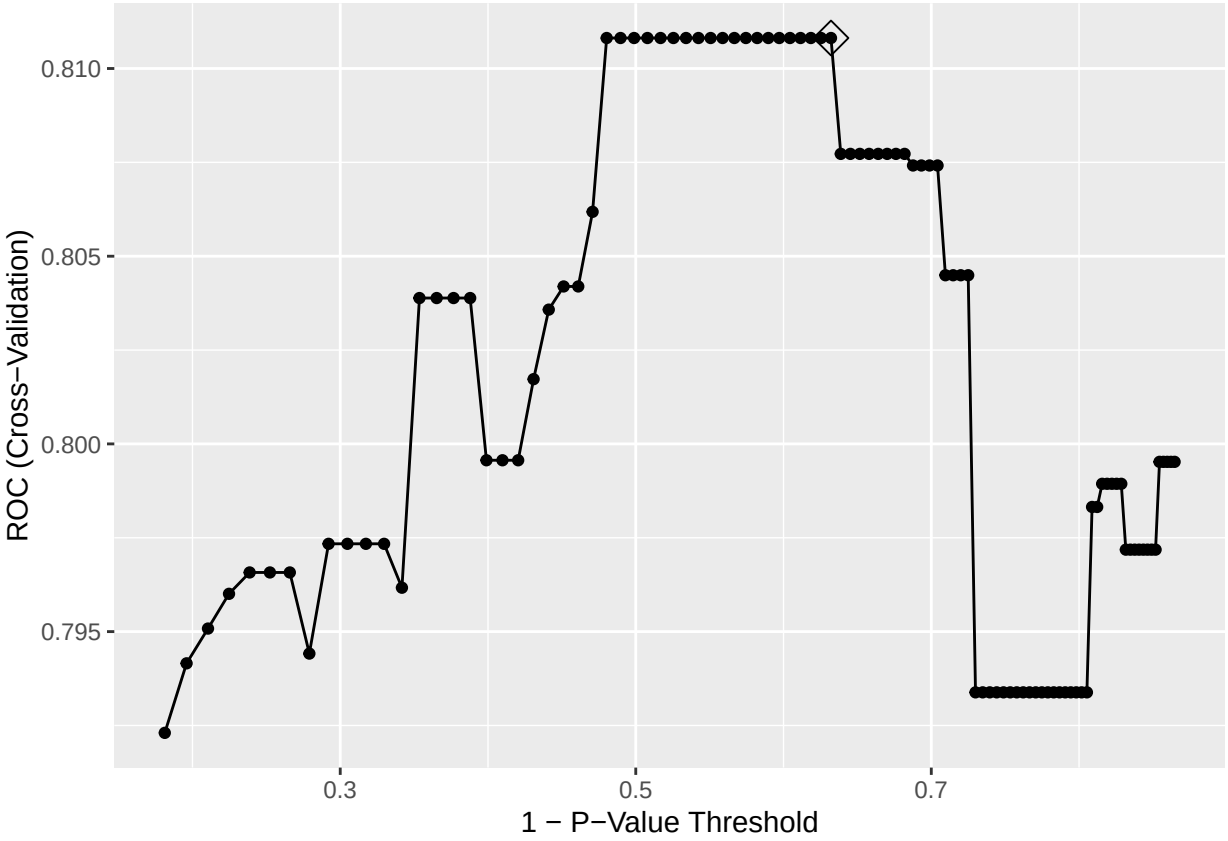



```
rpart.plot(rpart.fit$finalModel)
```

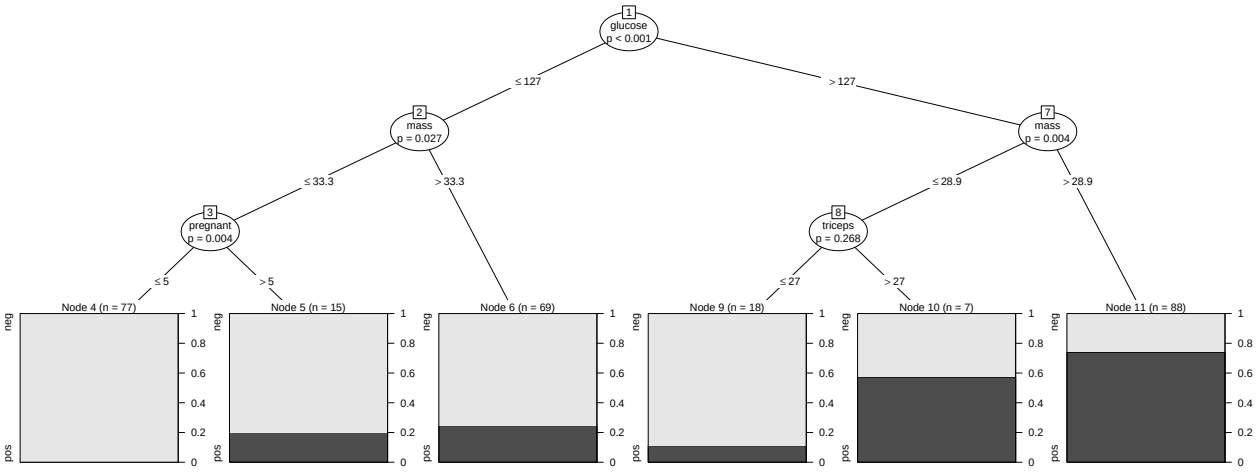


CIT

```
set.seed(1)
ctree.fit <- train(diabetes ~ . ,
                   training_data,
                   method = "ctree",
                   tuneGrid = data.frame(mincriterion = 1-exp(seq(-2, -0.2, length = 100))),
                   metric = "ROC",
                   trControl = ctrl)
ggplot(ctree.fit, highlight = TRUE)
```




```
plot(ctree.fit$finalModel)
```

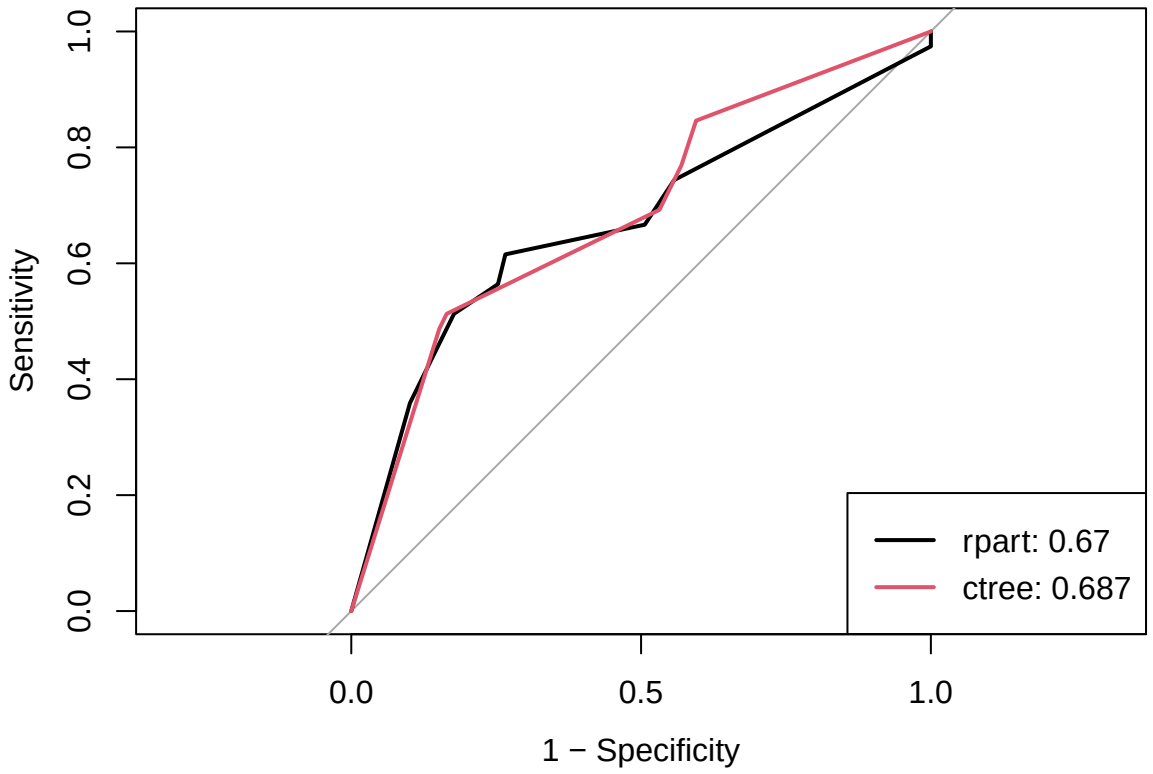



```
summary(resamples(list(rpart.fit, ctree.fit)))
```

```
##
## Call:
## summary.resamples(object = resamples(list(rpart.fit, ctree.fit)))
##
## Models: Model1, Model2
## Number of resamples: 10
##
## ROC
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
```

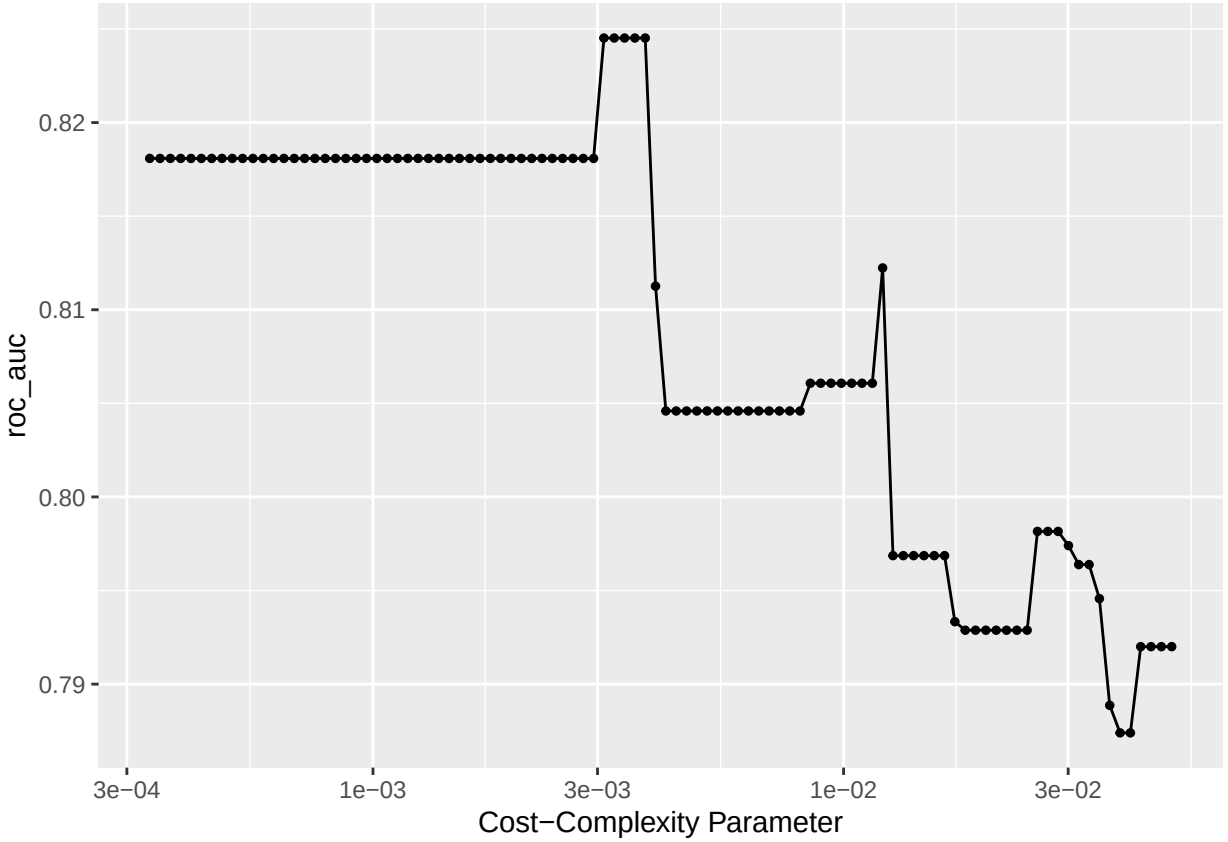
```
## Model1 0.7037037 0.7600309 0.8130279 0.8165367 0.8748782 0.9351852    0
## Model2 0.7160494 0.7808642 0.7952891 0.8108122 0.8530702 0.9012346    0
##
## Sens
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## Model1 0.6842105 0.7361111 0.8114035 0.8257310 0.8932749 1.0000000    0
## Model2 0.7222222 0.7894737 0.8391813 0.8479532 0.9305556 0.9444444    0
##
## Spec
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## Model1 0.3333333 0.5833333 0.7388889 0.6811111 0.7777778 0.8888889    0
## Model2 0.4444444 0.5555556 0.6666667 0.6811111 0.7583333 1.0000000    0
```

```
rpart.pred <- predict(tree1, newdata = testing_data)[,1]

rpart.pred2 <- predict(rpart.fit, newdata = testing_data,
                       type = "prob")[,1]

ctree.pred <- predict(ctree.fit, newdata = testing_data,
                       type = "prob")[,1]

roc.rpart <- roc(testing_data$diabetes, rpart.pred2)
roc.ctree <- roc(testing_data$diabetes, ctree.pred)

auc <- c(roc.rpart$auc[1], roc.ctree$auc[1])

plot(roc.rpart, legacy.axes = TRUE)
plot(roc.ctree, col = 2, add = TRUE)


modelNames <- c("rpart","ctree")
legend("bottomright", legend = paste0(modelNames, ": ", round(auc,3)),
       col = 1:2, lwd = 2)
```

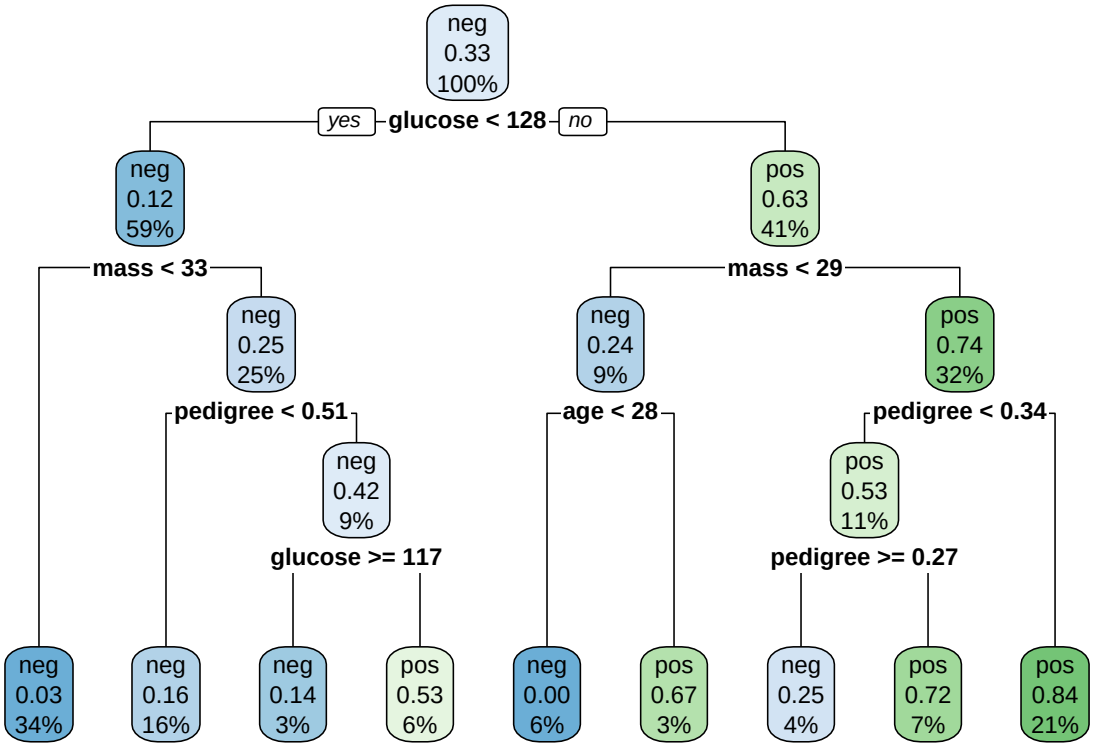


tidymodels

```
set.seed(2)
cv_folds <- vfold_cv(training_data, v = 10)

# Model specification
rpart_spec <- decision_tree(cost_complexity = tune(), tree_depth = 30, min_n = 20) %>%
  set_engine("rpart") %>%
  set_mode("classification")

# Tuning grid
rpart_grid_set <- dials::parameters(cost_complexity(range = c(-8, -3), trans = log_trans()))
rpart_grid <- grid_regular(rpart_grid_set, levels = c(100))

# Set up the workflow
rpart_workflow <- workflow() %>%
  add_model(rpart_spec) %>%
  add_formula(diabetes ~ .)

rpart_tune <- rpart_workflow %>%
  tune_grid(resamples = cv_folds,
            grid = rpart_grid)

autoplot(rpart_tune, metric = "roc_auc")
```




```r
rpart_best <- select_best(rpart_tune, metric = "roc_auc")

# Update the model spec
final_rpart_spec <- rpart_spec %>%
  update(cost_complexity = rpart_best$cost_complexity)

rpart_fit <- parsnip::fit(final_rpart_spec, formula = diabetes ~ ., data = training_data)
rpart.plot(rpart_fit$fit, roundint = FALSE)
```


neg
0.33
100%
yes
glucose < 128
no
neg
0.12
59%
pos
0.63
41%
mass < 33
mass < 29
neg
0.25
25%
neg
0.24
9%
pos
0.74
32%
pedigree < 0.51
age < 28
pedigree < 0.34
neg
0.42
9%
pos
0.53
11%
glucose >= 117
pedigree >= 0.27
neg
0.03
34%
neg
0.16
16%
neg
0.14
3%
pos
0.53
6%
neg
0.00
6%
pos
0.67
3%
neg
0.25
4%
pos
0.72
7%
pos
0.84
21%