Regression Trees

Langtao Chen

Initial: March 1, 2019 Update: April 6, 2021

Contents

1. Data **2**

2. Split Data into Training and Test Sets **4**

3. Train A Regression Tree **4**

4. Test Performance of the Regression Tree **8**

1. Data

As an example, let's use the CPU dataset to demonstrate how to implement regression tree.

```
library(MASS)
str(cpus)
```

```
## 'data.frame':    209 obs. of  9 variables:
##  $ name   : Factor w/ 209 levels "ADVISOR 32/60",..: 1 3 2 4 5 6 8 9 10 7 ...
##  $ syct   : int  125 29 29 29 29 26 23 23 23 23 ...
##  $ mmin   : int  256 8000 8000 8000 8000 8000 16000 16000 16000 32000 ...
##  $ mmax   : int  6000 32000 32000 32000 16000 32000 32000 32000 64000 64000 ...
##  $ cach   : int  256 32 32 32 32 64 64 64 64 128 ...
##  $ chmin  : int  16 8 8 8 8 8 16 16 16 32 ...
##  $ chmax  : int  128 32 32 32 16 32 32 32 32 64 ...
##  $ perf   : int  198 269 220 172 132 318 367 489 636 1144 ...
##  $ estperf: int  199 253 253 253 132 290 381 381 749 1238 ...
```

The cpus dataset contains 209 CPUs of 9 variables including:

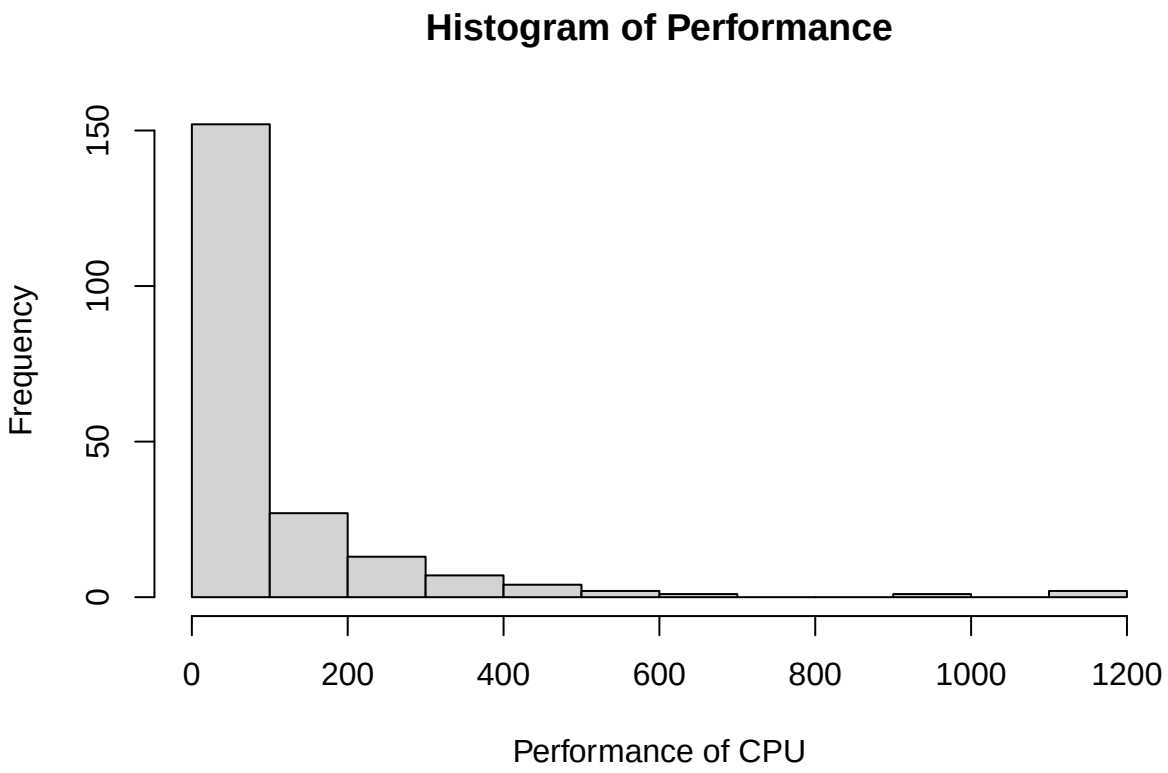
- name: manufacturer and model.
- syct: cycle time in nanoseconds.
- mmin: minimum main memory in kilobytes.
- mmax: maximum main memory in kilobytes.
- cach: cache size in kilobytes.
- chmin: minimum number of channels.
- chmax: maximum number of channels.
- perf: published performance on a benchmark mix relative to an IBM 370/158-3.
- estperf: estimated performance (by Ein-Dor & Feldmesser).

```
summary(cpus)
```

```
##              name          syct             mmin            mmax
##  ADVISOR 32/60 :  1   Min.   :  17.0   Min.   :   64   Min.   :   64
##  AMDAHL 470/7A :  1   1st Qu.:  50.0   1st Qu.:  768   1st Qu.: 4000
##  AMDAHL 470V/7 :  1   Median : 110.0   Median : 2000   Median : 8000
##  AMDAHL 470V/7B:  1   Mean   : 203.8   Mean   : 2868   Mean   :11796
##  AMDAHL 470V/7C:  1   3rd Qu.: 225.0   3rd Qu.: 4000   3rd Qu.:16000
##  AMDAHL 470V/8 :  1   Max.   :1500.0   Max.   :32000   Max.   :64000
##  (Other)       :203
##       cach            chmin            chmax             perf
##  Min.   :  0.00   Min.   : 0.000   Min.   :  0.00   Min.   :   6.0
##  1st Qu.:  0.00   1st Qu.: 1.000   1st Qu.:  5.00   1st Qu.:  27.0
##  Median :  8.00   Median : 2.000   Median :  8.00   Median :  50.0
##  Mean   : 25.21   Mean   : 4.699   Mean   : 18.27   Mean   : 105.6
##  3rd Qu.: 32.00   3rd Qu.: 6.000   3rd Qu.: 24.00   3rd Qu.: 113.0
##  Max.   :256.00   Max.   :52.000   Max.   :176.00   Max.   :1150.0
##
##     estperf
##  Min.   :  15.00
##  1st Qu.:  28.00
##  Median :  45.00
```

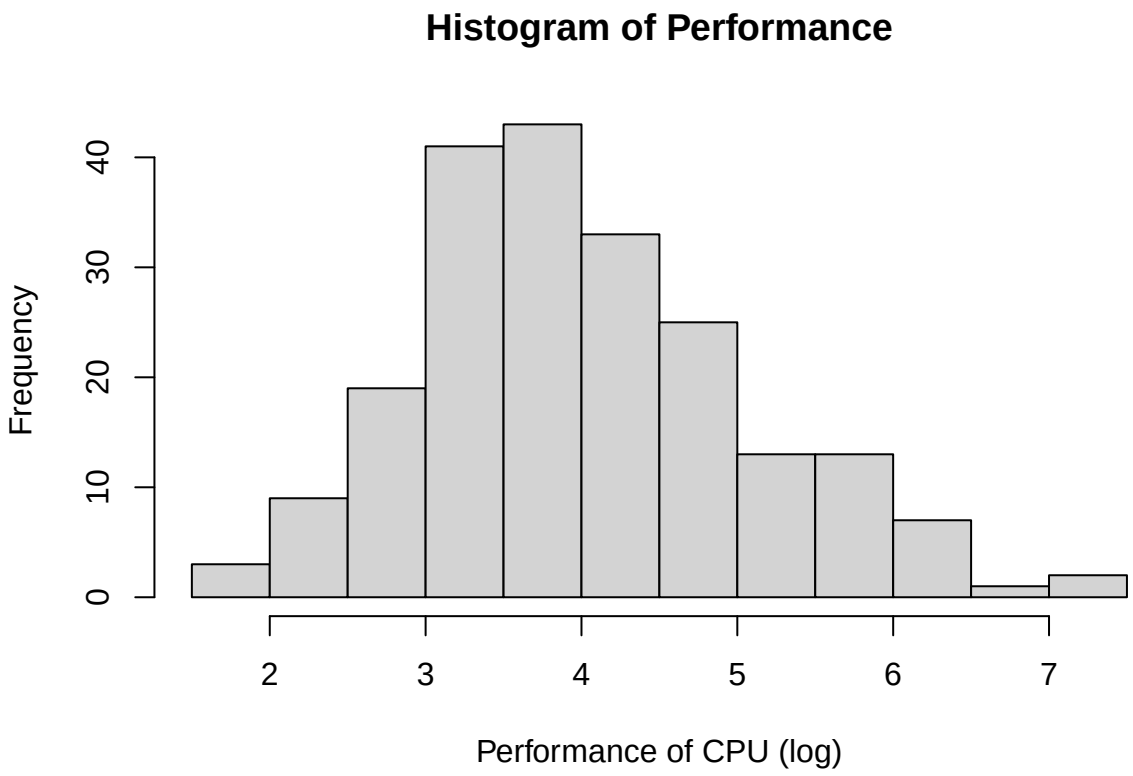
```
##  Mean   :  99.33
##  3rd Qu.: 101.00
##  Max.   :1238.00
##
```

```
hist(cpus$perf,
     main = 'Histogram of Performance',
     xlab = 'Performance of CPU')
```



From the histogram, we can see that the distribution of performance is very skewed to the right. Let's log transform the performance variable to see whether we can alleviate the skewness issue. Indeed, log transformation makes its distribution more normal, as shown in the following histogram.

```
hist(log(cpus$perf),
     main = 'Histogram of Performance',
     xlab = 'Performance of CPU (log)')
```



2. Split Data into Training and Test Sets

Let's split the data into training set (50%) and test set (50%).

```
set.seed(123)
train <- sample(1:nrow(cpus), nrow(cpus)/2)

# Num of observations in training set
length(train)
```

```
## [1] 104
```

3. Train A Regression Tree

We use the tree() method in the tree package to fit a regression tree to the training data.

Note:

- Since the performance is very skewed to the right, let's log transform it.
- The name of a CPU is a unique identifier, so it cannot be used as a predictor.

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.0.4
```

```
# Fit a regression tree
cpus_rt <- tree(log(perf) ~ syct+mmin+mmax+cach+chmin+chmax,
                data = cpus[train,])

# Print the regression tree
cpus_rt
```

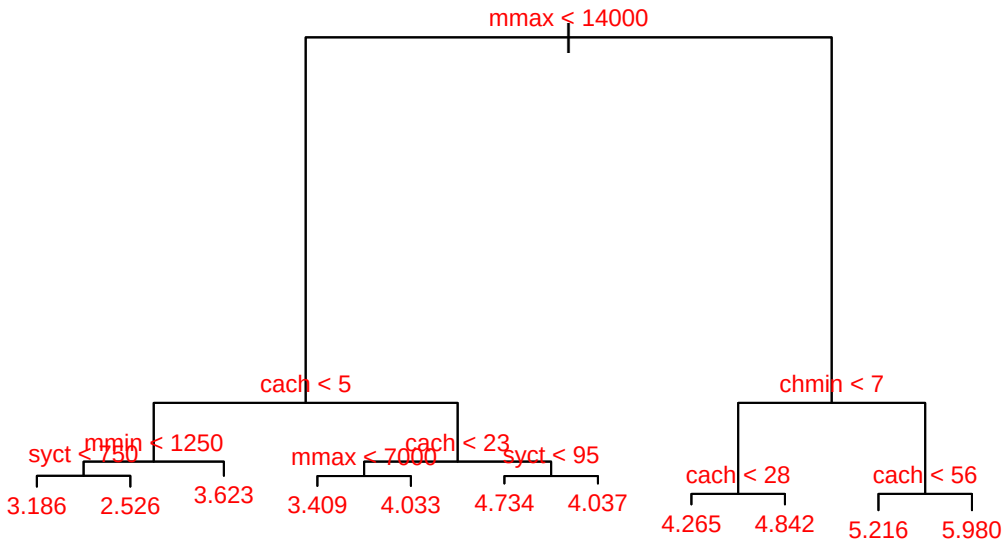
```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 104 112.2000 4.162
##    2) mmax < 14000 63  27.2300 3.552
##      4) cach < 5 36   9.2110 3.215
##        8) mmin < 1250 26   6.1130 3.059
##         16) syct < 750 21   3.6320 3.186 *
##         17) syct > 750 5   0.7227 2.526 *
##        9) mmin > 1250 10   0.7959 3.623 *
##      5) cach > 5 27   8.4820 4.002
##       10) cach < 23 17   2.4850 3.776
##         20) mmax < 7000 7   0.4423 3.409 *
##         21) mmax > 7000 10   0.4398 4.033 *
##       11) cach > 23 10   3.6550 4.386
##         22) syct < 95 5   2.1970 4.734 *
##         23) syct > 95 5   0.2442 4.037 *
##    3) mmax > 14000 41  25.6400 5.098
##      6) chmin < 7 21   4.4770 4.512
##       12) cach < 28 12   2.1840 4.265 *
##       13) cach > 28 9   0.5823 4.842 *
##      7) chmin > 7 20   6.3960 5.713
##       14) cach < 56 7   0.5177 5.216 *
##       15) cach > 56 13   3.2220 5.980 *
```

```
# Summary of the decision tree
summary(cpus_rt)
```

```
##
## Regression tree:
## tree(formula = log(perf) ~ syct + mmin + mmax + cach + chmin +
##     chmax, data = cpus[train, ])
## Variables actually used in tree construction:
## [1] "mmax"  "cach"  "mmin"  "syct"  "chmin"
## Number of terminal nodes:  11
## Residual mean deviance:  0.1611 = 14.98 / 93
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.743600 -0.232800 -0.001818  0.000000  0.213100  1.174000
```

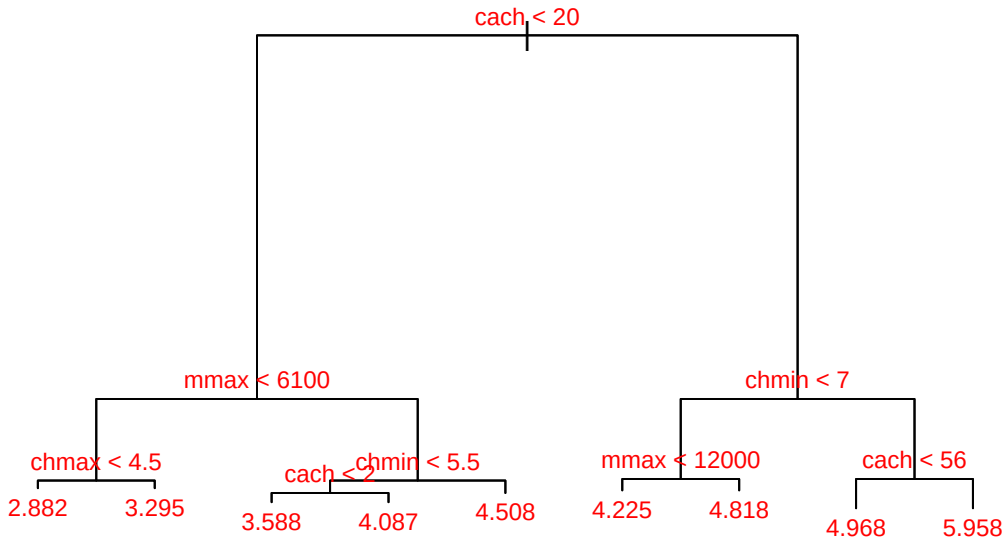
From the summary, we notice that five variables (i.e., mmax, cach, mmin, syct, and chmin) are used to construct the tree.

```
# Plot the decison tree
plot(cpus_rt)
text(cpus_rt, cex = 0.75, col = 'red')
```



Regression trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree. Let's use remove the last 14 observations in the dataset and fit a regression tree model. From the following result, you can find that the tree structure has been significantly changed.

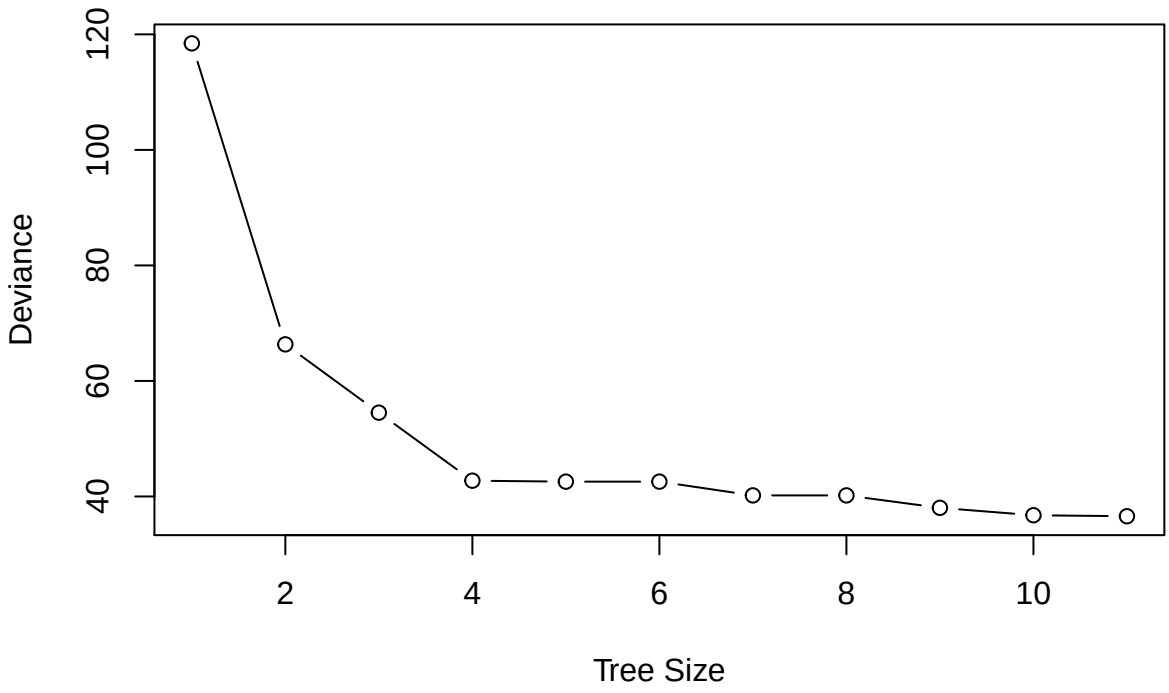
```
# Fit a regression tree
cpus_rt2 <- tree(log(perf) ~ syct+mmin+mmax+cach+chmin+chmax,
                 data = head(cpus[train,], 90))
# Plot the regression tree
plot(cpus_rt2)
text(cpus_rt2, cex = 0.75, col = 'red')
```



The regression tree predicts a performance of 17.84994 [i.e., exp(2.882)] for CPUs with cach $<$ 20, mmax $<$ 6100, and chmax $<$ 4.5.

We can use the cv.tree() method to check if pruning the tree can improve performance by using k-fold cross-validation.

```
cv_cpus <- cv.tree(cpus_rt)
plot(cv_cpus$size, cv_cpus$dev, type='b',
     xlab = 'Tree Size', ylab = 'Deviance')
```



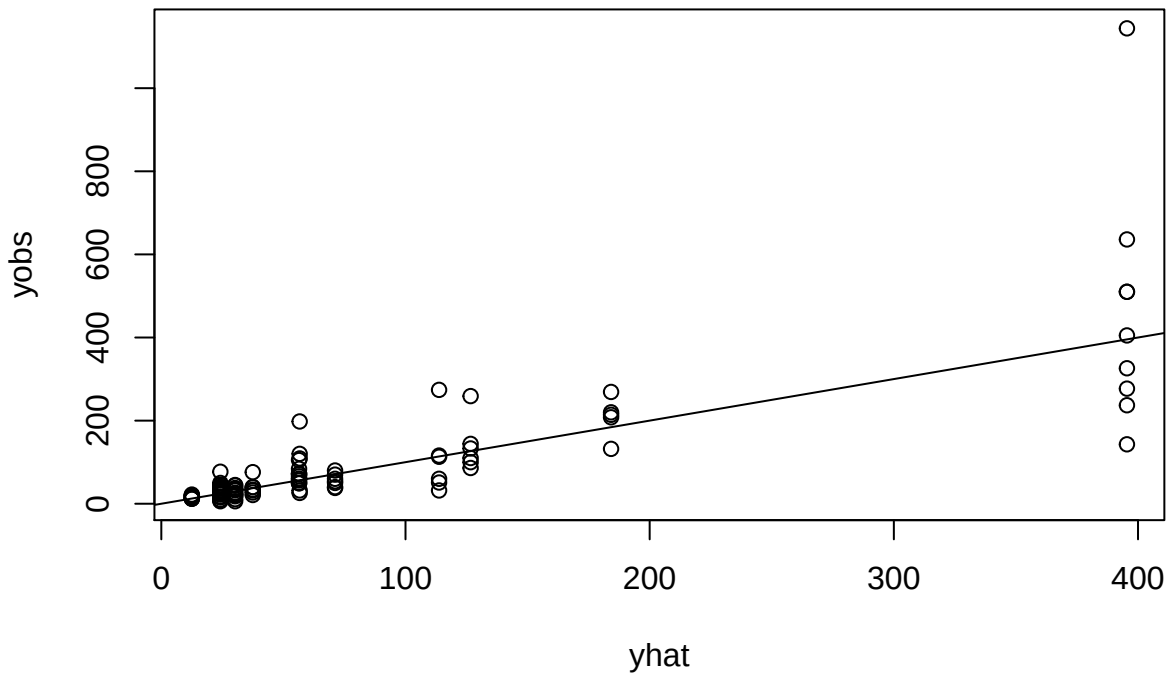
From the above cross-validation result, we can find the tree with the best performance for this case is the most complex tree (with 11 terminal nodes in the tree). So, there is no need to prune the tree. Next, we use the test dataset to test the formance of the regression tree.

4. Test Performance of the Regression Tree

As we log tranform the response variable, the predicted value of the response needs to be transformed back to the original scale.

```
yhat <- exp(predict(cpus_rt, newdata = cpus[-train,]))
yobs <- cpus[-train,'perf']

plot(yhat, yobs)
abline(0,1)
```




```
# Calculate performance of the tree
library(caret)
```

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

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

```
postResample(yhat, yobs)
```

```
##       RMSE   Rsquared        MAE
## 91.0471239  0.6551163 36.2004373
```