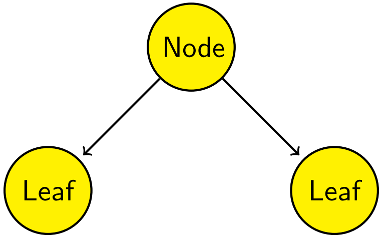
Part 10. Tree-based models

Dr. Nguyen Quang Huy

May 16, 2020

Tree-Based model



- Tree-based method is used for both regression and classification.
- Tree-based methods are simple and useful for interpretation.
- This method can not be compatitive with other algorithms in accuracy.
- Combine the tree-based method with the technique of bagging, random forests or boosting, they often result in dramatic improvements in prediction accuracy.

Regression tree - Boston dataset

Applying tree model to predict value of **medv** in Boston dataset. We use variable **lstat** to build the tree:

- With a cut-point c_1, we divide training dataset into two part:

$$\begin{aligned} R_1 &: \text{lstat} < c_1 \\ R_2 &: \text{lstat} \geq c_1 \end{aligned}$$

  let μ_1 and μ_2 are the means of **medv** in R_1 and R_2.

- We find c_1 to minimize the following

$$\sum^{i \in R_1} (medv_i - \mu_1)^2 + \sum^{j \in R_2} (medv_j - \mu_2)^2$$

- We predict values of **medv** in test set as follows:
  - If **lstat** $< c_1$ then **medv** $= \mu_1$
  - If **lstat** $\geq c_1$ then **medv** $= \mu_2$

Regression tree - Boston data set

- Load Boston data set, standardize the numerical variables, split data into tranning and test set (80%-20%);

```
dat<-Boston
standardize<-function(x){x<-(x-mean(x,na.rm=TRUE))/sd(x,na.rm
for (col in names(dat)){
  if((col!="medv")&class(dat[,col]) %in% c("integer","numeric'
    dat[,col]<-standardize(dat[,col])
  }
}
set.seed(1)
test_index<-createDataPartition(dat$medv, times = 1, p = 0.2,I
train<-dat[-test_index,]
test<-dat[test_index,]
```

Regression tree - Boston data set

- We will build a tree for *medv* based on *lstat* variable using training set. The first step is to find c_1 such that

$$\mathbf{R}_1 = \mathbf{lstat} \leq c_1; \mathbf{R}_2 = \mathbf{lstat} > c_1$$
$$\mu_1 = \mathbb{E}\left(medv|\mathbf{R}_1\right); \mu_2 = \mathbb{E}\left(medv|\mathbf{R}_2\right)$$
$$\sum^{i \in R_1} \left(medv_i - \mu_1\right)^2 + \sum^{j \in R_2} \left(medv_j - \mu_2\right)^2 \to min$$

Regression tree - Boston data set

- We will build a tree for *medv* based on *lstat* variable using training set. The first step is to find c_1 such that

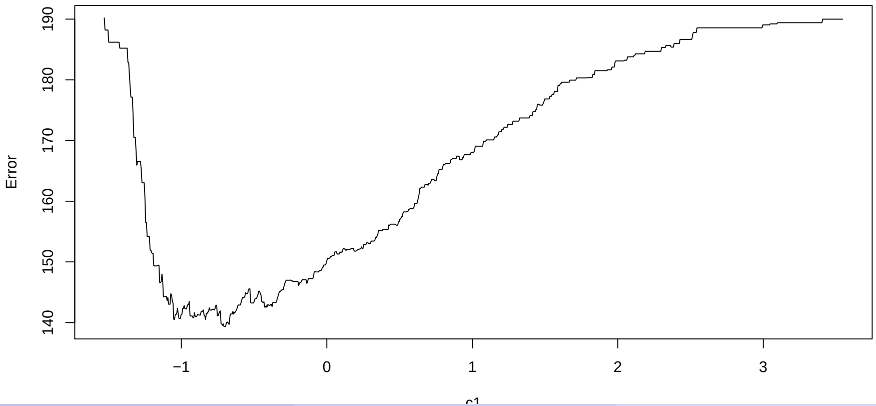
$$\mathbf{R}_1 = \mathbf{lstat} \leq c_1; \mathbf{R}_2 = \mathbf{lstat} > c_1$$
$$\mu_1 = \mathbb{E}\left(medv|\mathbf{R}_1\right); \mu_2 = \mathbb{E}\left(medv|\mathbf{R}_2\right)$$
$$\sum^{i\in R_1} \left(medv_i - \mu_1\right)^2 + \sum^{j\in R_2} \left(medv_j - \mu_2\right)^2 \rightarrow min$$

```
K<-1000
c1<-seq(min(train$lstat),max(train$lstat),length=K)
Error<-rep(0,length(c1))
for(i in 1:K){
  m1<-mean(train$medv[train$lstat<c1[i]])
  m2<-mean(train$medv[train$lstat>=c1[i]])
  Error[i]<-sqrt(sum((train$medv[train$lstat<c1[i]]-m1)^2)+
                    sum((train$medv[train$lstat>=c1[i]]-m2)^2))
```

Regression tree - Boston data set

Find c_1 that minimizes the error (on training dataset)

```
plot(c1,Error,type="l")
```



Regression tree - Boston data set

Find the next node: using the following function

```r
Tree.cut<-function(y,x,K){
c1<-seq(min(x),max(x),length=K)
Error<-rep(0,length(c1))
for(i in 1:K){
  m1<-mean(y[x<c1[i]])
  m2<-mean(y[x>=c1[i]])
  Error[i]<-sqrt(sum((y[x<c1[i]]-m1)^2)+
                  sum((y[x>=c1[i]]-m2)^2))
}
result<-list()
result$cutpoint<-c1[which.min(Error)]
result$error<-min(Error)
Tree.cut<-result}
```

Regression tree - Boston data set

```r
c1<-Tree.cut(train$medv,train$lstat,500)$cutpoint
train.R1<-filter(train,lstat<=c1)
train.R2<-filter(train,lstat>c1)
tree1<-Tree.cut(train.R1$medv,train.R1$lstat,500)
tree2<-Tree.cut(train.R2$medv,train.R2$lstat,500)
tree1$error
```

```
## [1] 74.03281
```

```r
tree2$error
```

```
## [1] 75.6511
```

Regression tree - Boston data set

```r
c1<-Tree.cut(train$medv,train$lstat,500)$cutpoint
train.R1<-filter(train,lstat<=c1)
train.R2<-filter(train,lstat>c1)
tree1<-Tree.cut(train.R1$medv,train.R1$lstat,500)
tree2<-Tree.cut(train.R2$medv,train.R2$lstat,500)
tree1$error
```

```
## [1] 74.03281
```

```r
tree2$error
```

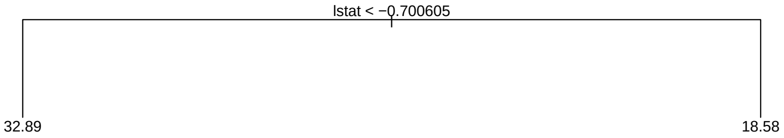
```
## [1] 75.6511
```

```r
ifelse(tree1$error<=tree2$error,
          tree1$cutpoint,tree2$cutpoint)
```

```
## [1] -1.147438
```

Regression tree - Boston data set

```
## [1] -0.705836
```



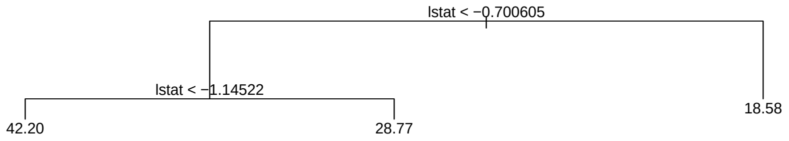
This is a tree with 2 leaves and 1 node.

Regression tree - Boston data set

Applying the same procedure of finding cut-point in each region R_1 and R_2, we have a large tree

```
## [1] -0.705836
```

```
## [1] -1.147438
```

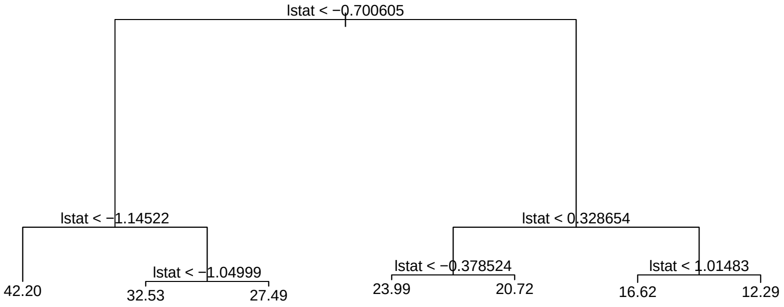


This is a tree with 3 leaves (3 predictions to medv) and 2 nodes

Regression tree - Boston dataset

Function *tree()* in library **tree** fits the data set by tree-based model.

```r
tree.fit<-tree(medv~lstat,data=train)
plot(tree.fit)
text(tree.fit,pretty=0)
```



Regression tree - Boston dataset

- The tree grows until a stopping criteria is reached.
- We could change these stopping criterias using *control = tree.control(...)* inside the tree function.
- There are 4 criterias inside the tree.control: nobs, mincut (5), minsize (10), mindev (0.01).
    - *nobs*: the number of observations you have available
    - *mincut*: the minimum number of observations each child node has to contain
    - *mínize*: specifies the minimum number of observations a node must contain to be split
    - *mindev*: specifies the minumum ratio of the deviance at the root-node to the deviance at the node under consideration.

Regression tree - Boston dataset

```r
# A LARGE TREE
setup1<-tree.control(nobs=nrow(train),
                     mincut=2,minsize=4,mindev=0.001)
tree.fit.full<-tree(medv~lstat,data=train,
  control = setup1)
plot(tree.fit.full)
text(tree.fit.full,pretty=0)
```

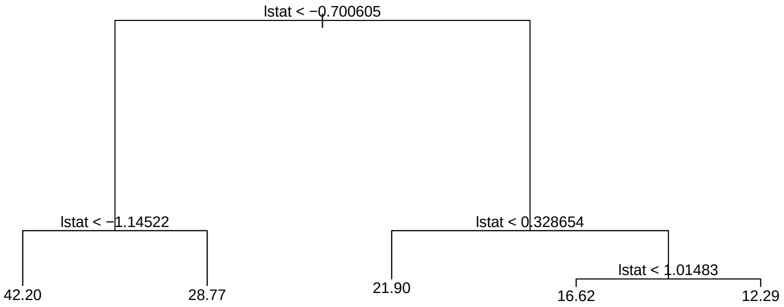
```r
# A SMALL TREE
setup2<-tree.control(nobs=nrow(train),
                     mincut=250,minsize=500,mindev=0.001)
tree.fit.full<-tree(medv~lstat,data=train,
  control = setup2)
plot(tree.fit.full)
text(tree.fit.full,pretty=0)
```

Regression tree - tree pruning

- A large tree produce good predictions on the training set, but is likely to overfit the data (low bias but large variance).
- The such tree may produce poor performance on the test set in general.
- A smaller tree may result in lower variance at the cost of a little bias.
- Number of nodes (or leaves) in the smaller tree should be selected using cross validation approach.
- Function *prune.tree(treename,best* $=$ *L)* is used to prune a large tree until number of leaves is L

Regression tree - tree pruning

```r
tree.fit<-tree(medv~lstat,data=train)
prune.tree<-prune.tree(tree.fit,best = 5) # 5 leaves
plot(prune.tree)
text(prune.tree,pretty = 0)
```



Regression tree - cross validation

- Function **cv.tree** can be used to perform cross-validation on trainning data set.

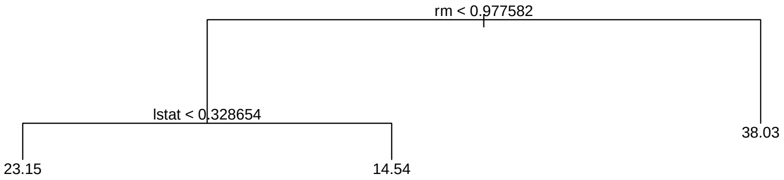
```
setup<-tree.control(nobs=nrow(train),
                    mincut=2,minsize=4,mindev=0.005)
tree.fit<-tree(medv~lstat,data=train,control = setup)
cv.tree1<-cv.tree(tree.fit,K=5)
L<-cv.tree1$size[which.min(cv.tree1$dev)]
mytree<-prune.tree(tree.fit,best=L)
tree.pred<-predict(mytree,newdata=test)
sqrt(mean((tree.pred-test$medv)^2))
```

```
## [1] 5.726087
```

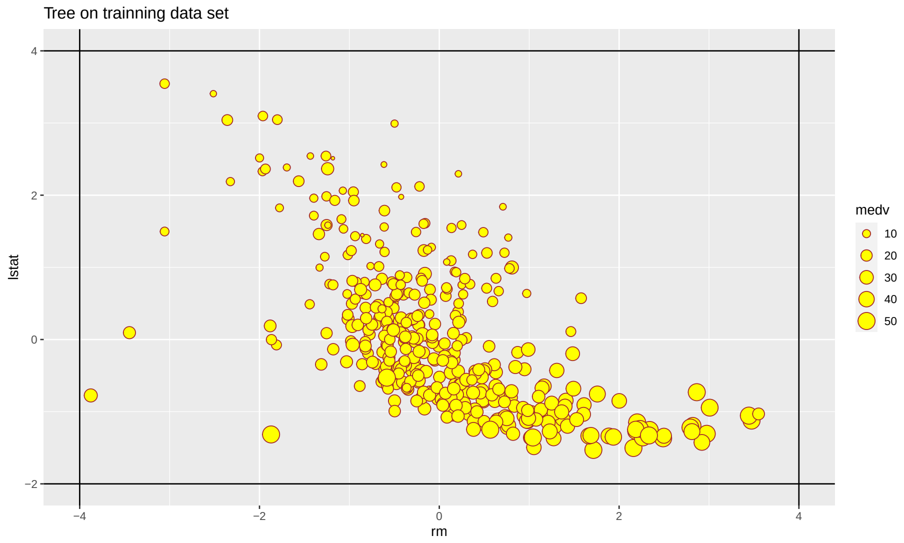
Regression tree - multiple variables

Similar to linear models, we can use multi-variables in a tree model.

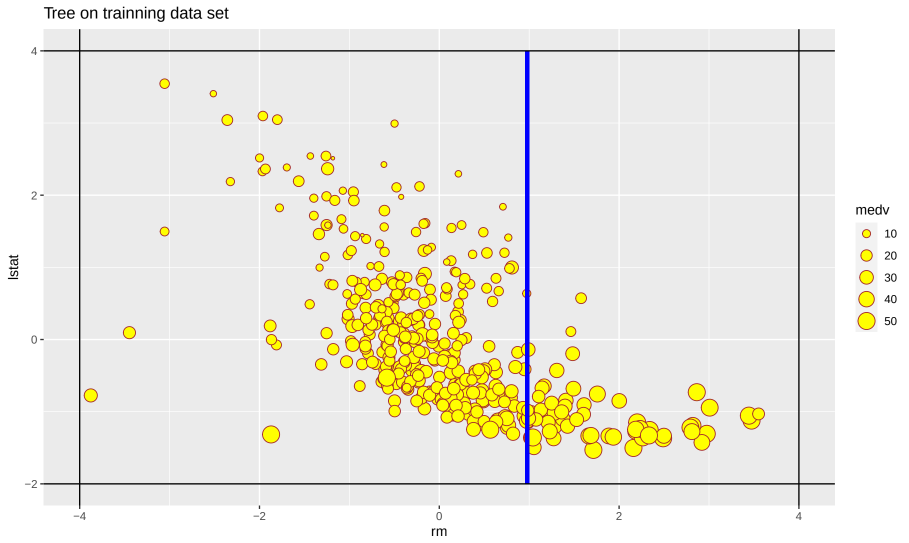
```
tree.fit2<-tree(medv~lstat+rm,data = train)
prune.tree2<-prune.tree(tree.fit2,best = 3)
plot(prune.tree2)
text(prune.tree2,pretty=0)
```



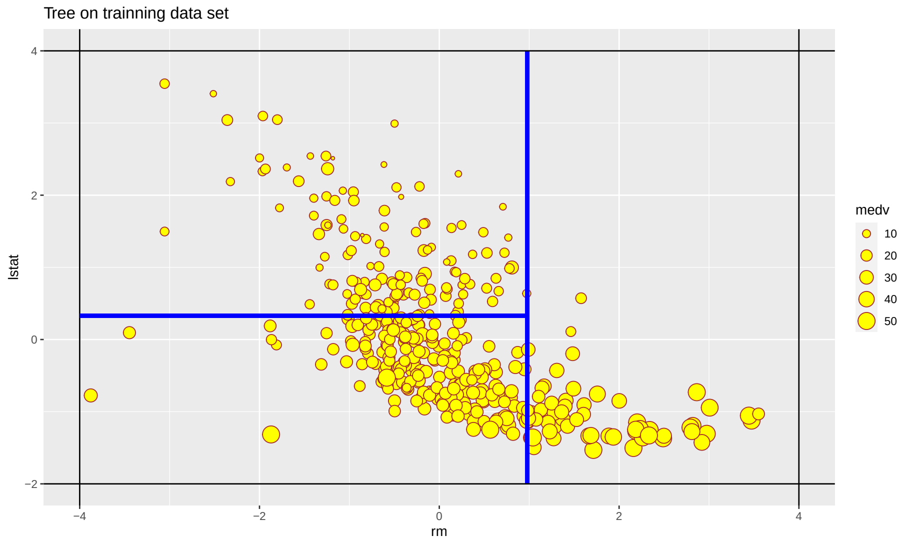
Regression tree - Boston dataset



Regression tree - Boston dataset

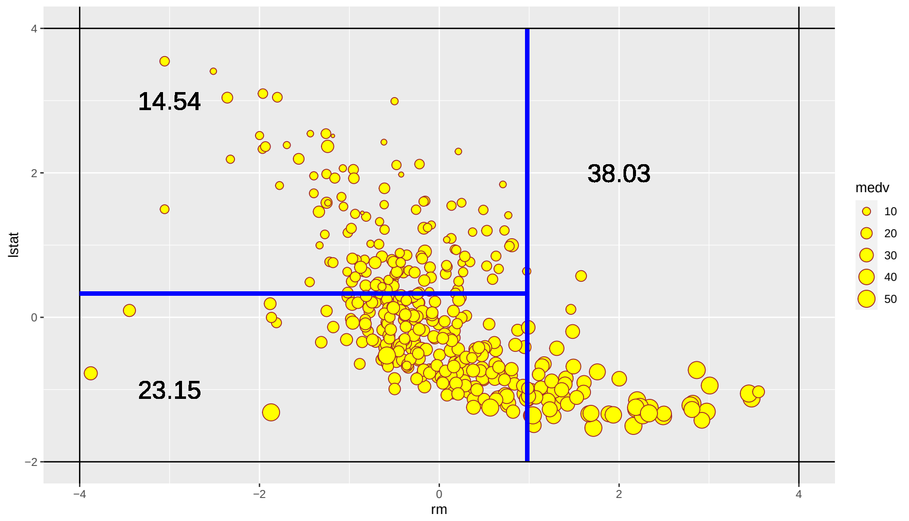


Regression tree - Boston dataset

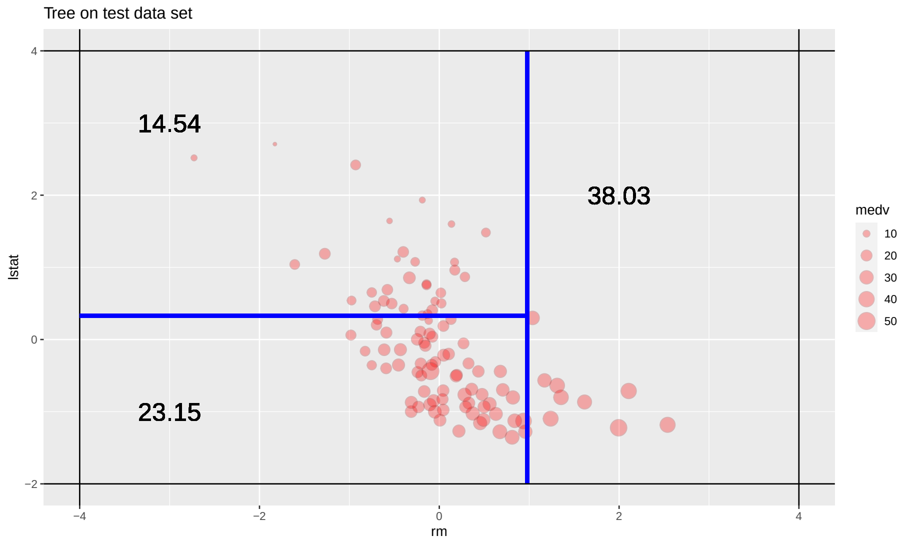


Regression tree - Boston dataset

Tree on trainning data set



Regression tree - Boston dataset



Regression tree

The process of building a regression tree

- Step 1: divide the set of possible values for $X_1, X_2, \cdots, X_p$ into J distinct and non-overlapping rectanges (hyper-rectanges in high dimension) $R_1, R_2, \cdots, R_J$ that minimize

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2$$

- For every observation that falls into the region R_j, we make the same prediction, which is the mean of the response values for the training observations in R_j i.e. $\bar{y}_{R_j}$

However, it is impossible to consider every possible partition of the feature space into J boxes. In practice, they take the recursive binary splitting algorithm

Regression tree

Recursive binary splitting algorithm

- is a top-down algorithm where it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- is a greedy algorithm because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.
- At each step, we select a predictor X_j and cutpoint c that minimize

$$\sum_{i \in R_1} (y_i - \bar{y}_{R_1})^2 + \sum_{k \in R_2} (y_k - \bar{y}_{R_2})^2$$

where $R_1 = X|X_j \leq c$ and $R_2 = X|X_j > c$

Regression tree

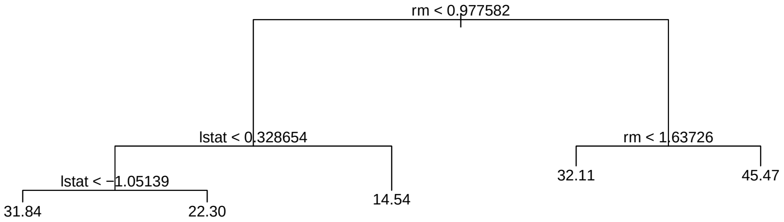
```
setup<-tree.control(nobs=nrow(train),
                    mincut=2,minsize=4,mindev=0.005)
tree.fit<-tree(medv~lstat+rm,data=train,control = setup)
cv.tree1<-cv.tree(tree.fit,K=5)
L<-cv.tree1$size[which.min(cv.tree1$dev)]
# number of leaves by cross validation
L
```

```
## [1] 11
```

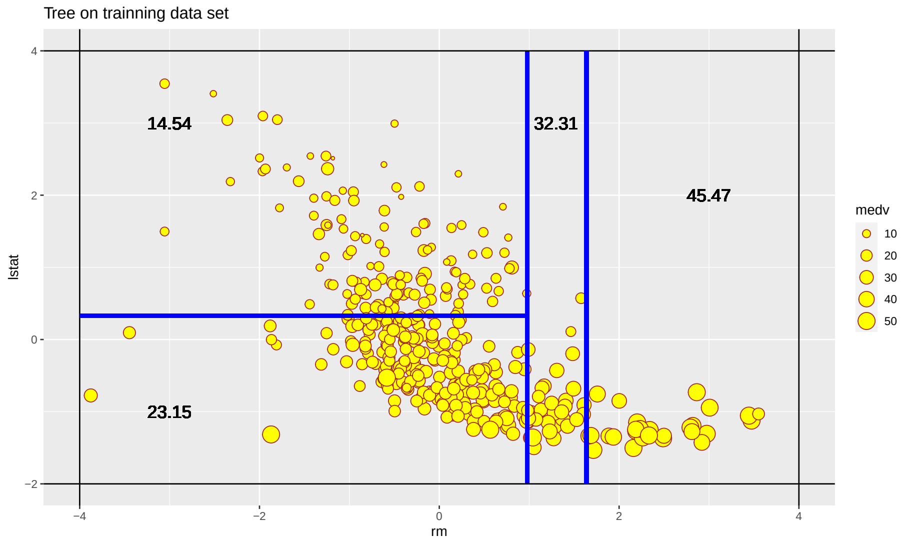
```
mytree<-prune.tree(tree.fit,best=L)
tree.pred<-predict(mytree,newdata=test)
sqrt(mean((tree.pred-test$medv)^2))
```

```
## [1] 5.134981
```

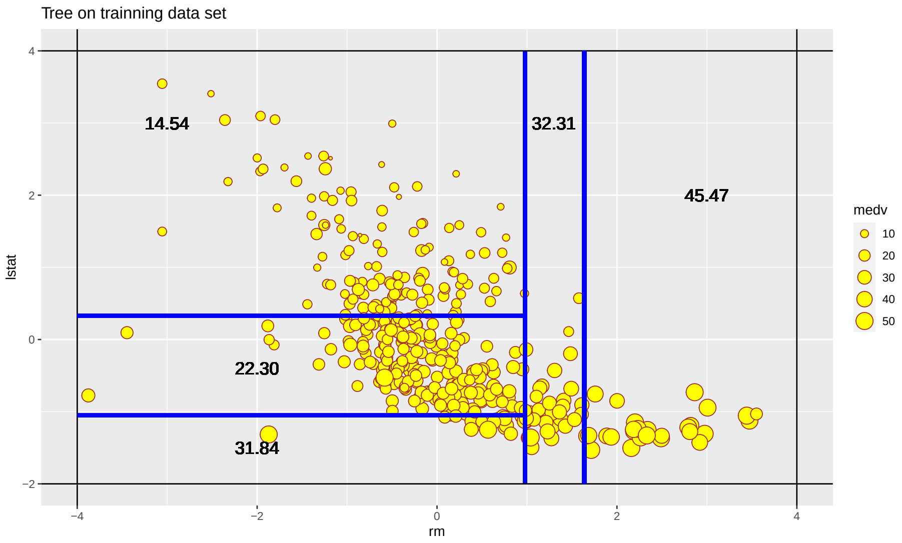
Regression tree



Regression tree - Boston data set



Regression tree - Boston data set



Regression tree - Boston data set

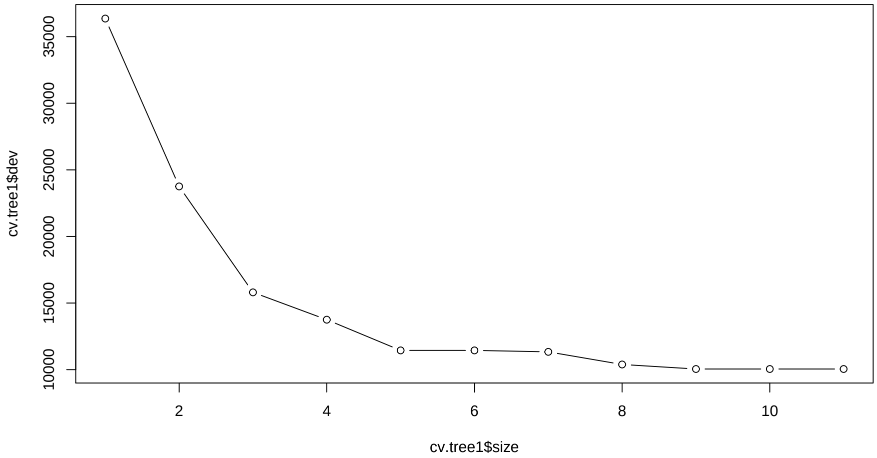
We can build a tree with all variable dataset (the syntax is similar to *lm()* function)

```r
setup<-tree.control(nobs=nrow(train),
                    mincut=2,minsize=4,mindev=0.005)
tree.fit<-tree(medv~.,data=train,control=setup)
plot(tree.fit)
text(tree.fit,pretty = 0)
```

- A large tree is grown until a stopping criteria is reached.
- Function *cv.tree()* is used to perform cross-validation.
- The tree with the lowest cross validation error will be chosen.

```r
cv.tree1<-cv.tree(tree.fit,K=5)
plot(cv.tree1$size,cv.tree1$dev,type="b")
```

Regression tree - Boston data set



Regression tree - Boston data set

```
setup<-tree.control(nobs=nrow(train),
                    mincut=2,minsize=4,mindev=0.005)
tree.fit<-tree(medv~.,data=train,control=setup)
L<-cv.tree1$size[which.min(cv.tree1$dev)]
# number of leaves by cross validation
L
```

```
## [1] 11
```

```
mytree<-prune.tree(tree.fit,best=L)
tree.pred<-predict(mytree,newdata=test)
sqrt(mean((tree.pred-test$medv)^2))
```

```
## [1] 4.5396
```

Classification tree

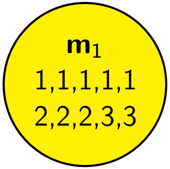
- When the response variable Y is categorial we are in the classification setting.
- Growing a classification tree is quite similar to the task of growing a regression tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits. An alternative to RSS is the classification error rate: At a node m, the *error rate* is calculated as follow

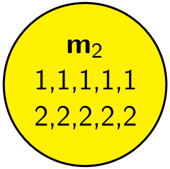
$$E = 1 - max(p_{mk})$$

where p_{mk} is the proportion of training observations in the node that are from k^{th} class, $k = 1, 2, \cdots, K$.

Classification tree

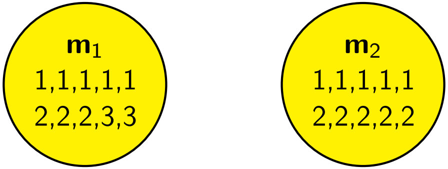
Calculate the error rate in each note





Classification tree

Calculate the error rate in each note



- Node m_1: we have $p_1 = 1/2; p_2 = 3/10$ and $p_3 = 2/10$

$$E(m_1) = 1 - max\{1/2, 3/10, 2/10\} = 0.5$$

- Node m_2: we have $p_1 = 1/2; p_2 = 1/2$ and $p_3 = 0$

$$E(m_2) = 1 - max\{1/2, 1/2, 0\} = 0.5$$

$\rightarrow$ the error rate is not sensitive for growing a tree.

Classification tree

- In practice, two other measures are preferable: *Gini index* and *Entropy*.

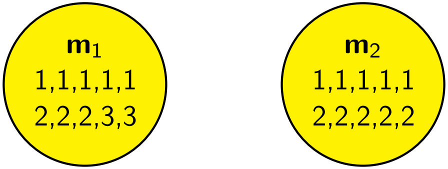
$$Gini\ index = \sum_{i=1}^{K} p_{mi} \times (1 - p_{mi})$$

$$Entropy = -\sum_{i=1}^{K} p_{mi}\ log\,(p_{mi})$$

- The error rate, the Gini index, and the Entropy are measures of node purity i.e. these measures will take on a small value if the node is pure.
- The Gini index and the entropy are quite similar numerically.
- In a classification tree, either the Gini index or the entropy are used to evaluate the quality of a split.

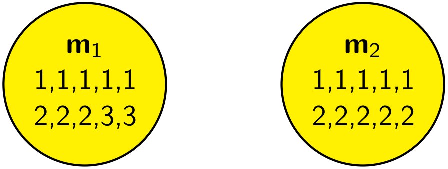
Classification tree

Calculate the Gini index and the entropy in each note



Classification tree

Calculate the Gini index and the entropy in each note



- Node m_1: we have $p_1 = 1/2; p_2 = 3/10$ and $p_3 = 2/10$

$$\begin{aligned} G(m_1) &= 1/2 \times 1/2 + 3/10 \times 7/10 + 2/10 \times 8/10 = 0.62 \\ D(m_1) &= -(1/2 log(1/2) + 3/10 log(3/10) + 2/10 log(2/10)) = 1.03 \end{aligned}$$

- Node m_2: we have $p_1 = 1/2; p_2 = 1/2$ and $p_3 = 0$

$$\begin{aligned} G(m_2) &= 1/2 \times 1/2 + 3/10 \times 7/10 + 2/10 \times 8/10 = 0.5 \\ D(m_2) &= -(1/2 log(1/2) + 3/10 log(3/10) + 2/10 log(2/10)) = 0.69 \end{aligned}$$

Classification tree

Write a function to calculate the error rate, the Gini index and the Entropy of a categorial variable.

```r
purity<-function(v){
  x<-as.numeric(v)
  m<-min(x)
  M<-max(x)
  n<-length(x)
  p<-rep(0,M-m+1)
  for (j in m:M){
    p[j]<-sum(x==j)/length(v)
  }
  result<-list()
  result$err<-1-max(p)
  result$gini<-sum(p*(1-p))
  result$entropy<-sum(-p*log(p))
  purity<-result
```

Classification tree

Load **Default** data set from ISRL package, split data into trainning - test set (50% - 50%) and build a tree model where *default* variable depends on *balance* variable.

- Build a tree with two leaves?
- Which measure that function *tree* uses to split tree nodes?

```
dat<-Default
standardize<-function(x){x<-(x-mean(x,na.rm=TRUE))/sd(x,na.rm
for (col in names(dat)){
  if(class(dat[,col]) %in% c("integer","numeric")){
    dat[,col]<-standardize(dat[,col])
  }
}

set.seed(1)
test_index<-createDataPartition(dat$default, times = 1, p = 0.
```

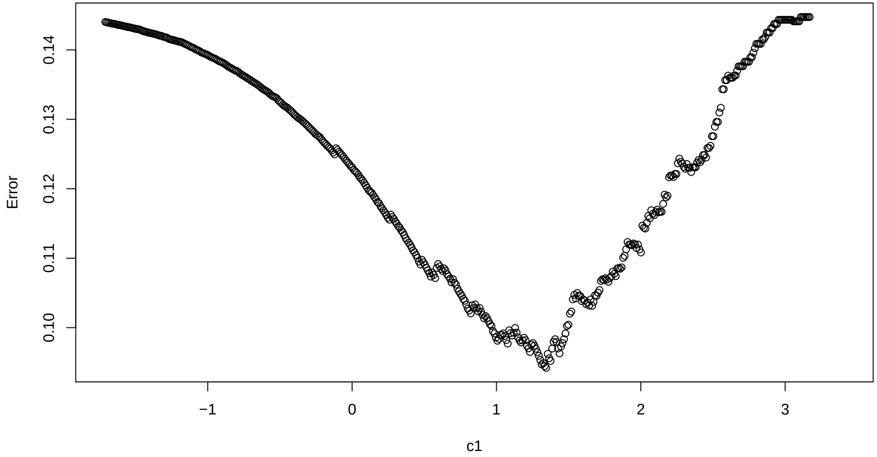
Classification tree

```
K<-500
N<-dim(train)[1]
c1<-seq(min(train$balance)*0.99,max(train$balance)*0.99,length
Error<-rep(0,length(c1))
for(i in 1:K){
  leaf1<-train$default[train$balance<=c1[i]]
  leaf2<-train$default[train$balance>c1[i]]
  Error[i]<-length(leaf1)/N*purity(leaf1)$entropy+length(leaf
}

c1[which.min(Error)]
plot(c1,Error)
```

Classification tree

```
## [1] 1.344704
```



Classification tree

Build a tree model where *default* variable depends on other variables. Evaluate the performace of this model.

```
setup<-tree.control(nobs=nrow(train),
                   mincut=2,minsize=4,mindev=0.005)
tree.fit<-tree(default~.,data=train,control = setup)
cv.tree1<-cv.tree(tree.fit,K=5)
L<-cv.tree1$size[which.min(cv.tree1$dev)]
# number of leaves by cross validation
L
```

```
## [1] 6
```

```
mytree<-prune.tree(tree.fit,best=L)
tree.pred<-predict(mytree,newdata=test,type="class")
table(tree.pred,test$default)
```

```
##
```

Decision tree versus linear model

In general, linear models assume the following form of f:

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i$$

whereas tree models assume a function of the form

$$f(\mathbf{x}) = \sum_{k=1}^{M} c_k \times \mathbb{I}_{\mathbf{x} \in R_k}$$

- Linear models are high-bias and low-variance method while tree models are low-bias and high-variance method
- If the relationship between the features and the response is approximated by a linear model, then linear models will outperform tree models.
- If there is a non-linear and complex relationship between the features and the response, then decision trees may outperform classical

Advantages and disadvantages of tree-based models

- Trees are very easy to explain to people (even easier to explain than linear regression).
- Decision trees more closely mirror human decision-making than other approaches.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert.
- Trees can easily handle qualitative predictors.
- Trees generally do not have the same level of predictive accuracy as other methods.
- Trees can be very non-robust i.e. a small change in the data can cause a large change in the final estimated tree (high variance)

Bagging, random forests, and boosting use trees to construct more powerful prediction models.