STAT 621 Lecture Notes
Regression and Classification Trees

Tree methods predict values of a response, either continuous or categorical, by partitioning the predictor space. The predicted value of a response for a given set of regressors is determined by summarizing observed responses from observations that fall into the same subset of the partiton.

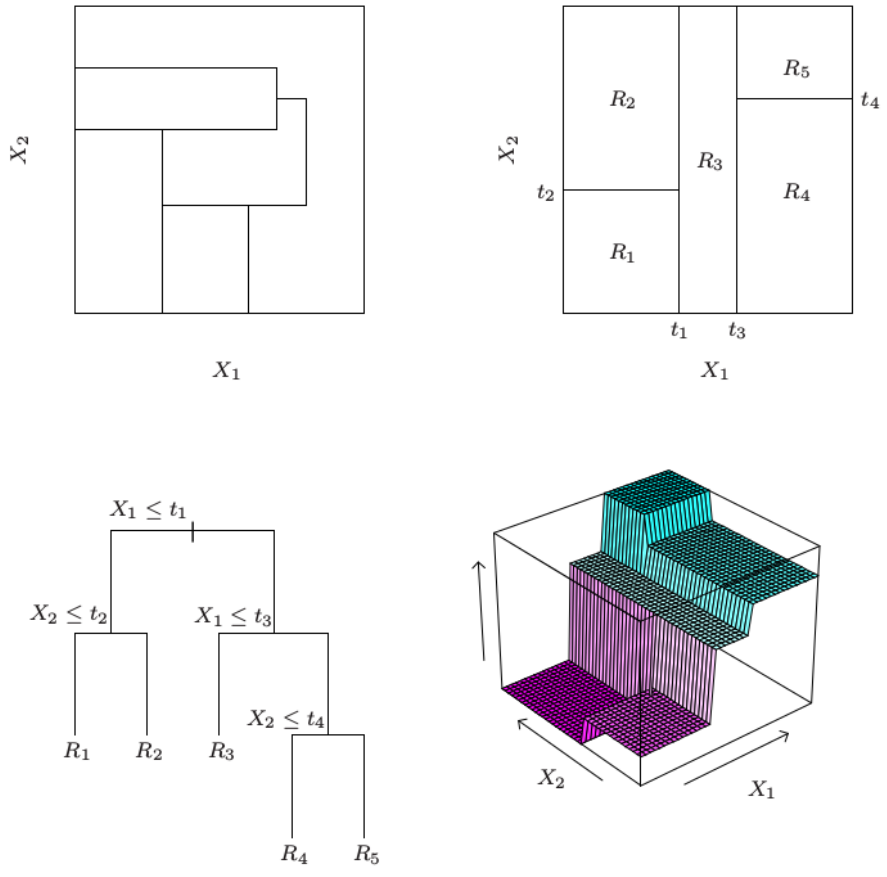
In general these methods are called *Decision Trees*. The term *Regression Trees* refers to those decisions trees where the response variable is numeric, and the term *Classification Trees* refers to those where the response is categorical. This topic is covered in more detail in Chapter 9 of ESL and Chapter 8 of AISL.

First we'll describe the idea for the simple case of a response Y and two predictors X_1 and X_2. There are two main steps.

(1) Partition the predictor space into J regions, $R_1, \ldots, R_J$.

(2) For every observation that falls into region R_j, the predicted response is a summary of the responses over the observations in that region. For continuous responses, this is most often taken as the mean $\overline{Y}_j$. For categorical responses, this is usually the category with the highest sample proportion, $\widehat{p}_j$.

A few stages in the process are shown in the following figure taken from ESL (Figure 9.2). Discuss.



Regression Trees: For convenience, we will describe the tree-fitting process for a numeric response Y. After that we will generalize to categorical responses; the idea is basically the same.

Most tree-fitting algorithms use what is called *recursive binary partitioning*. This is shown in the upper-right of the figure on page 1. First, the entire predictor space is first split into two regions, determined by the predictor and split point that gives the lowest sum of squared errors (SSE). For example in the top right of the previous figure, the first split point occurs at t_1. Describe what this means in terms of the sum of squared errors.

Describe the rest of the steps in the partitioning for that example.

The figure in the upper-right corner is a nice visual summary of the partition, and makes it easy to understand how the response is predicted. However it's inconvenient when there are more than two regressors. The tree diagram in the lower-left is equivalent and can easily be generalized to higher dimensions. Verify that this is the same model.

Finally a predicted surface is shown in the lower right panel of the figure. Here for each subregion the predicted surface is modeled as the mean response over that region. This may be written as

$$\widehat{g}(X_1, X_2) = \sum_{j=1}^{5} \overline{Y}_j I\{(X_1, X_2) \in R_j\}.$$

Where $\overline{Y}_j$ is the average of the responses in subset R_j.

There's still the question of how we actually identify the "best" partition, leading to the most accurate preditions of the response. Finding the best possible partition could be a difficult optimization problem. To reduce computations we consider what are called *greedy algorithms*. This term refers to algorithms that only look one step ahead, rather than attempting to find a best overall partition. Suppose we have regressors $X_1, \ldots, X_p$. Let $R_1(j, s)$ and $R_2(j, s)$ be the regions we get when, on the first step, we split the predictor space where $X_j = s$. We need to find the value s to mininimize

This is done for all the predictors. We choose the variable X_j and point s that gives the smallest minimum. This divides the space into the two regions, and completes the first step. The process is repeated on these resulting two regions R_1 and R_2. And so on.... Until....?

A problem with an algorithm like this is that if left unchecked, it will overfit the data. We can always improve our predictions by making finer and finer splits, until there is exactly one data point in each region and the squared error is zero.

A common strategy is to first create a large tree that is likely overfitting the data. Then that tree is pruned using a cross validation technique and/or incorporating a penalty term into a measure of fit. Let's describe one of these approches to selecting a best tree model: *Cost Complexity Pruning* (see Algorighm 8.1 page 309 from AISL). The following steps are applied to a training data set.

Cost Complexity Pruning Algorithm: First divide the data into training and testing sets. Apply the following steps to the training data.

(1) Use recursive binary splitting to grow a large tree on the training data. Stop when all terminal nodes have fewer than some minimum number of observations. Call this tree T_0.

(2) Let $\alpha_1 < \alpha_2 < \cdots < \alpha_K$ be real numbers. For each α_k find the subtree $T \subset T_0$ such that the following expression is minimized

$$\sum_{j=1}^{|T|} \sum_{x_i \in R_j} (Y_i - \overline{Y}_{R_j})^2 + \alpha_k |T|$$

where $|T|$ is the number of terminal nodes in the tree T. This results in the set of trees $T_1, \ldots T_K$.

(3) Choose the best tree using B-fold crossvalidation. Specifically,

(a) Divide the training data into B sets.

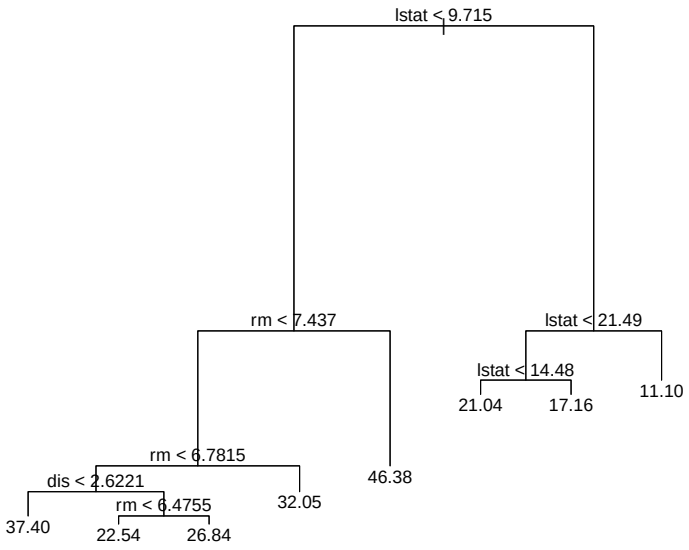
(b) Compute the predicted squared error on each set, with each of the K trees.

(c) Average the prediction errors for each tree.

(d) Choose the tree T_k with the smallest average prediction error. Use this to predict on the test data.

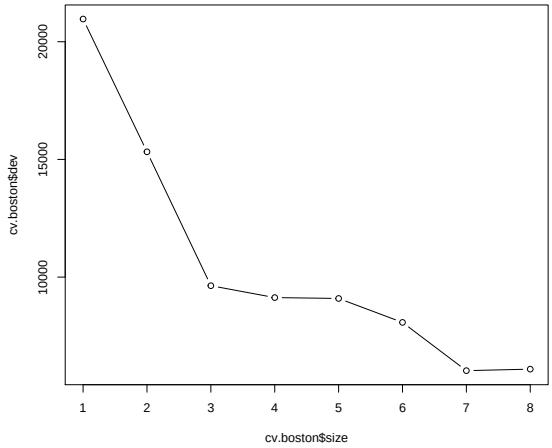
Example: Regression Tree This example follows an exercise in IESL, Chapter 8. Data are housing prices in the Boston suburbs. Below the initial tree is fit and plotted using functions from the `tree` package. Note that the current version of this package requires R 3.6. I'm still using 3.4.4, so I installed an earlier version.

```
require(devtools)
install_version("tree", version = "1.0-39", repos = "http://cran.us.r-project.org")

library(tree)
library(MASS)
attach(Boston)
head(Boston)
     crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat medv
1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98 24.0
2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14 21.6
3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03 34.7
4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94 33.4
5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33 36.2
6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21 28.7

set.seed (1)
train = sample(1: nrow(Boston), nrow (Boston)/2)
tree.boston = tree(medv~., Boston, subset = train)
summary (tree.boston)

   Regression tree:
   tree(formula = medv ~ ., data = Boston, subset = train)
   Variables actually used in tree construction:
   [1] "lstat" "rm"    "dis"
   Number of terminal nodes:  8
   Residual mean deviance:  12.65 = 3099 / 245
   Distribution of residuals:
        Min.   1st Qu.    Median      Mean   3rd (
   -14.10000  -2.04200  -0.05357   0.00000   1.96(

plot(tree.boston)
text(tree.boston)

> tree.boston
node), split, n, deviance, yval
      * denotes terminal node

 1) root 253 20890.0 22.67
   2) lstat < 9.715 103  7765.0 30.13
     4) rm < 7.437 89  3310.0 27.58
       8) rm < 6.7815 61  1995.0 25.52
        16) dis < 2.6221 5   615.8 37.40 *
        17) dis > 2.6221 56   610.3 24.46
          34) rm < 6.4755 31   136.4 22.54 *
          35) rm > 6.4755 25   218.3 26.84 *
       9) rm > 6.7815 28   496.6 32.05 *
     5) rm > 7.437 14   177.8 46.38 *
   3) lstat > 9.715 150  3465.0 17.55
     6) lstat < 21.49 120  1594.0 19.16
      12) lstat < 14.48 62   398.5 21.04 *
      13) lstat > 14.48 58   743.3 17.16 *
     7) lstat > 21.49 30   311.9 11.10 *
```



Next let's see if we should do some pruning. The function `cv.tree` does the cross valdation computations using the cross complexity criterion.

```
cv.boston = cv.tree (tree.boston)
cbind(cv.boston$size, cv.boston$dev)
        [,1]      [,2]
   [1,]    8  6089.100
   [2,]    7  6025.145
   [3,]    6  8074.534
   [4,]    5  9092.334
   [5,]    4  9128.957
   [6,]    3  9634.660
   [7,]    2 15322.652
   [8,]    1 20964.599
plot(cv.boston$size, cv.boston$dev, type='b')
```

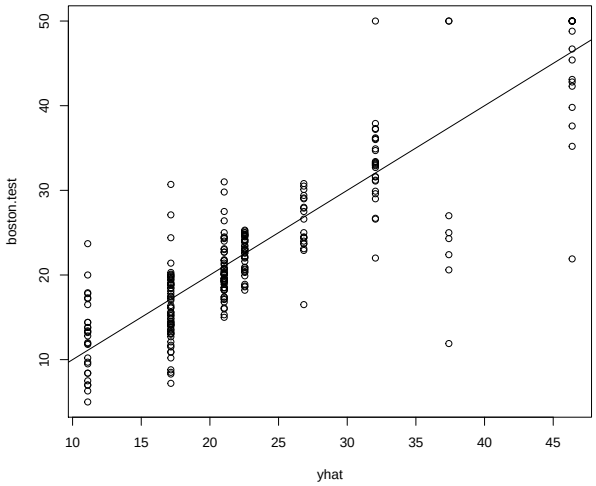


We probably don't need to do any pruning. But if we wanted to, we could use the `prune.tree` function as shown below.

```
prune.boston = prune.tree(tree.boston, best=5)
summary(prune.boston)

   Regression tree:
   snip.tree(tree = tree.boston, nodes = c(6L, 8L))
   Variables actually used in tree construction:
   [1] "lstat" "rm"
   Number of terminal nodes:  5
   Residual mean deviance:  18.45 = 4575 / 248
```

Finally we'll predict the test set, using the unpruned model.

```
yhat = predict(tree.boston, newdata=Boston[-train,])
boston.test = Boston[-train,"medv"]
plot(yhat,boston.test)
abline(0,1)
mean(( yhat-boston.test)^2)
   [1] 25.04559
```



Classification Trees

Algorithms for fitting a classification tree work in basically the same way as those for regression trees. Here we want to predict a class response for Y, based on where the values of the predictors fall in the partitioned predictor space. The predicted value of Y will be taken as the class that occurs most often among observations in the same region of the partition.

The best partition is found again by optimizing recursive binary splits. The main difference is that we need a quantity to optimize other than SSE, one that's appropriate for categorical responses. An obvious one would be the classification error, the percentage of observations in the region that are not in the most common class. It turns out that this one doesn't perform very well in general, so other measures are more common. One is the Gini Index for categorical responses, defined for a response Y with K levels as

$$G = \sum_{k=1}^{K} \widehat{p}_{mk}(1 - \widehat{p}_{mk})$$

Here $\widehat{p}_{mk}$ is the sample proportion of responses in region m that are in class k. This measure will be close to 0 when the bulk of responses are from the same class.

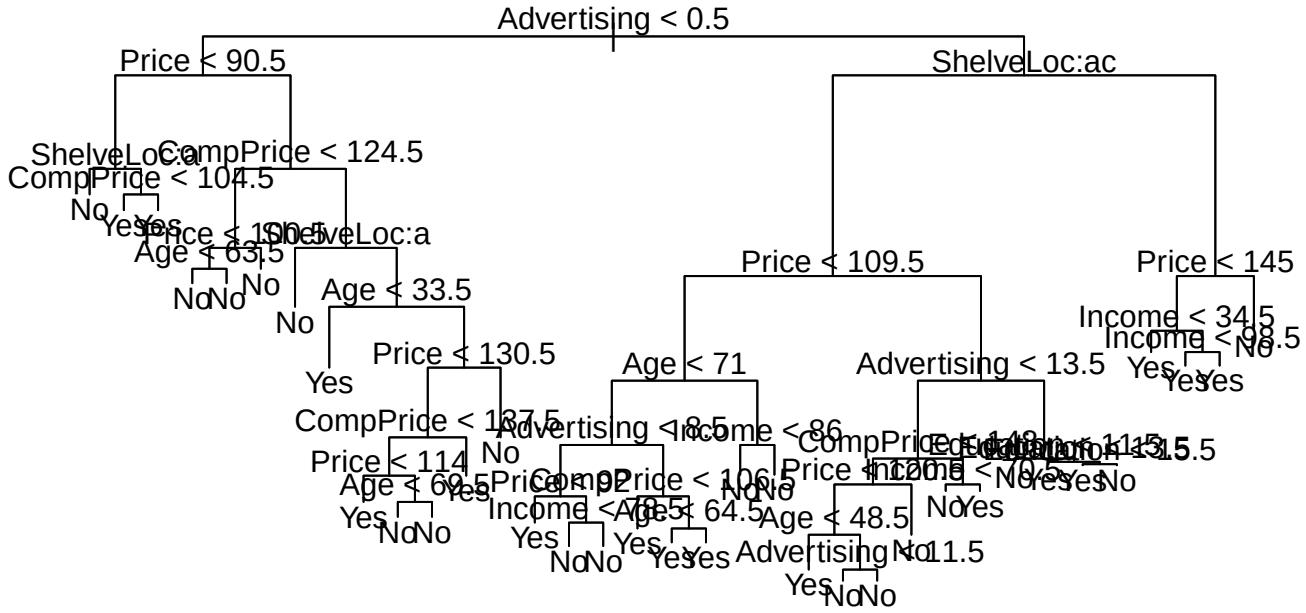
Other things that we need to define for categorical responses?

Example: Classification Tree This example also follows and exercise in Chapter 8 of IESL. Here we fit a classification tree to a simulated data set representing sales of child car seats in 400 stores. Part of the data is shown below and a binary response variabel is created indicating whether or not sales are high.

```
library(ISLR)
attach(Carseats)
High = ifelse(Sales<=8 ,"No","Yes")
Carseats=data.frame(Carseats, High)

head(Carseats)
  Sales CompPrice Income Advertising Population Price ShelveLoc Age Education Urban  US  High
1  9.50       138     73          11        276   120       Bad  42        17   Yes Yes  Yes
2 11.22       111     48          16        260    83      Good  65        10   Yes Yes  Yes
3 10.06       113     35          10        269    80    Medium  59        12   Yes Yes  Yes
4  7.40       117    100           4        466    97    Medium  55        14   Yes Yes   No
5  4.15       141     64           3        340   128       Bad  38        13   Yes  No   No
6 10.81       124    113          13        501    72       Bad  78        16    No Yes  Yes
```

Next we fit a tree model to predict `High` from all regressors except `Sales`. Disucss.

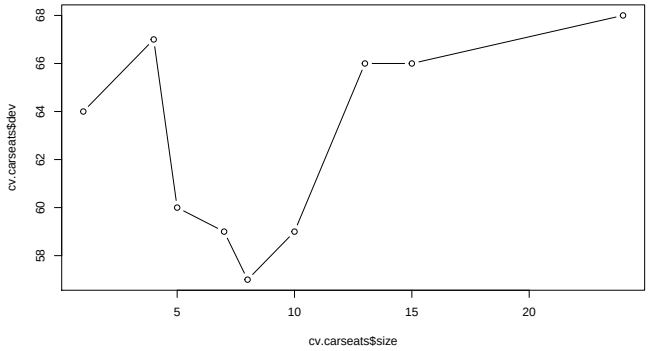
```
tree.carseats = tree(High~.-Sales, Carseats)
summary(tree.carseats)

   Classification tree:
   tree(formula = High ~ . - Sales, data = Carseats, split = "gini")
   Variables actually used in tree construction:
   [1] "Advertising" "Price"       "ShelveLoc"   "CompPrice"   "Age"         "Income"         "Education"
   Number of terminal nodes:  35
   Residual mean deviance:  0.4603 = 168 / 365
   Misclassification error rate: 0.115 = 46 / 400

plot(tree.carseats)
text(tree.carseats)
```



That model is likely overfit. Let's see if pruning is in order. I'll split the data into test and training sets first. I run the `cv.tree` function on the model fit with the training data. Note that because the response is binary, we need to specify `FUN=prune.misclass` in the cv function.

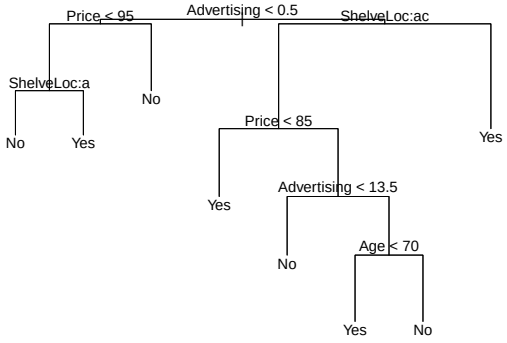
```
set.seed(2)
train=sample(1:nrow(Carseats), 200)
Carseats.test = Carseats [-train,]
High.test = High[-train]

tree.carseats=tree(High~.-Sales, Carseats, subset = train, split="gini" )
cv.carseats =cv.tree(tree.carseats, FUN = prune.misclass)
```

```
plot(cv.carseats$size, cv.carseats$dev, type ="b")
names(cv.carseats)
   [1] "size"   "dev"    "k"      "method"
cbind(cv.carseats$size, cv.carseats$dev)
     [,1] [,2]
 [1,]   24   68
 [2,]   15   66
 [3,]   13   66
 [4,]   10   59
 [5,]    8   57   <-----
 [6,]    7   59
 [7,]    5   60
 [8,]    4   67
 [9,]    1   64
```



Now I prune the tree at 7 terminal nodes. Note again the syntax for binary data.

```
prune.carseats = prune.misclass(tree.carseats, best=8)
plot(prune.carseats)
text(prune.carseats)
```



Finally let's predict on the training data. How well do we do?

```
tree.pred = predict(prune.carseats, Carseats.test, type ="class")
table(tree.pred, High.test)
```

```
           High.test
  tree.pred  No  Yes
       No    93   26
       Yes   23   58
```

Other Tree Topics: Bagging, Boosting, Random Forests,....