Topics in Data Science

Lecture 13

Department of Mathematics and Statistics

PURDUE UNIVERSITY NORTHWEST

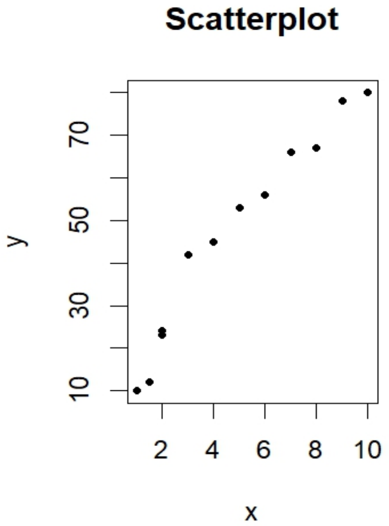
Regression Tree

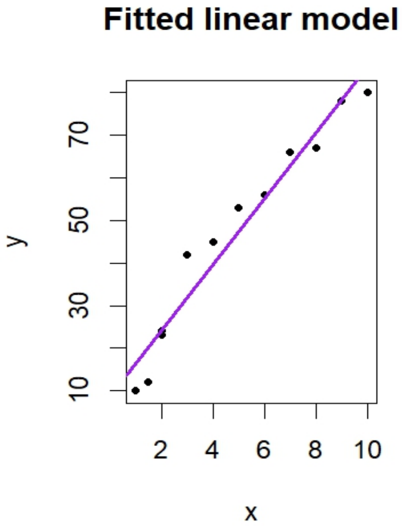
- A regression tree is a machine learning algorithm that predicts numerical outcomes using a tree-like structure.
- Regression trees are conceptually similar to classification trees, except that the label is a number instead of a class (just like in linear regression).
- Basic regression trees partition a data set into smaller groups and then fit a simple model (constant) for each subgroup. We typically the mean of the training data in the region to which it belongs
- Unfortunately, a single tree model tends to be highly unstable and a poor predictor. However, by bootstrap aggregating (bagging) regression trees, this technique can become quite powerful and effective.
- Moreover, this provides the fundamental basis of more complex tree-based models such as random forests and gradient boosting machines. This tutorial will get you started with regression trees and bagging.

Basics of Decision Trees

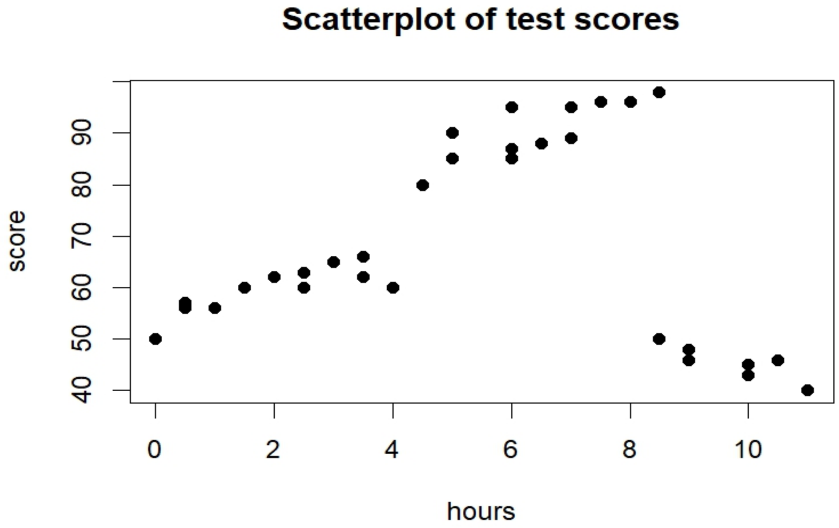
- We want to predict a response or class Y from inputs $X_1, X_2, \cdots X_p$. We do this by growing a binary tree.
- At each internal node in the tree, we apply a test to one of the inputs, say X_i .
- Depending on the outcome of the test, we go to either the left or the right sub-branch of the tree.
- Eventually we come to a leaf node, where we make a prediction.
- This prediction aggregates or averages all the training data points that reach that leaf.

Situation to use Linear regression vs. Regression Tree

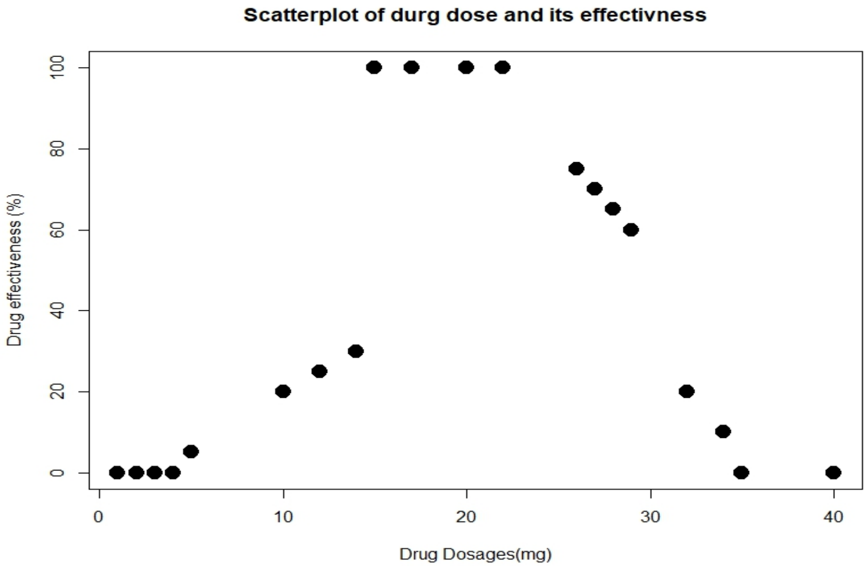




Situation to use Linear regression vs. Regression Tree



Situation to use Linear regression vs. Regression Tree

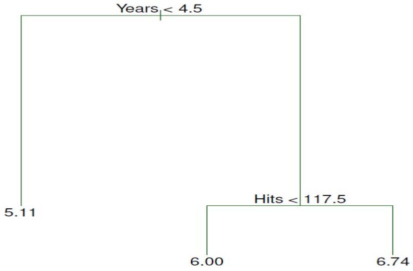


Prediction of baseball player's salary

- Let's predict a baseball player's Salary based on Years (the number of years that he has played in the major leagues) and Hits (the number of hits that he made in the previous year).
- We first remove observations that are missing Salary values, and log-transform Salary so that its distribution has more of a typical bell-shape.
- Note that Salary is measured in thousands of dollars.

Salary Distribution Tree

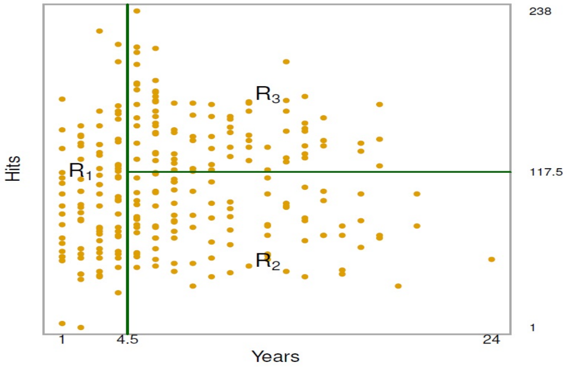
We use the Hitters data set avilable in ILSR package to predict a baseball player's Salary based on Years (the number of years that he has played in the major leagues) and Hits (the number of hits that he made in the previous year).



The tree represents a series of splits starting at the top of the tree. Note that the numbers are in log-transformed scale. So, the mean salary of the playes with less than 4.5 years of experience is 5.11 mean exp(5.11) which is $ 165,6704.

Regression Tree-Example

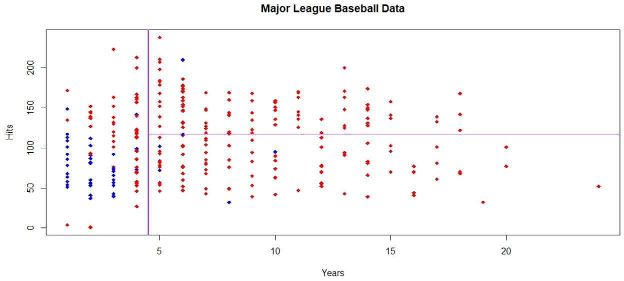
It is clearly observed that there is a three-region partition for the Hitters data set from the regression



- $R1 = X|Years < 4.5$
- $R2 = X|Years \geq 4.5, Hits < 117.5$
- $R3 = X|Years \geq 4.5, Hits \geq 117.5$

Baseball data using R

```
> library(ISLR)
> library(tree)
> Hitters <- na.omit(Hitters)
> attach(Hitters)
> plot(Years, Hits, col=ifelse(log(Salary)>=5.0,"red","blue"),
  pch=18, main="Major League Baseball Data" )
> abline(v=4.5, lwd=2, col="purple")
> clip(4.5,117.5, 25, 117.5)
> abline(h=117.5, lwd=2, col="purple")
```



Regression Tree

- The regions $R1, R2$, and $R3$ are known as terminal nodes or leaves of the tree.
- The decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as internal nodes.
- The two internal nodes are indicated by the text $Years < 4.5$ and $Hits < 117.5$.
- We refer to the segments of the trees that connect the nodes as branches

Regression Tree

- Years is the most important factor in determining Salary, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of hits that he made in the previous year seems to play little role in his salary.
- But among players who have been in the major leagues for five or more years, the number of hits made in the previous year does affect salary, and players who made more hits last year tend to have higher salaries.
- The regression tree is likely an over-simplification of the true relationship between Hits, Years, and Salary, but it's a very nice easy interpretation over more complicated approaches

How do we build the regression tree ?

- 1. We divide the predictor space—that is, the set of possible values for $X_1, X_2, \cdots, X_p$ —into J distinct and non-overlapping regions, $R_1, R_2, \cdots, R_J$.
- 2. For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j

Suppose that in Step 1, we obtain two regions and that the response mean of the training observations in the first region is 10, while the response mean in the second region is 20. Then for a given observation $X = x$, if $x \in R_1$, we will predict a value of 10, and if $x \in R_2$, we will predict a value of 20.

But how do we actually construct the regions?

Constructing the regions

- The regions in theory could have any shape.
- However, we choose to divide the predictor space into high-dimensional rectangles or boxes (for simplicity and ease of interpretation of the resulting predictive model).

Our goal is to find boxes $R_1, R_2, \cdots, R_J$ that minimize the RSS given by

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

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Issue with the tree construction

- Computationally infeasible to consider every possible partition of the feature space into J boxes.
- Thus, we take a top-down, greedy approach called recursive binary splitting.
  - Called top-down since it begins at the top of the tree (all observations below to a single region) and then successively splits the predictor space.
  - Each split is indicated via two new branches further down on the tree.
  - It is greedy since at each step of the tree building process, the best split is made at that particular split (rather than looking ahead and picking a split that will lead to a better tree in a future split)

How to build regression trees?

- One alternative is to the build the tree so long as the decrease in RSS due to each split exceeds a threshold (high).
- This results in smaller trees, however, this is problematic since a worthless split early on in the three might be followed by a very good split later on – that is, a split that leads to a large reduction in RSS later on.
- A better strategy is to grow a very large tree and then prune it back to obtain a subtree.

Pruning decision trees

Pruning is a technique used to simplify decision trees by removing branches that provide little information gain or lead to overfitting. here are two types of pruning:

- pre-pruning (stopping the growth of the tree early)
- post-pruning (removing branches after the tree has been grown).

Pruning can help reduce the complexity of the tree and improve its ability to generalize to new data.

Prepruning

As the name suggests, the criteria are set as parameter values while building the model. Below are some of the pre-pruning criteria that can be used. The tree stops growing when it meets any of these pre-pruning criteria, or it discovers the pure classes.

- maxdepth: This parameter is used to set the maximum depth of a tree. Depth is the length of the longest path from a Root node to a Leaf node. Setting this parameter will stop growing the tree when the depth is equal the value set for maxdepth.
- minsplit: It is the minimum number of records that must exist in a node for a split to happen or be attempted. For example, we set minimum records in a split to be 5; then, a node can be further split for achieving purity when the number of records in each split node is more than 5.
- minbucket: It is the minimum number of records that can be present in a Terminal node. For example, we set the minimum records in a node to 5, meaning that every Terminal/Leaf node should have at least five records. We should also take care of not overfitting the model by specifying this parameter. If it is set to a too-small value, like 1, we may run the risk of overfitting our model.

Postpruning

The idea here is to allow the decision tree to grow fully and observe the cp value.
Next, we prune/cut the tree with the optimal cp value.
In the context of regression trees using the rpart package in R, the "cp" value, or complexity parameter, is a crucial parameter that controls the size and complexity of the tree, influencing both tree growth and pruning, with smaller values leading to larger, potentially overfitted trees

Pruning

- A better strategy is to grow a very large tree To and then prune it back to obtain a subtree.
- How to we find the best subtree?
- We want to select a subtree that leads to the lowest test error rate.
- Given a subtree, we can estimate the test error rate using cross-validation (CV).
- Note that estimate the CV for every possible subtree would take a long time since there are many subtrees.
- Thus, we need a way to select a small set of subtrees to consider

Cost complexity pruning or weakest link pruning gives us a way to do just this!
Rather than looking at all possible subtrees, we consider a sequence of trees indexed by a nonnegative tuning parameter α.

Algorithm for Building a Regression Tree

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α
3. Use K -fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \cdots, K$:
    - Repeat Steps 1 and 2 on all but the kth fold of the training data.
    - Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α. Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α

Algorithm for Building a Regression Tree (continued)

- $|T|$ indicates the number of terminal nodes of the tree T
- R_m is the rectangle or box corresponding to the mth terminal node
- $\hat{y}_{R_m}$ is the predicted response associated with R_m
- α controls a trade-off between the subtree's complexity and its fit to the training data
- for each value of α, there is a corresponding subtree $T \in T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

  is as small as possible

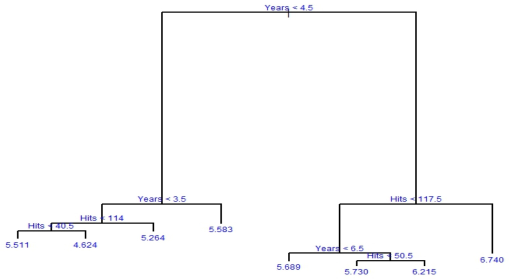
Regression tree-Using R

```
> library(ISLR)
> library(tree)
> attach(Hitters)
> Hitters <- na.omit(Hitters)
> Salary <- na.omit(Salary)
> treefit <- tree(log(Salary) ~ Years + Hits, data=Hitters)
> summary(treefit)
Regression tree:
tree(formula = log(Salary) ~ Years + Hits, data = Hitters)
Number of terminal nodes:  8
Residual mean deviance:  0.2708 = 69.06 / 255
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.2400 -0.2980 -0.0365  0.0000  0.3233  2.1520
```

- There are 8 terminal nodes or leaves of the tree.
- Here "deviance" is just mean squared error; this gives us an RMS error of 0.27

Regression Tree- Using R

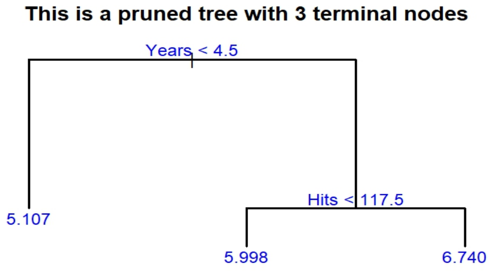
```
> plot(treefit, lwd=2)
> text(treefit,cex=0.75, col="blue")
```



Regression tree for predicting log (salary) from hits and years played. At each internal node, we ask the associated question, and go to the left child if the answer is "yes", to the right child if the answer is "no". Note that leaves are labeled with log (salary).

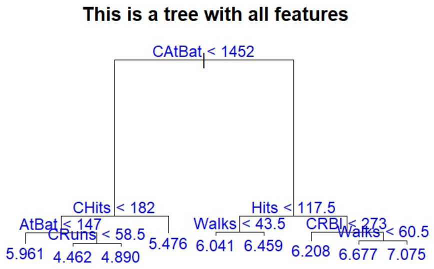
Pruning the Tree

```
> prune.mytree = prune.tree(treefit, best=3)
> plot(prune.mytree, lwd=2)
> text(prune.mytree, col="blue")
> title("This is a pruned tree with 3 terminal nodes")
```



Regression tree with more variables

```
> fulltree=tree(log(Salary)~., data=Hitters)
> summary(fulltree)
> plot(fulltree)
> text(fulltree,cex=0.7, col="blue")
> title("This is a tree with all features")
```



```
Number of terminal nodes: 9
Residual mean deviance: 0.1694 = 43.03 / 254
```

Cross-Validation and Pruning

- The tree package contains functions prune.tree and cv.tree for pruning trees by cross-validation. The function prune.tree takes a tree you fit by tree, and evaluates the error of the tree and various prunings of the tree, all the way down to the stump.
- The evaluation can be done either on new data, if supplied, or on the training data (the default).
- If you ask it for a particular size of tree, it gives you the best pruning of that size.
- If you don't ask it for the best tree, it gives an object which shows the number of leaves in the pruned trees, and the error of each one. This object can be plotted.

Pruning the tree

- The prune.tree has an optional method argument.
- The default is method="deviance", which fits by minimizing the mean squared error (for continuous responses) or the negative log likelihood (for discrete responses).
- The function cv.tree does k-fold cross-validation (default is 10).
- It requires as an argument a fitted tree, and a function which will take that tree and new data. By default, this function is prune.tree.

R code-general

```
my.tree = tree(y ~ x1 + x2, data=my.data) # Fits tree
prune.tree(my.tree,best=5) # Returns best pruned tree
prune.tree(my.tree,best=5,newdata=test.set)
my.tree.seq = prune.tree(my.tree) # Sequence of pruned
# tree sizes/errors
plot(my.tree.seq) # Plots size vs. error
my.tree.seq$dev # Vector of error
# rates for prunings, in order
opt.trees = which(my.tree.seq$dev == min(my.tree.seq$dev))
# Positions of
# optimal (with respect to error) trees
min(my.tree.seq$size[opt.trees])
# Size of smallest optimal tree
```

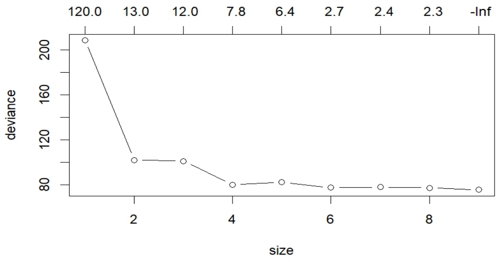
Cross-validation using R

- We can use the cross-validation function "cv.tree()" to determine the optimal level of tree complexity, i.e., the best tree size; cost complexity pruning is used in order to select a sequence of trees for consideration.
- The "cv.tree()" function reports the number of terminal nodes of each tree considered (size) as well as the corresponding error rate (dev) and the value of the cost-complexity parameter

Cross validation using R

```
> fulltree=tree(log(Salary)~., data=Hitters)
> cv.tree=cv.tree(fulltree)
> cv.tree
$size
[1] 9 8 7 6 5 4 3 2 1
$dev
[1] 73.739 73.908 74.499 74.272 83.109 85.306 103.481 103.470 207.694
$k
[1] -Inf  2.315 2.424 2.713   6.377   7.769  11.970 12.696 117.858

> plot(cv.tree,type="b")
```



Split Data into Training and Test Sets

There are different ways to split data into training and test sets in R:

- Use Base R

```
sample <- sample(c(TRUE, FALSE), nrow(df), replace=TRUE, prob=c(0.7,0.3))
train  <- df[sample, ]
test   <- df[!sample, ]
```

- Use caTools package

```
library(caTools)
sample <- sample.split(df$any_column_name, SplitRatio = 0.7)
train  <- subset(df, sample == TRUE)
test   <- subset(df, sample == FALSE)
```

- Use caret package

```
> library(caret)
> trainIndex <- createDataPartition(data$var, p = 0.8, list = FALSE)
> train <- dat[trainIndex, ]
> test <- data[-trainIndex, ]
```

- Choose certain number of observations

```
> train_indices = sample(1:nrow(data), 170)
> train= dat[train_indices, ]
> test= dat[-train_indices, ]
```

Regression tree-Example

```
>library(ggplot2)
>library(tidyverse)
> library(rpart)
> library(rattle)
> library(caTools)
> data("diamonds")
> head(diamonds)
> split = sample.split(diamonds$color, SplitRatio = 0.75)
> train = subset(diamonds, split == TRUE)
> test = subset(diamonds, split == FALSE)
> tree <- rpart(price ~ carat + cut + color + clarity,
               data=train, method="anova",maxdepth=3, cp = 0)
> fancyRpartPlot(tree)
```

Measuring accuracy

Because regression trees predict continuous values instead of classes, we cannot say a model is “80% accurate” or “90% accurate”. Instead, we have to find another way measuring prediction accuracy.
There are many different ways to measure the predictive performance of a regression tree. One intuitive approach is to consider the mean absolute error (MAE), which can be interpreted as the average difference between the predicted and actual values.

Measuring Accuracy-Using R

```
> library(ggplot2)
> library(tidyverse)
> library(caTools)
> data("diamonds")
> head(diamonds)
> split = sample.split(diamonds$color, SplitRatio = 0.75)
> train = subset(diamonds, split == TRUE)
> test = subset(diamonds, split == FALSE)
> tree <- rpart(price ~ carat + cut + color + clarity,
     data=train, method="anova",maxdepth=3, cp = 0)
> pred = predict(tree,test,type="vector")
> actual <- test$price
> MAE <- sum(abs(pred-actual))/length(actual)
> print(MAE)
```