Trees

Data Science 101 Team

Reading

Material from:

James, G., Witten, D., Hastie, T. and Tibshirani, R., 2021. An Introduction to Statistical Learning, Second Edition.

Referred to as ISLR in the following. Freely available online.

If you're interested: Chapter 8.

Example

Figure 8.1 ISLR.

Predict the (log) salary of a baseball player, based on the number of years he has played and the number of hits he made.

Example

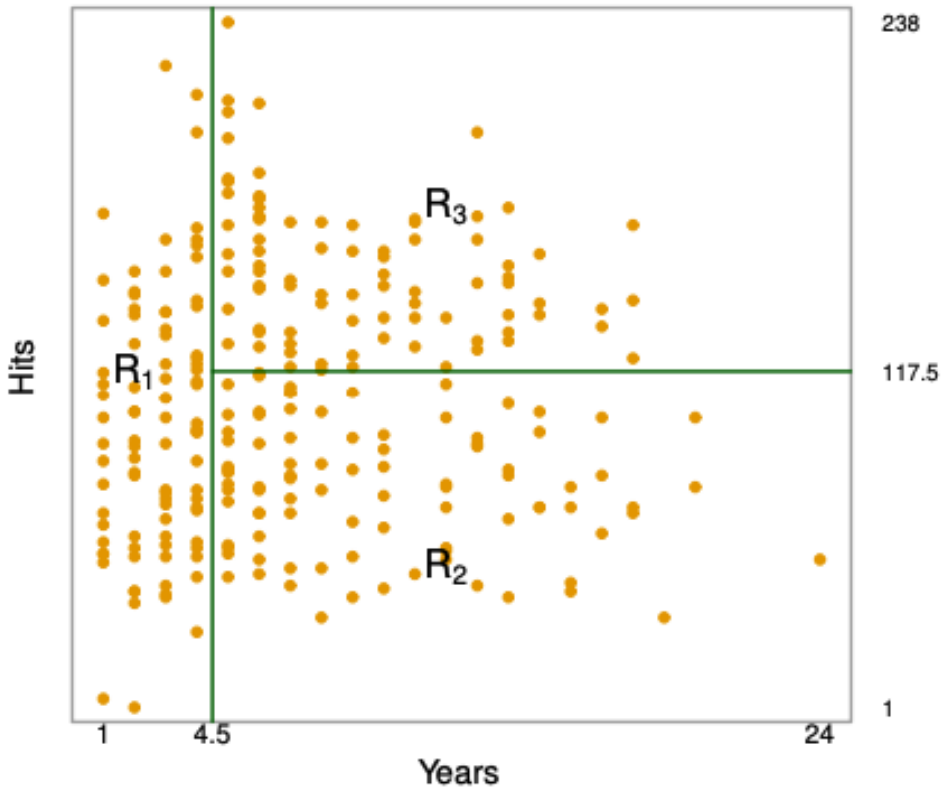
- Top split assigns observations Years < 4.5 to the left branch. The predicted salary for these players is given by the mean salary for players in the dataset with Years < 4.5. For such players, the mean log salary is 5.11, so we would predict a log salary of 5.11.
- Players with Years $>= 4.5$ are assigned to the right branch. We also see that the right branch is further divided by the number of hits.

Example

The tree segments / stratifies the players into three regions of the "predictor space":

1. Players who played less than 4.5 years.
2. Players who played for 4.5 of more years and who made less than 117.5 hits of mean years.
3. Players who played for 4.5 of more years and who made 117.5 or more hits of mean years.

Example



ISLR Figure 8.2.

Example

- These "regions" are called terminal nodes / leaves.
- The trees are typically drawn upside down (leaves are at the bottom of the tree).
- The points along the tree where the predictor space is split are called internal nodes.

Example

- "Years" is the most important factor in determining salary. 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 had little role in his salary.
- However, for players who have played 4.5 or more years, the number of hits does affect the salary, and players who made more hits last year tend to have higher salaries.

Example

- Probably an over-simplification of the true relationship.
- Has the advantage of easier interpretability.

Today: Trees!

- Idea: Segment "predictor space" into simple regions.
- To make a prediction, use e.g. the mean of the observations to which it belongs.
- Note: Trees are simple and useful for interpretation. Typically not competitive with "the best" supervised learning approaches in terms of accuracy.

How to fit trees

1. Divide the predictor space into distinct and non-overlapping regions (see previous picture).
2. For every observation that falls into a specific region R_j, make the same prediction (simply the mean response value for the training observations in R_j).

How to fit trees

- Example: Suppose in first step we only have two regions, R_1 and R_2. Suppose the mean response in the first region is 10, and is 20 in the second region.
- Then, for a given observation, if the observation falls into R_1, we will predict 10, and if it is in the second region, we will predict 20.

How to fit trees

- We want to find the regions or boxed R_1, .., R_J that minimize the 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 j-th box.

- Problem: It is computationally unfeasible to consider every possible partition of the features into J boxes.
- Therefore, use recursive binary splitting (top-down, greedy).

Why is it top-down, greedy?

- Top-down: Begin at the top of the tree, then successively split the predictor space.
- Greedy: At each step of the building process, take the *best split at the particular step*, instead of *looking ahead* and trying to find a split that leads to a better tree in some future step.

How does it work? Step 1

- Notation: $\{X|X_j < s\}$ means the region of the predictor space in which X_j takes on a value that is less than s.
- Step 1: Select the predictor X_j and the cutpoint s such that splitting the predictor space into the region where $X_j < s$ and $X_j \geq s$ leads to the greatest possible reduction in RSS (regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$).
- This means that we consider all predictors $X_1, ..., X_p$ and all possible values of the cutpoint s for each of the predictors, and choose the predictor and the cutpoint such that the resulting tree has lowest RSS.

How does it work? Step 2

- Repeat the process of looking for the best predictor and the best cutpoint in order to split the data further to minimize the RSS within each of the resulting regions.
- However, now split one of the previously identified regions, instead of the entire predictor space.
- Now we have 3 regions.

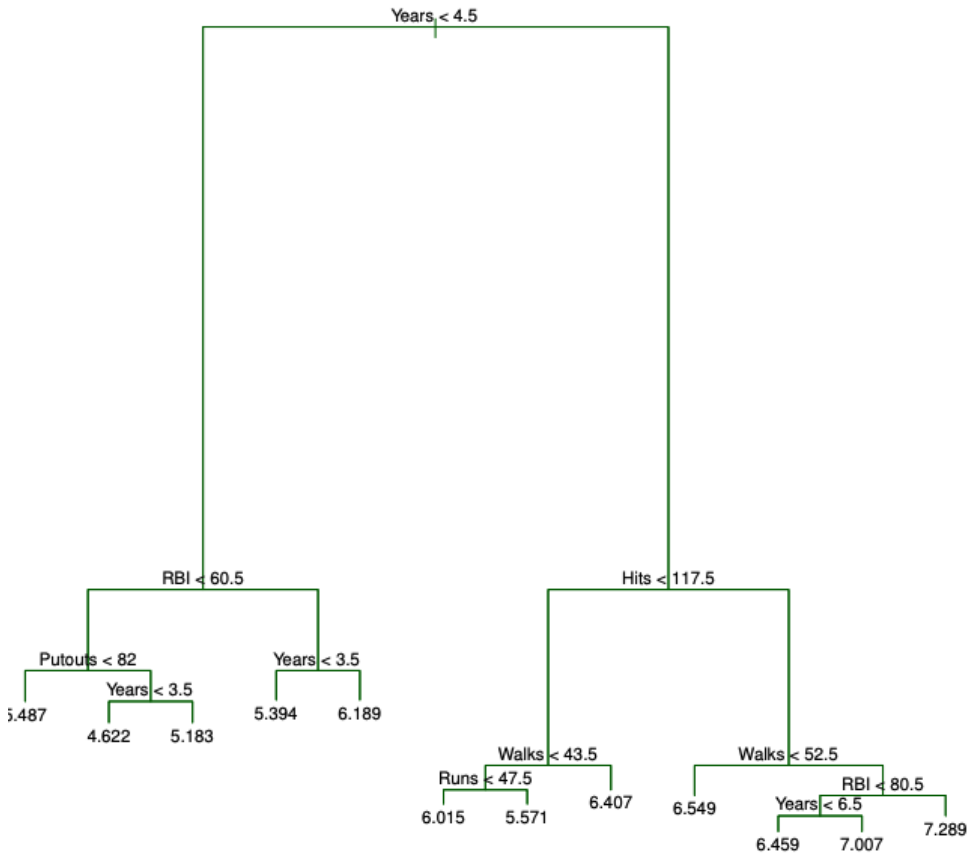
How does it work? Further Steps

- Continue the process until some stopping criterion is reached.
- An example of a stopping criterion might be that no region should contain more than 5 observations.

Tree Pruning

- The above process likely overfits, leading to poor test set performance. The reason might be that the tree is too complex.
- A smaller tree with fewer splits (fewer regions) might have lead to "lower variance" and "better interpretation" at the cost of a bit more bias.
- What to do? Prune the tree! Grow a large tree, then prune / cut it back to obtain a subtree.
- Given a subtree, we can estimate its test error using e.g. the test set approach, or cross validation.

Unpruned Tree Example



ISLR figure 8.4.

Pruned Tree Example

- Using the test set / cross validation approach, we can select the tree that has the smallest error.
- In our example, it turns out the best number of nodes is 3. This is how we would get the tree shown previously:

What about Classification?

- Previously we have looked at regression trees.
- What if we have a qualitative response, rather than a quantitative one?
- For the regression tree, we predicted the mean response of the training observations.
- For classification trees, we predict that each observation belongs to the *most commonly occurring* class of observations.

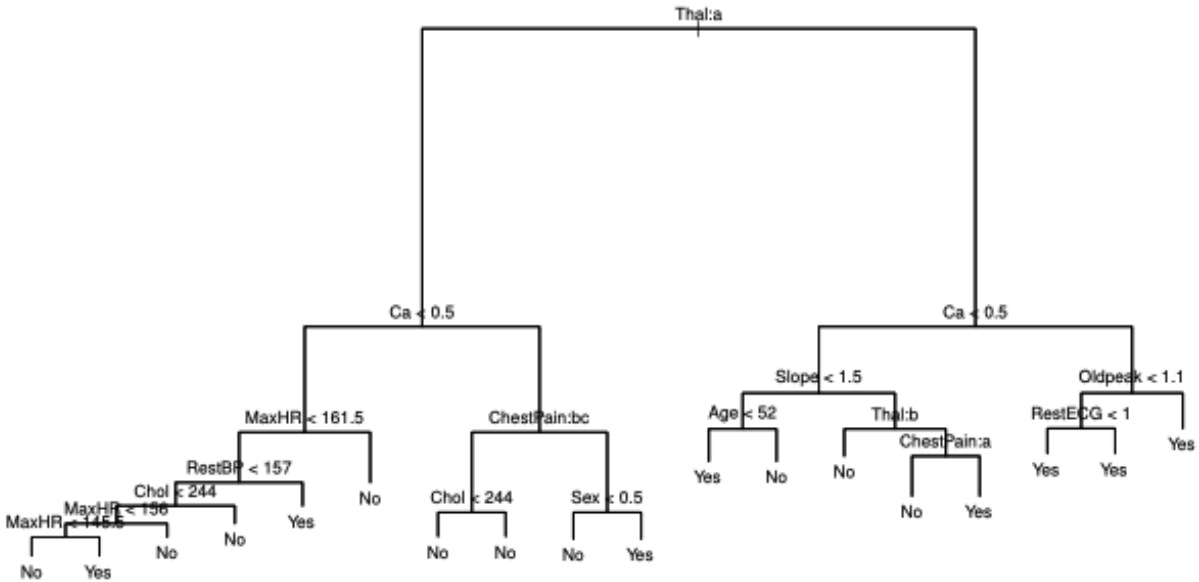
How to fit Classification Trees?

- We use the same approach to splitting (regursive binary splitting).
- However, we cannot use the RSS.
- Idea: Use Classification error rate - Fraction of training observations in each region that do not belong to the most common class (i.e. fraction we would get wrong if we were to predict the most common class).
- Often used measures: Gini index, entropy (see previous lectures). Why? Gini index and entropy will take on a small value if there is "little diversity" in each region.

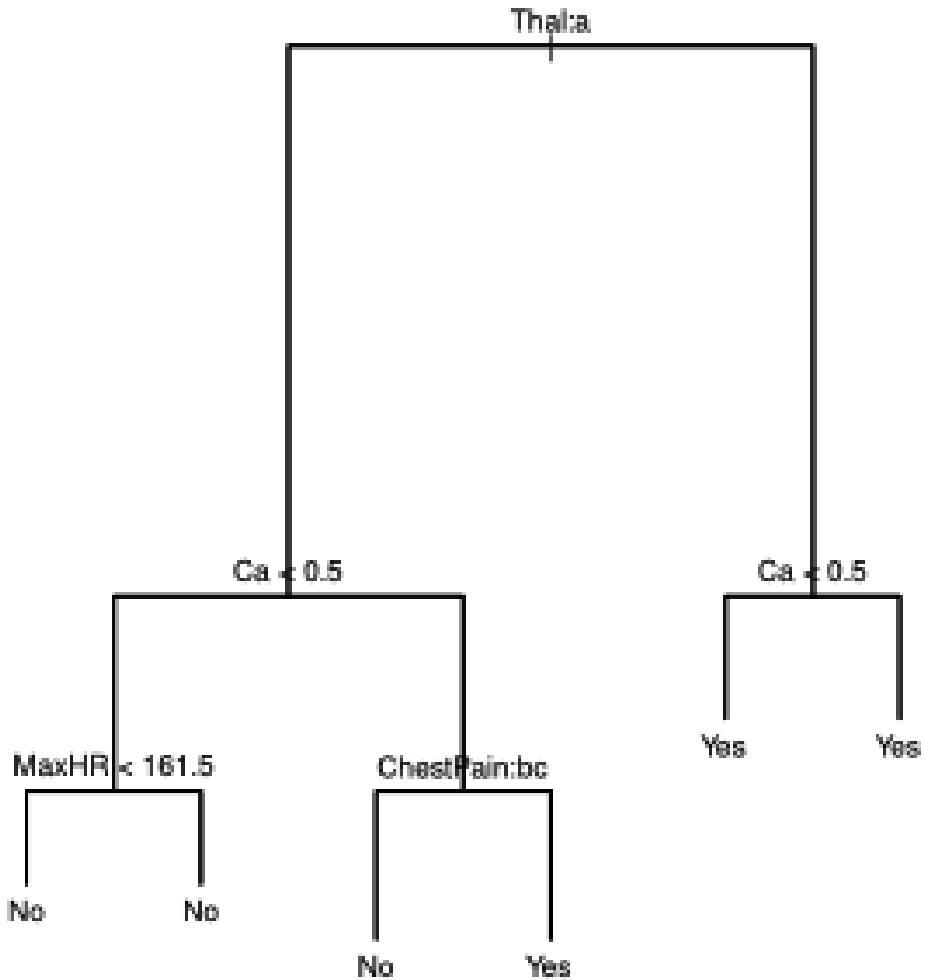
Example

- Data on binary outcome of heart disease. "Yes" means presence of heart disease.
- There are 13 predictors, such as age, sex, chol (cholesterol), etc.
- Build tree, then prune back (use test set / cross-validation).

Example


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Slope < 1.5
Oldpeak < 1.1
Age < 52
Thal:b
RestECG < 1
Yes
RestBP < 157
Chol < 244
Chol < 244
Sex < 0.5
No
ChestPain:a
No
No
Yes
No
Yes
Yes
Yes
MaxHR < 156
MaxHR < 145.5
No
No
No
No
No
No
No
Yes
No
Yes

Example: Pruned



Notes

- So far we have focused on quantitative predictors, but it is also possible to use qualitative predictors. In fact, we see that the tree splits on both quantitative and qualitative variables.
- Some of the splits yield two terminal nodes with the same predicted value! Why do we split then?
- We would split because it leads to "less diversity" in each node. Even though the split would not reduce the classification error, it would improve the Gini index or entropy (which are more sensitive to "diversity").

Example

- Consider the split on RestECG < 1. A response of "Yes" is predicted for both.
- All 9 observations in the right-hand leaf have response "Yes" and $7/11$ in the left-hand leaf have a response of "Yes".
- Why do we care about this? If a test observation belongs in the right-hand leaf, then we can be pretty certain that the response is "Yes". In contrast, if the test observation belongs to the left-hand leaf, then the response is probably "Yes", but we are less certain.

Using Linear Models or Trees?

- It depends on the problem ...
- If the relationship between the feature and the response is well approximated by linearity, then linear regression will likely work well and might be better than a regression tree (which does not exploit linear structure).
- If there is a highly nonlinear and complex relationship between the features and the response as indicated by model, then decision trees may outperform classical approaches.
- Assess the relative performance of trees or classical approaches using the validation set approach, or cross validation.

Example

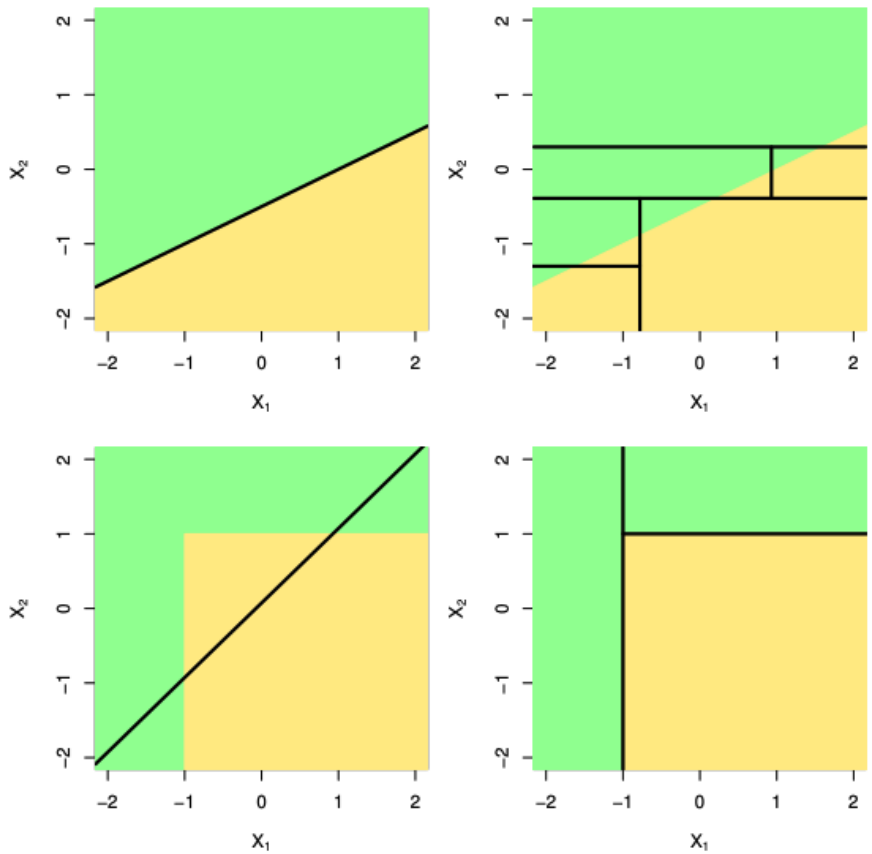


Figure 8.7 ISLR. Top row: True decision boundary linear (linear methods better than tree). Bottom row: Decision boundary is non-linear (tree would do better).

Trees: Advantages

- Easy to understand! Even easier than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than other approaches we've seen.
- Can be displayed graphically and interpreted by a non-expert.
- Can easily handle qualitative predictors (without dummy variables).

Tree: Disadvantages

- Trees generally do not have the same level of predictive accuracy as some other regression and classification approaches.
- 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.

Bagging

- The bootstrap can be used to improve decision trees!
- Bagging stands for "Bootstrap Aggregation".
- Remember that if we have a set of n independent observations, $X_1, ..., X_n$, each with variance σ^2, then the variance of the mean $\bar{X}$ is given by σ^2/n.
- What does this say? If we average a set of observations, then it reduces variance.

Bagging

- Natural idea to reduce variance and hence increase test accuracy: Take many training sets from the population, build a separate prediction model for each training set, then average the resulting predictors.
- This means: Calculate the predictions $\hat{f}^1(x), ..., \hat{f}^B(x)$ for B separate trainign sets, then average them and obtain

$$\hat{f}_{avg}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

Bagging

- Of course, we don't generally have access to multiple training sets.
- Instead: Apply the bootstrap, o.e. generate B different bootstrapped training set. Then train model on *bth* bootstrapped training set to get $\hat{f}^{*b}(x)$, then average the predictions:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

- This is called bagging.

Bagging

- Bagging can improve predictions ofr many regression methods, but it is particularly useful for decision trees.
- Applied to trees: Construct B regression trees using B botstrapped training sets, and average the predictions. .
- These trees are not pruned Each individual tree has high variance, low bias. But averaging reduces the variance.
- What if we have a classification problem (Y is qualitative)? Could for example just take the majority vote: The overall prediction that is most commonly occuring among the B predictions.

What is a random forest?

- As in bagging, we build a number of trees on bootstrapped training samples.
- However, when building these trees, each time a split is considered, **only a random sample of m predictors** is chosen as potential split candidates instead of choosing from the full set of p predictors.
- The split is allowed to use only one of those m predictors.
- Typically use $m \approx \sqrt{p}$.

Why does a random forest make sense?

- At each split, we are not allowed to consider most of available predictors.
- Bagged trees are very similar to each other. Why?
- Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.
- In the collection of bagged trees, most or all of the trees will use this strong predictor in the top split.
- Hence, the bagged trees will be highly correlated.

Why does a random forest make sense?

- It turns out that if we average highly correlated quantities, then this does not lead to as a large reduction in variance as uncorrelated quantities.
- Random forest overcome the problem by forcing each split to consider only a subset of the predictors.
- This decorrelates the trees (make the average of the resulting trees more reliable).

Random Forest vs. Bagging

- Main difference: Bagging allows to choose from p predictors at each split.
- Random forests only allows to choose from m predictors. If $m = p$, then this is the same as bagging.

Boosting

- Another approach for improving predictions! We focus on boosting for decision trees.
- Recall bagging: Create multiple copies of the original training data set using the bootstrap then fit a separate decision tree to each copy and combine all of them together to create a single model
- Each tree is build on a bootstrap dataset, independent of the other trees.
- In boosting works in a similar way, trees are grown sequentially: Each tree is grown using information from previously grown trees!
- Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.

Boosting

- Instead of fitting a single large decision tree to the data, boosting learns slowly:
- Given the current model, we fit a tree to the residuals of the model.
- That is, we fit a tree using the current residuals, rather than the outcome Y, as the response.
- Each of these trees can be small, say just d splits

Boosting

- By fitting small trees to the residuals, we slowly improve the fit in areas where it does not perform well.
- Then add this new decision tree into the fitted function to "update the residuals". We can also add this decision tree with "shrinkage".