Tree-based methods (Chapter 8, ISLR)

https://madina-k.github.io/dse_mk2021

Tree-based methods

Tree-based methods are very popular among ML practitioners because:

- they help to **partition**/**segment** the predictor space
- ... for both **regression** (i.e., continuous target variables) and **classification** (categorical target variables)
- and they do not require normalizing the predictors before fitting

Decision trees vs other tree-based methods

- Tree-based methods include random forests, bagged trees, and boosted trees, which are all based on the core method – the decision tree.
- Decision trees are easy to interpret, but they usually perform worse than Ridge or Lasso in prediction
- Other tree-based methods combine many decision trees: they become less interpretable, but they gain in prediction precision

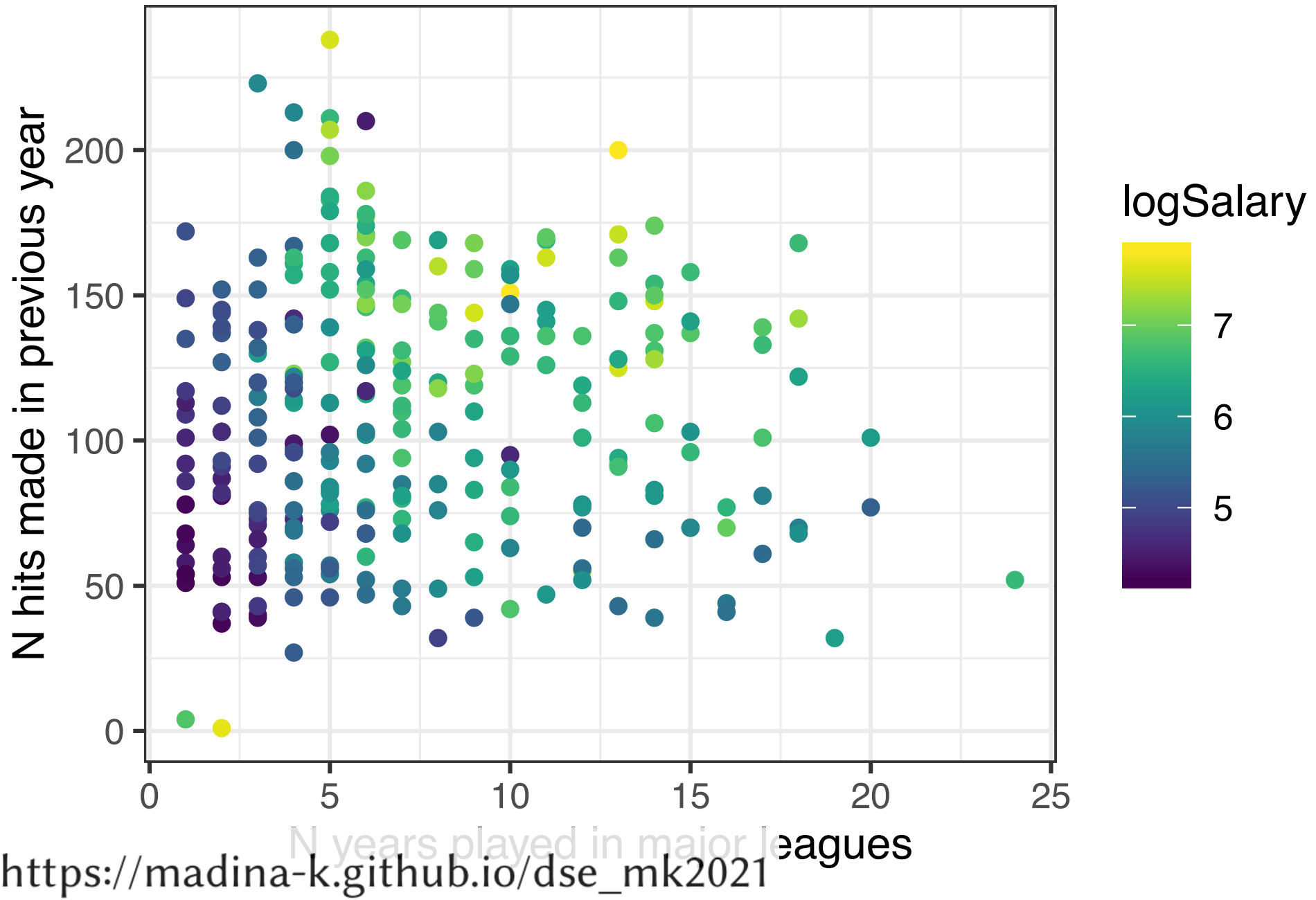
Example: baseball players' salary

Suppose we want to predict baseball players' salary based on the information on:

- Number of years the player played in major baseball leagues
- Number of hits he made in the previous year

Let's visualize the log-transformation of the salary against the number of years and number of hits. (You can access the hitters dataset yourself in the folder with tutorials.)

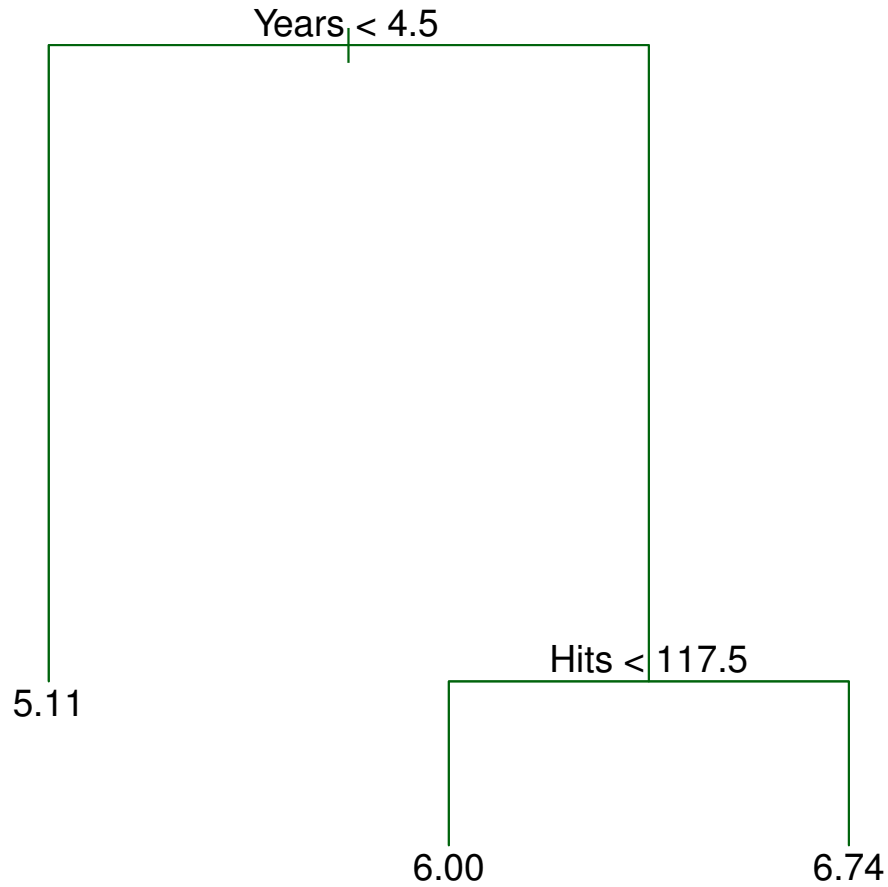
Baseball players' salary plot: Color indicates log(Salary)



https://madina-k.github.io/dse_mk2021

Baseball players' salary: Decision tree

If we train a decision tree to predict the log(Salary) based on the player's characteristics, this is an example of tree that we may get:



Baseball players' salary: Decision tree explanation

How to read that decision tree?

Take any baseball player:

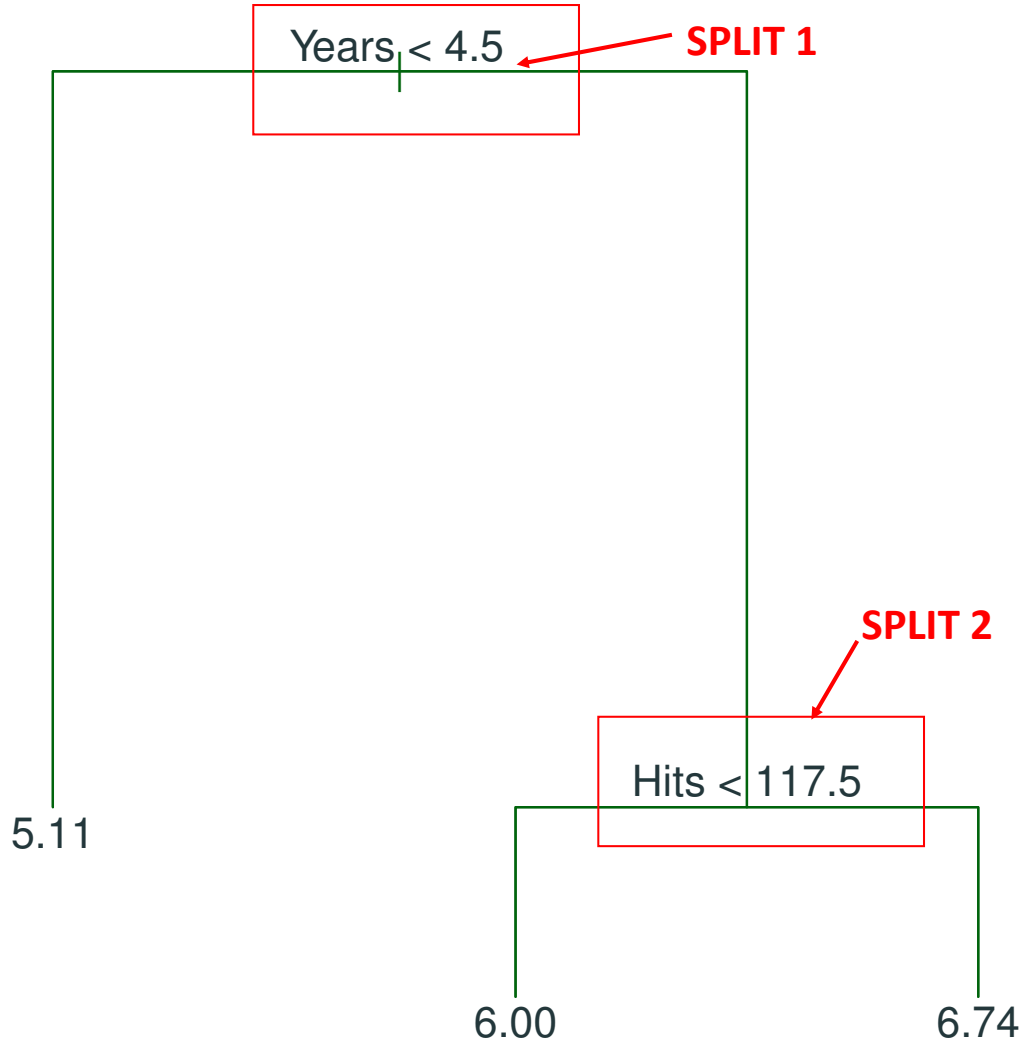
1. Rule 1: Is the number of years below 4.5?
    - If yes, then predict his log(Salary) to be 5.11
    - If no, then continue to Rule 2
2. Rule 2: Is the number of hits below 117.5?
    - If yes, then predict log(Salary) to be 6.00
    - If no, then predict log(Salary) to be 6.74

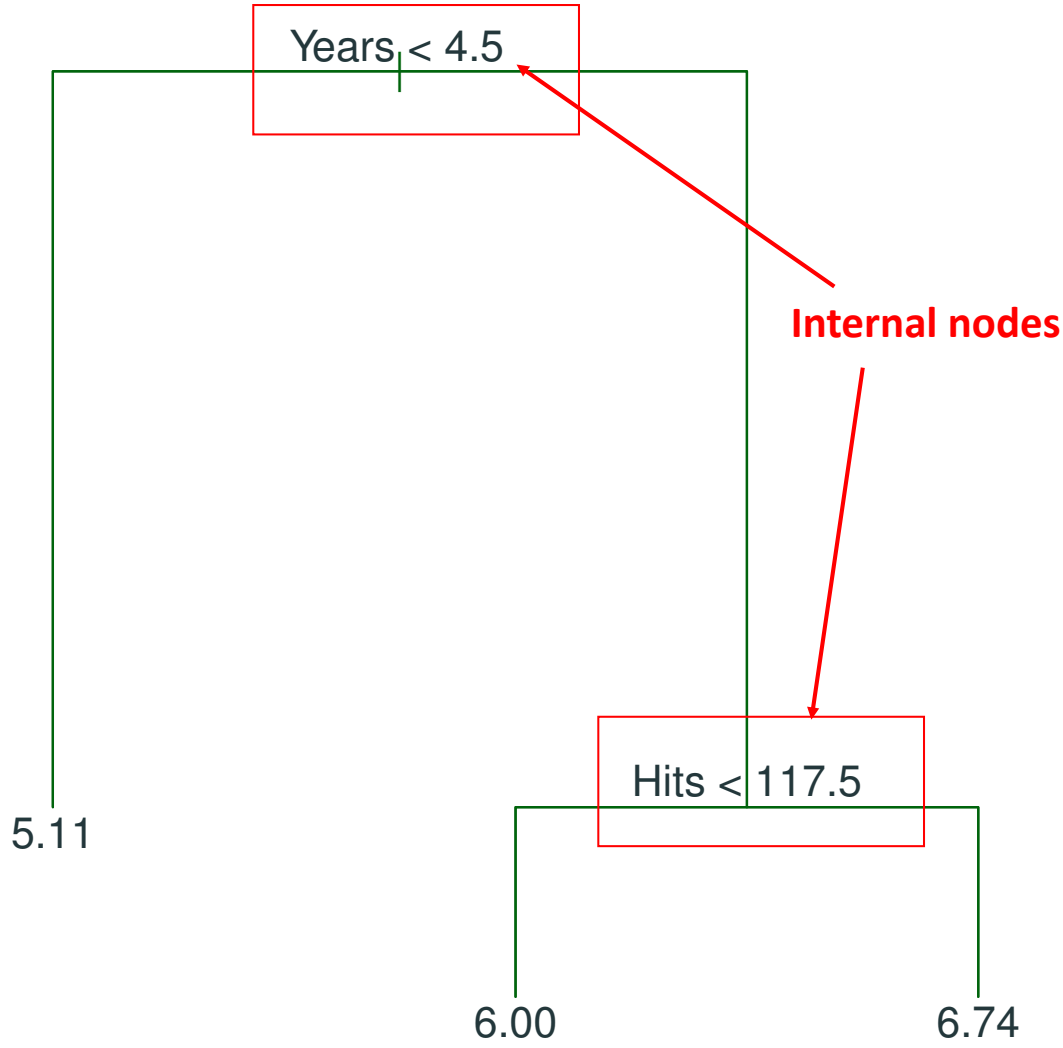
Baseball players' salary: Decision tree components

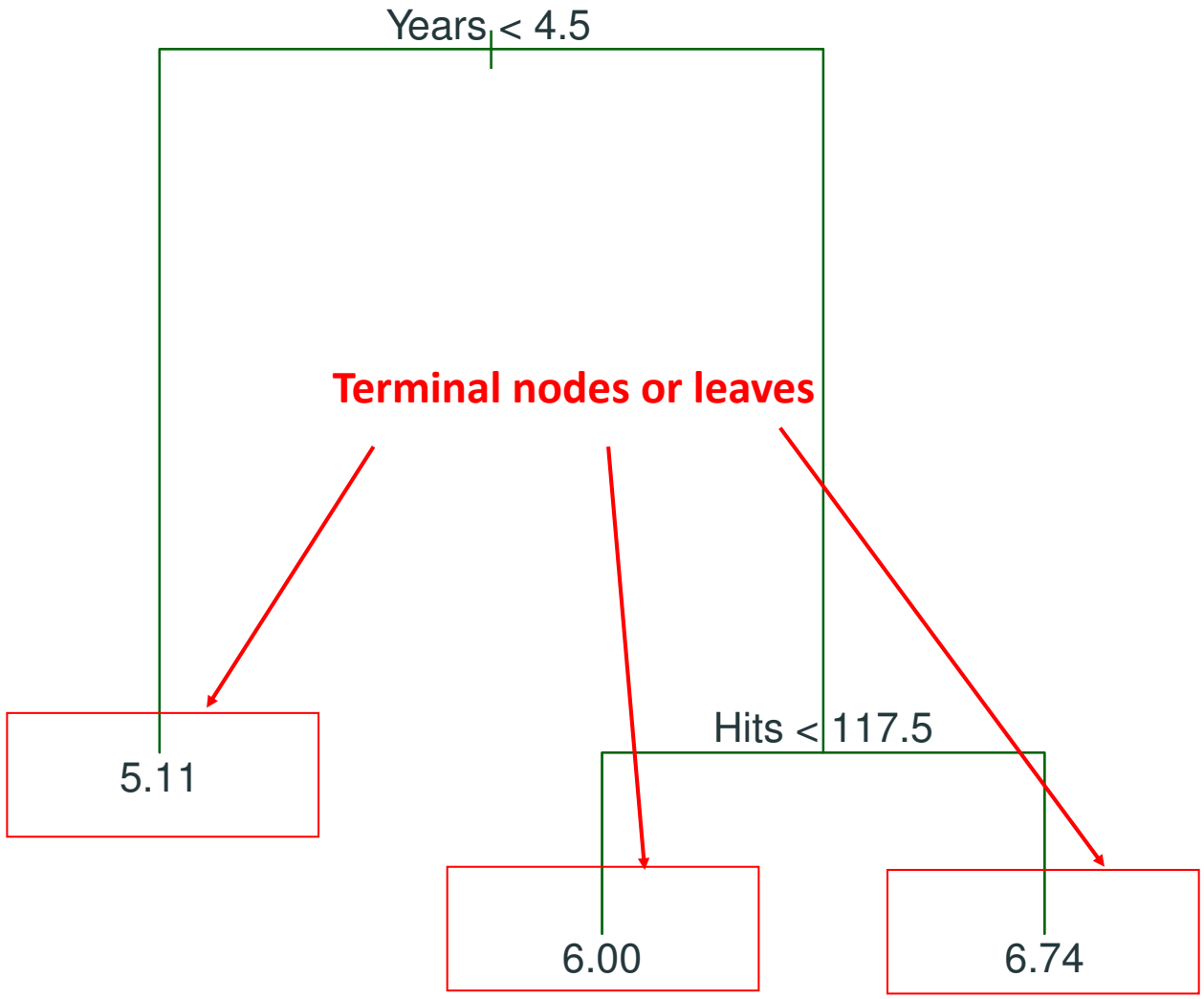
As you can see, this decision tree is made of:

- Two decision rules (or two splits)
- Three terminal nodes (or three leaves)

Each terminal node contains a **prediction** value which is based on the **mean of the target value** for the players in the dataset who **belong to that terminal node**


Years < 4.5
SPLIT 1
SPLIT 2
Hits < 117.5
5.11
6.00
6.74


Years < 4.5
Internal nodes
Hits < 117.5
5.11
6.00
6.74

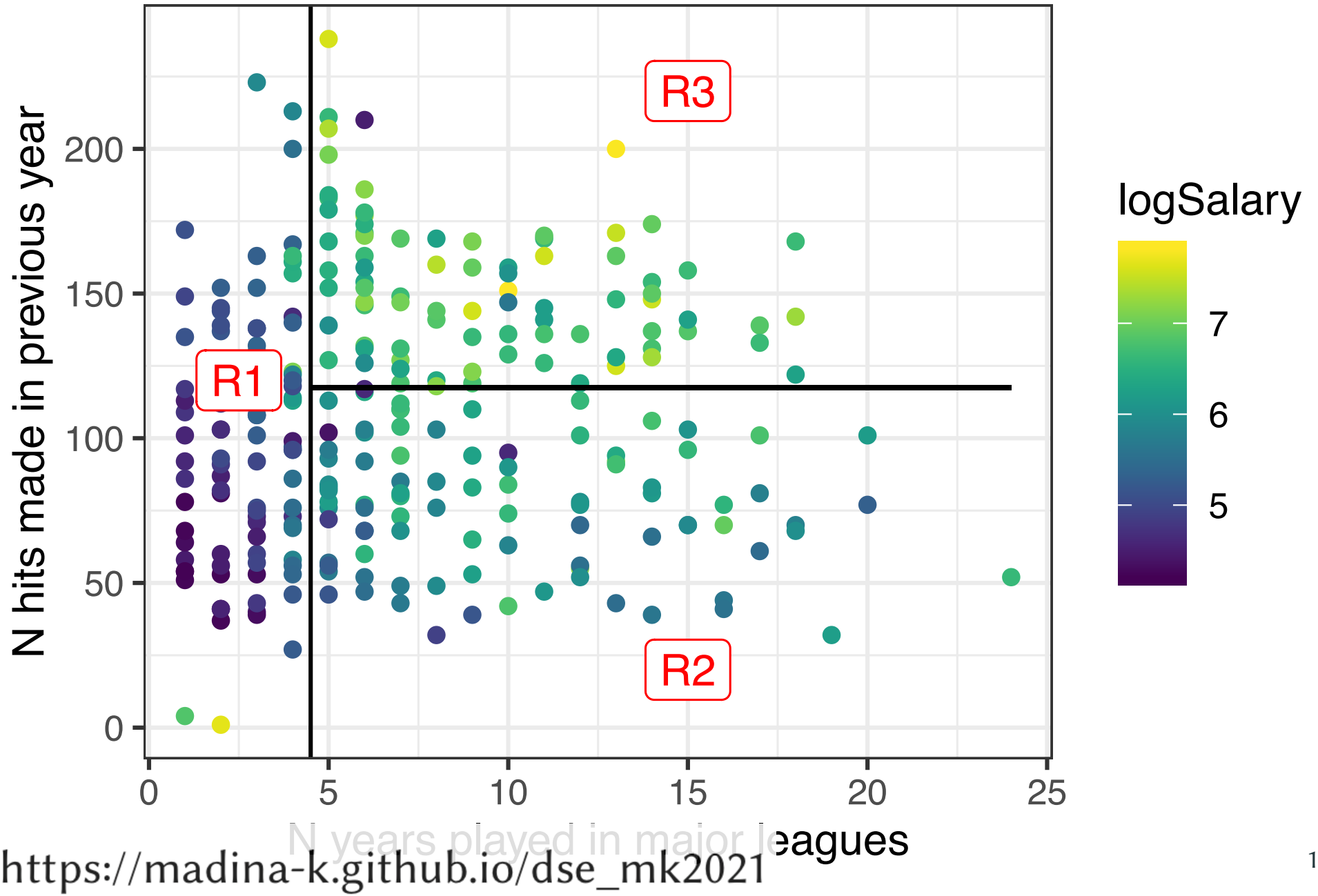

Years < 4.5
Terminal nodes or leaves
Hits < 117.5
5.11
6.00
6.74

Baseball players' salary: predictor space stratification

Overall, the decision tree segments the predictor space into three regions (terminal nodes, leaves):

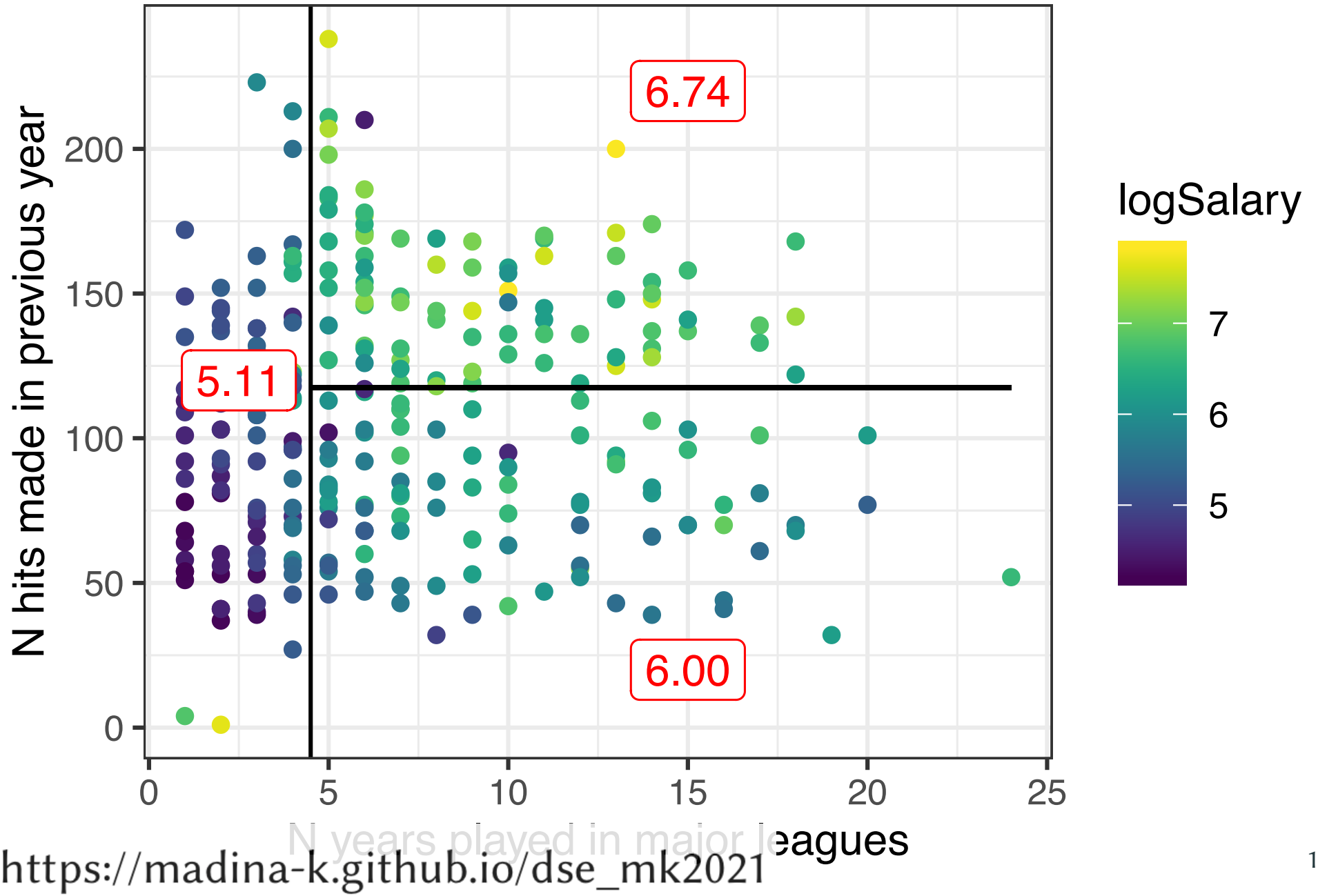
1. $R_1 = \{X | \text{Years} < 4.5\}$
   (All players who have played for four or less years)
2. $R_2 = \{X | \text{Years} \geq 4.5, \text{Hits} < 117.5\}$
   (All players who have played at least for five years and who made fewer than 118 hits last year)
3. $R_3 = \{X | \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$
   (All players who have played at least for five years and who made fewer than 118 hits last year)

Baseball players' salary: regions R_j, $j \in \{1, 2, 3\}$



https://madina-k.github.io/dse_mk2021

Baseball players' salary: predictions $\hat{y}_{R_j}, j \in \{1, 2, 3\}$



https://madina-k.github.io/dse_mk2021

Prediction

Hence, for any new player we predict his log-salary depending on which region (R_1, R_2, or R_3) he belongs to, i.e.,:

$$\hat{y}_i(X_i) = \hat{y}_{R_j} \text{ such that } X_i \in R_j, j \in \{1, 2, 3\} \tag{1}$$

How to build a regression tree?

We build a regression tree in two stages:

1. We decide how to split the predictor space into regions
2. For each region we estimate the mean of the response values for the training observations which fall into that region

Stage 1: How do we decide to split data into regions?

The goal is to find the best possible partition of X-space into $R_1, \ldots, R_J$ that minimizes the RSS:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \underbrace{\hat{y}_{R_j}}_{\text{prediction for } R_j})^2 \tag{2}$$

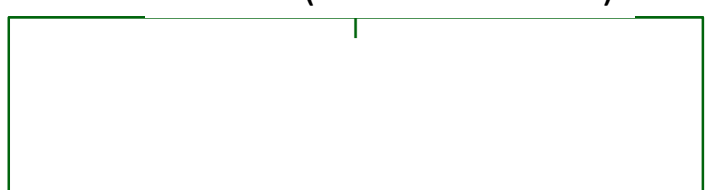
We search for the best partition using **recursive binary splitting.**

Recursive binary splitting

- First allocate all observations in a single node
- Then we successively split the predictor space in that single node:
    1. For each predictor j find the cutpoint s, such that the split into two regions $X|X_j < s$ and $X|X_j \geq s$ leads to the largest drop in RSS
    2. Actually split the observations using the best predictor (and its best cutpoint) that gives the largest drop in RSS
    3. Now we have two nodes
- Continue splitting observations at each node until a stopping criteria is reached

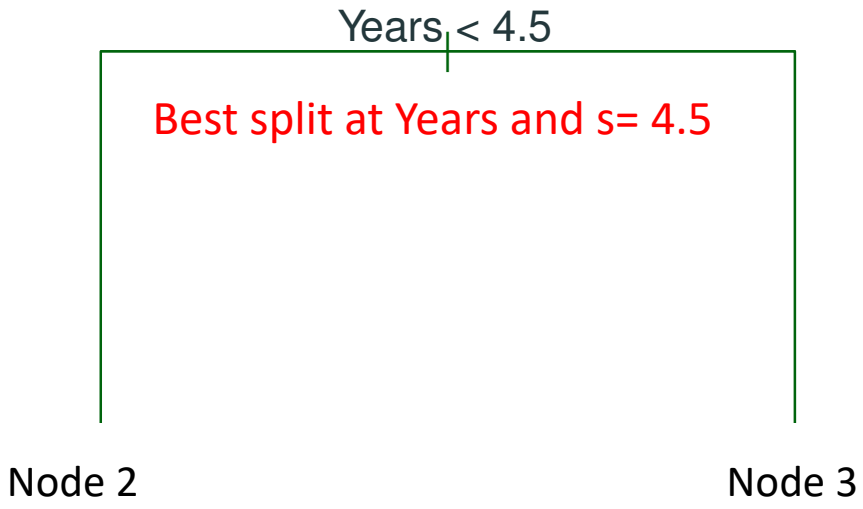
Let's return back to our example.

STEP 1

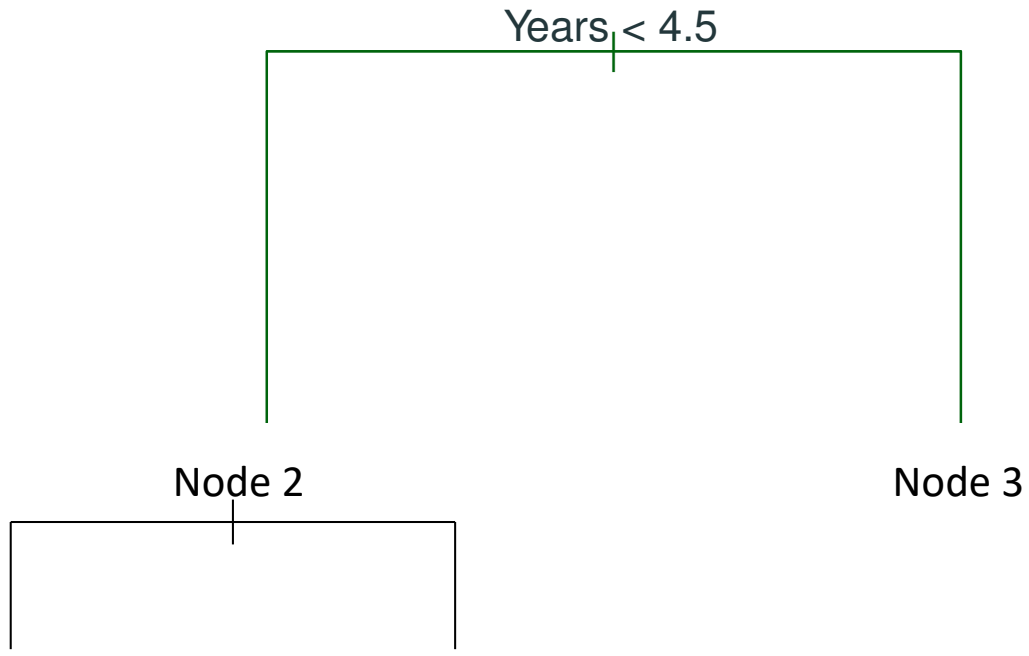


Best split at Years or at Hits? At which cutpoint?

STEP 1

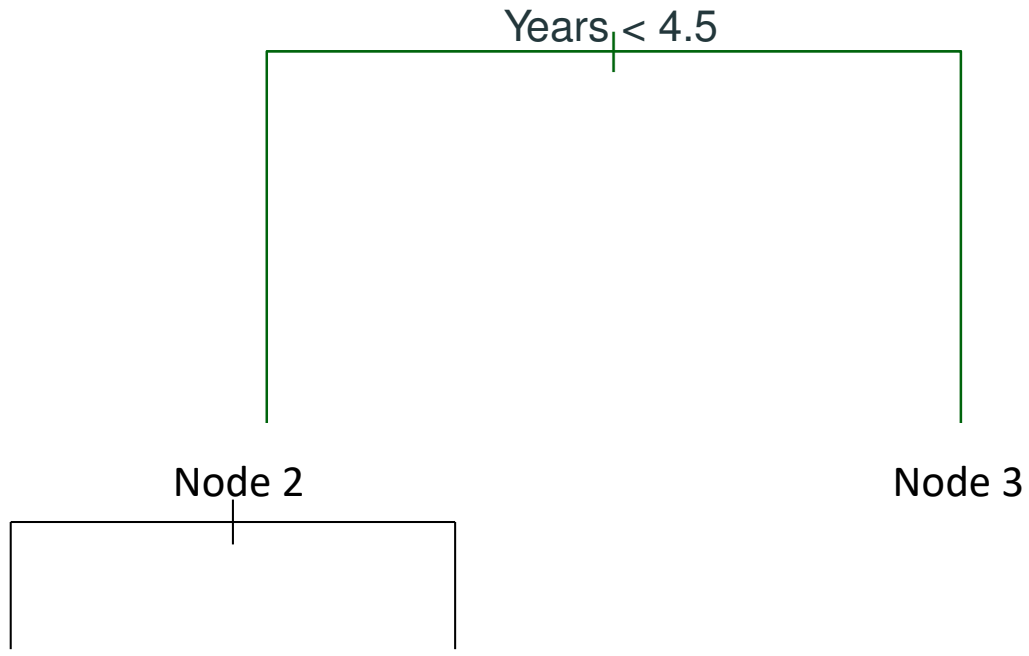


STEP 2



Best split at Years or at Hits? At which cutpoint?

STEP 2

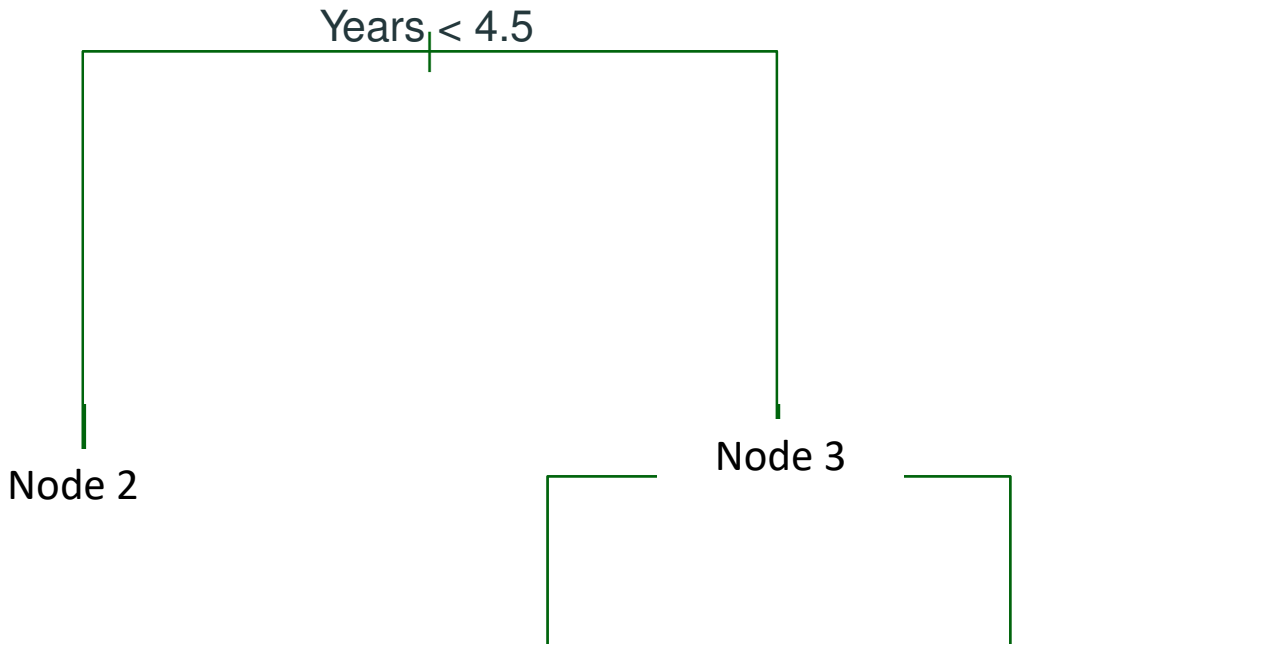


The best split does not pass through the stopping criteria, Node 2 becomes a leaf

STEP 2

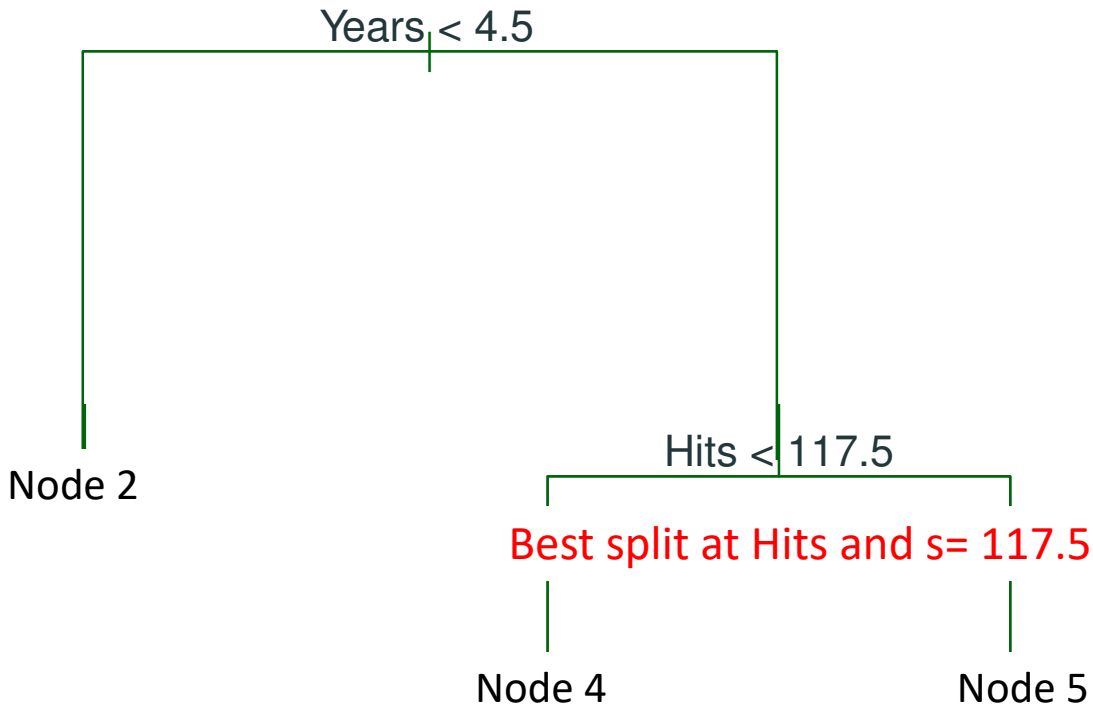
Terminal node

STEP 3

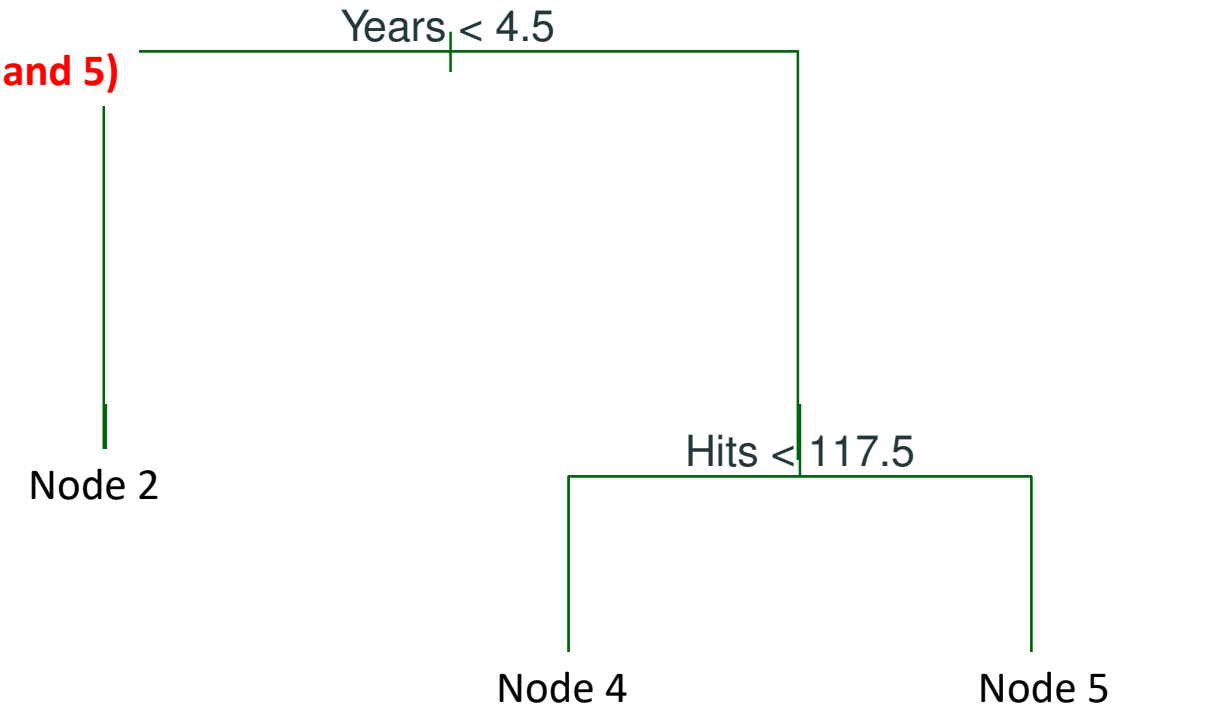


Best split at Years or at Hits? At which cutpoint?

STEP 3


Years < 4.5
Node 2
Hits < 117.5
Best split at Hits and s= 117.5
Node 4
Node 5

STEP 4 and 5
(repeat for node 4 and 5)



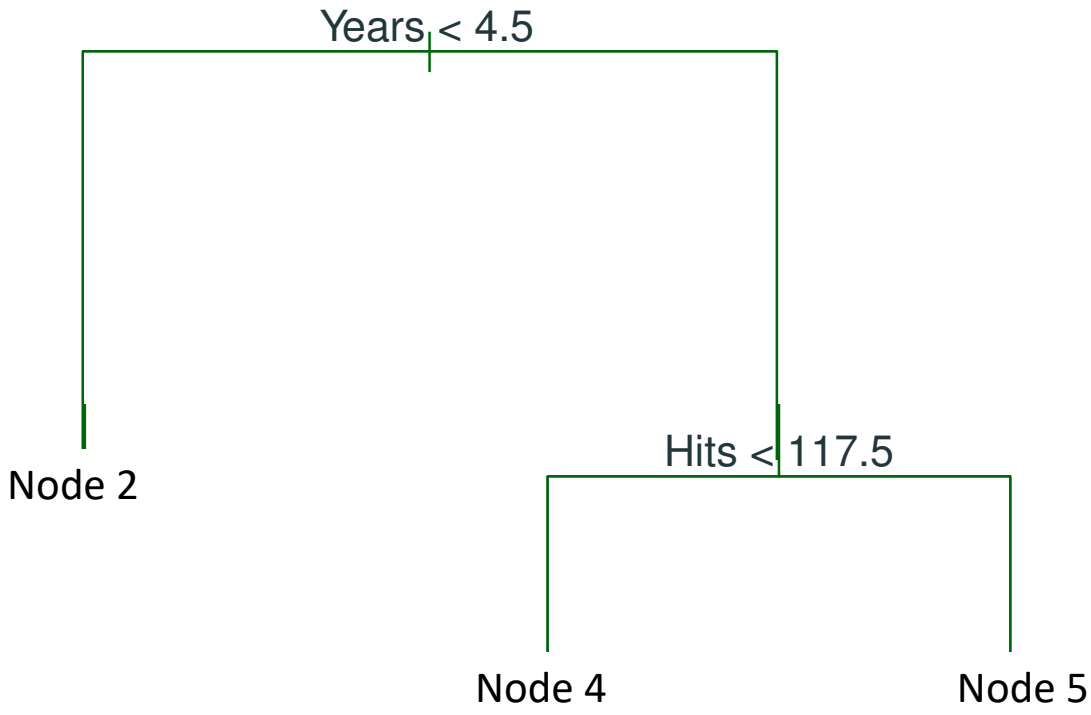
Best split at Years or at Hits? At which cutpoint?

Best split at Years or at Hits? At which cutpoint?

STEP 4 and 5
(repeat for node 4 and 5)

Node 4
Terminal node
(stopping criteria reached)

Node 5
Terminal node
(stopping criteria reached)



All end nodes are now terminal, algorithm stops

Final stage: calculate predictions

Finally, for each terminal node calculate the mean response value (target variable) for the observations in the training sample which fall into that terminal node.

Overfitting problem

In principle, we can grow our tree until each observation is in its own terminal node, i.e., number of leaves = number of observations. Clearly, the in-sample RSS is going to be zero: **Perfect in-sample fit**!

However, such a tree is likely **overfitting** and going to be very bad in predicting out-of-sample for some new observations.

What to do? We should **prune the tree**!

Tree pruning

Strategy:

- Grow a very large tree with a mild stopping criteria (e.g., min n obs per leaf > 5)
- Prune it to different subtrees
- Choose the pruned subtree that gives the lowest cross-validated error (e.g., through **cost-complexity pruning**)

Cost complexity pruning

Modify objective function: now we want to minimize SSR and penalty for complexity by subsetting our large tree

$$\sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \tag{3}$$

where $|T|$ is the number of terminal nodes.

- If $\alpha = 0$, there is no penalty on the tree complexity (our tree remains as large as we grew it)
- If α is very large, then we chop our tree to just one node

Cross-validation

In other words, α creates a trade-off between in-sample fit and the penalty for complexity (just like the λ parameter in Lasso and Ridge).

Hence, just like in Lasso and Ridge, we can apply **cross-validation** to choose optimal α that gives the lowest expected MSE.

Regression trees: summary

To sum up, the steps to get the regression tree:

1. Use recursive splitting to grow large tree
2. Create grid of α and apply cost complexity pruning for each value of α
3. Use K-fold cross-validation to find optimal α
4. Return the subtree that corresponds to the optimal α

https://madina-k.github.io/dse_mk2021

Classification trees

Growing classification trees is similar to regression trees, but the objective function and the prediction formula is, of course, different:

The prediction formula:

- for any observation in R_m the prediction is simply the most commonly occurring class of training observations in that region.
- If $\hat{p}_{mk}$ is the proportion of obs. in the mth region from the kth class
- then the prediction is $max_k(\hat{p}_{mk})$

Which objective function?

There are different objective functions that we can choose to minimize:

1. Classification error: $min \sum_{j=1}^{J} E_j$

$$E = 1 - max_k(\hat{p}_{mk}) \tag{4}$$

2. or Gini index: $min \sum_{j=1}^{J} G_j$

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{5}$$

3. or Entropy: $min \sum_{j=1}^{J} D_j$

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk}) \tag{6}$$

Classification error, Gini index, or Entropy?

Which one is better?

- G and D are usually numerically similar
- G and D are usually “better”, i. e., more sensitive to the purity of the node, than E

Pros and Cons of decision trees (vs linear models)

Pros:

+ Results are easy to interpret
+ Many business applications require segmentations (e.g., customer categories)
+ No need to standardize predictors
+ No need to create dummies for categorical variables

Cons:

- Predictions based on a decision tree are not that accurate, because...
- ...not very robust to small variations in a training sample (high variance!)

Bagging and **boosting** help to improve the accuracy and robustness at the expense of interpretability.