Machine learning

Decision trees

July 12th, 2021

What is Machine Learning?

The short version:

- Machine learning (ML) is a subset of statistical learning that focuses on prediction

The longer version:

- ML focuses on constructing data-driven algorithms that *learn* the mapping between predictor variables and response variable(s).
    - We do not assume a parametric form for the mapping *a priori*, even if technically one can write one down *a posteriori* (e.g., by translating a tree model to a indicator-variable mathematical expression)
    - e.g., linear regression is NOT considered a ML algorithm since we can write down the linear equation ahead of time
    - e.g., random forests are considered an ML algorithm since we have what the trees will look like in advance

Which algorithm is best?

That's not the right question to ask.

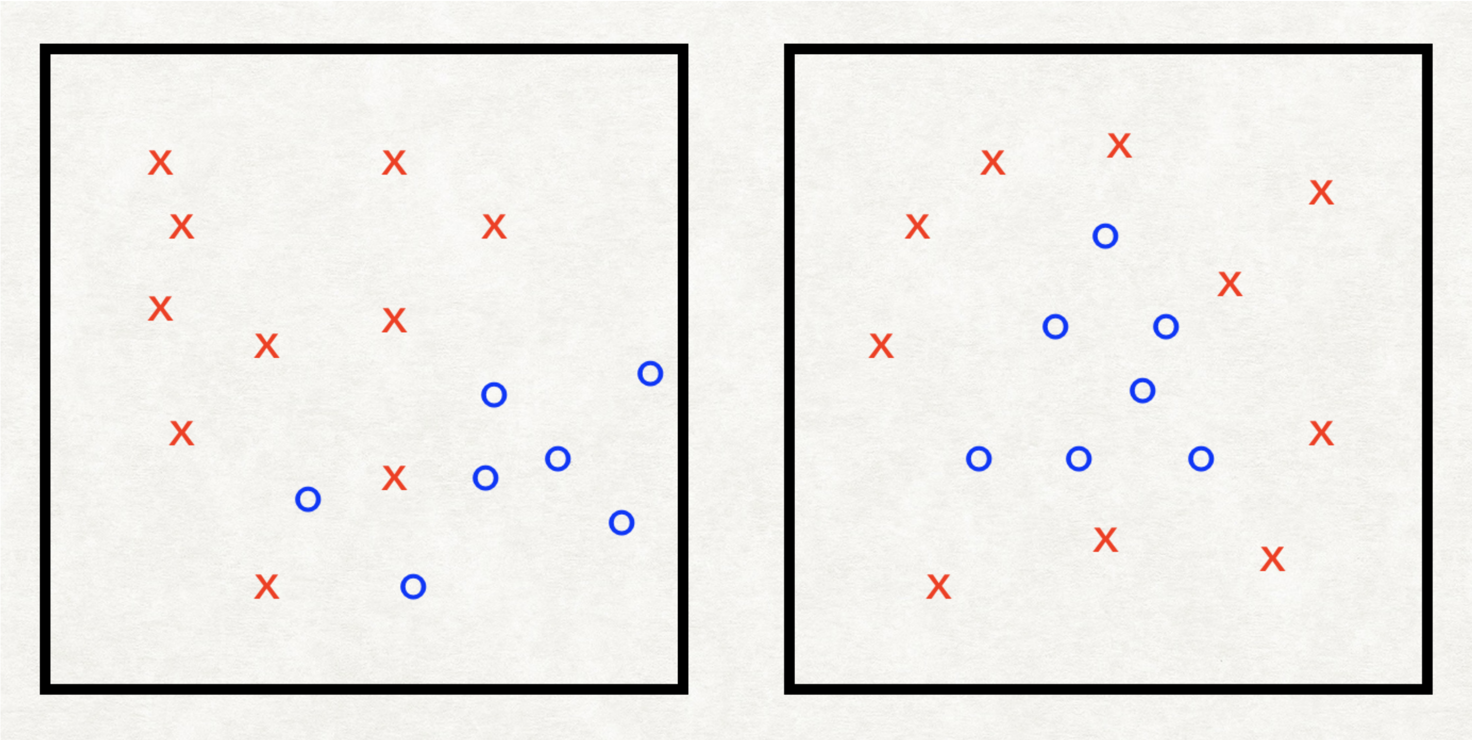
(And the answer is *not* deep learning. Because if the underlying relationship between your predictors and your response is truly linear, *you do not need to apply deep learning*! Just do linear regression. Really. It's OK.)

The right question is ask is: **why should I try different algorithms?**

The answer to that is that without superhuman powers, you cannot visualize the distribution of predictor variables in their native space.

- Of course, you can visualize these data *in projection*, for instance when we perform EDA
- And the performance of different algorithms will depend on how predictor data are distributed…

Data geometry



- Two predictor variables with binary response variable: x's and o's
- **LHS**: Linear boundaries that form rectangles will peform well in predicting response
- **RHS**: Circular boundaries will perform better

Decision trees

Decision trees partition training data into **homogenous nodes / subgroups** with similar response values

The subgroups are found **recursively using binary partitions**

- i.e. asking a series of yes-no questions about the predictor variables

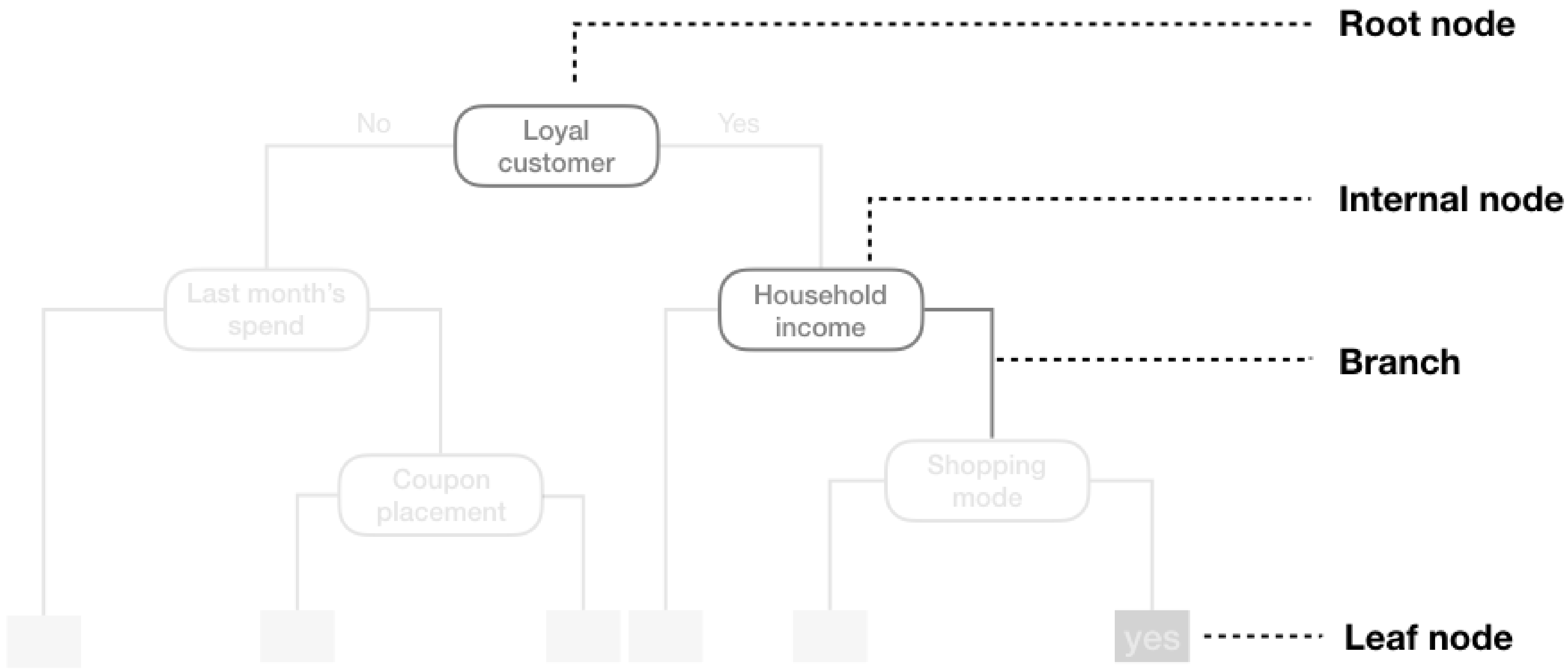
We stop splitting the tree once a **stopping criteria** has been reached (e.g. maximum depth allowed)

For each subgroup / node predictions are made with:

- Regression tree: **the average of the response values** in the node
- Classification tree: **the most popular class** in the node

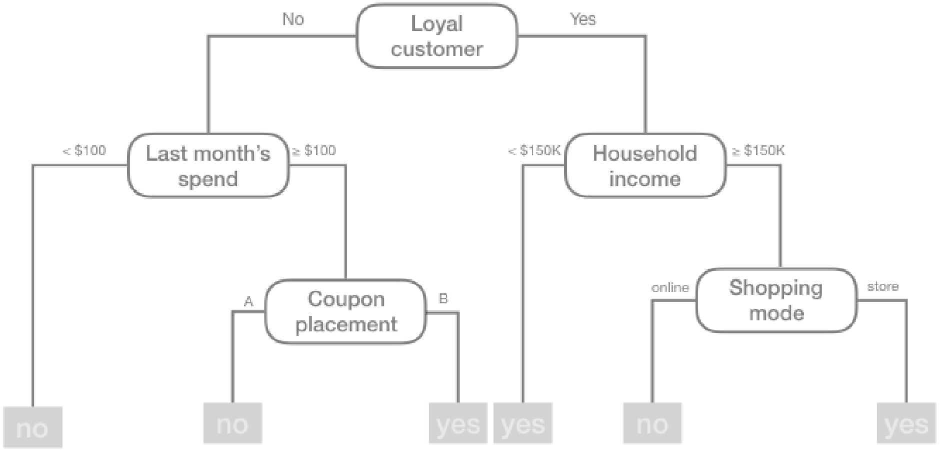
Most popular approach is Leo Breiman's **C**lassification **A**nd **R**egression **T**ree (CART) algorithm

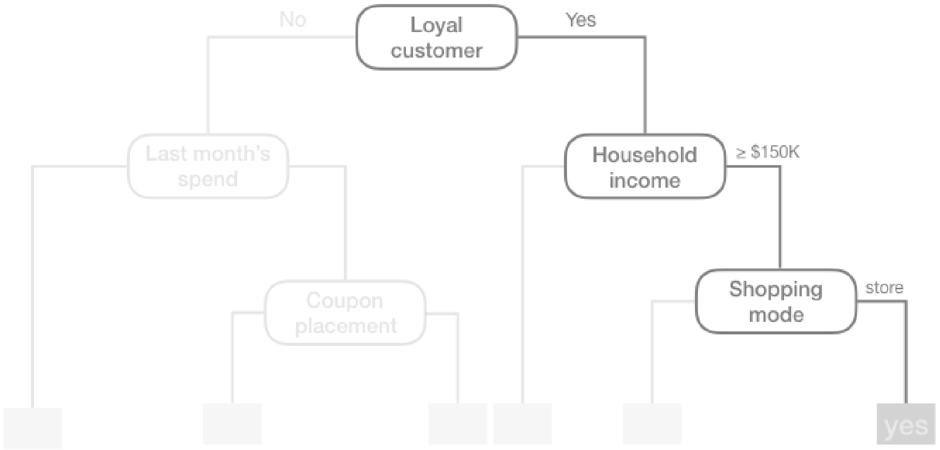
Decision tree structure



Decision tree structure

We make a prediction for an observation by **following its path along the tree**





- Decision trees are **very easy to explain** to non-statisticians.
- Easy to visualize and thus easy to interpret **without assuming a parametric form**

Recursive splits: each *split / rule* depends on previous split / rule *above* it

Objective at each split: find the **best** variable to partition the data into one of two regions, R_1 & R_2, to **minimize the error** between the actual response, y_i, and the node's predicted constant, c_i

- For regression we minimize the sum of squared errors (SSE):

$$SSE = \sum_{i \in R_1} (y_i - c_1)^2 + \sum_{i \in R_2} (y_i - c_2)^2$$

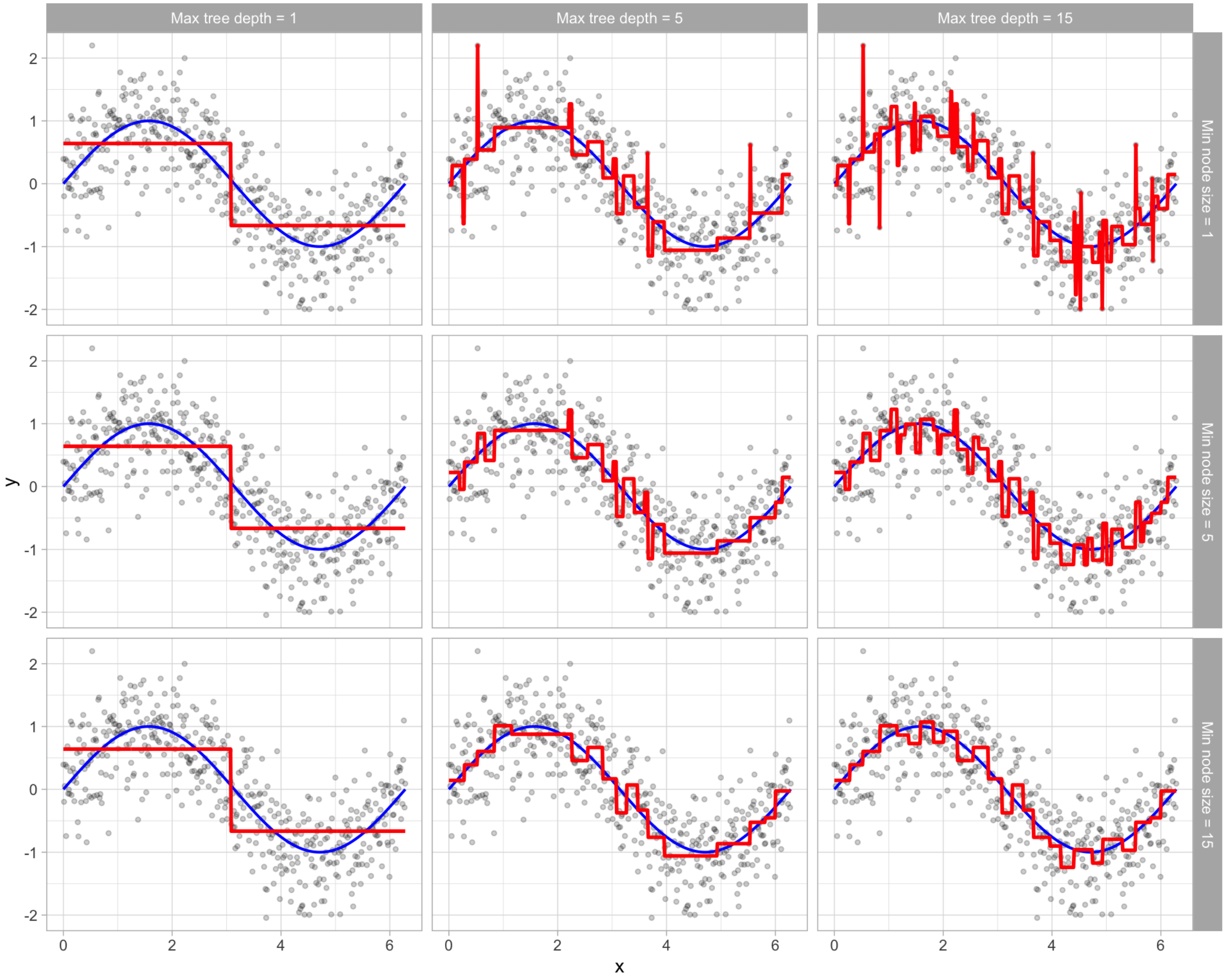
- For classification trees we minimize the node's *impurity* the **Gini index**
  - where p_k is the proportion of observations in the node belonging to class k out of K total classes
  - want to minimize $Gini$: small values indicate a node has primarily one class (*is more pure*)

$$Gini = 1 - \sum_k^K p_k^2$$

Splits yield **locally optimal** results, so we are NOT guaranteed to train a model that is globally optimal

How do we control the complexity of the tree?

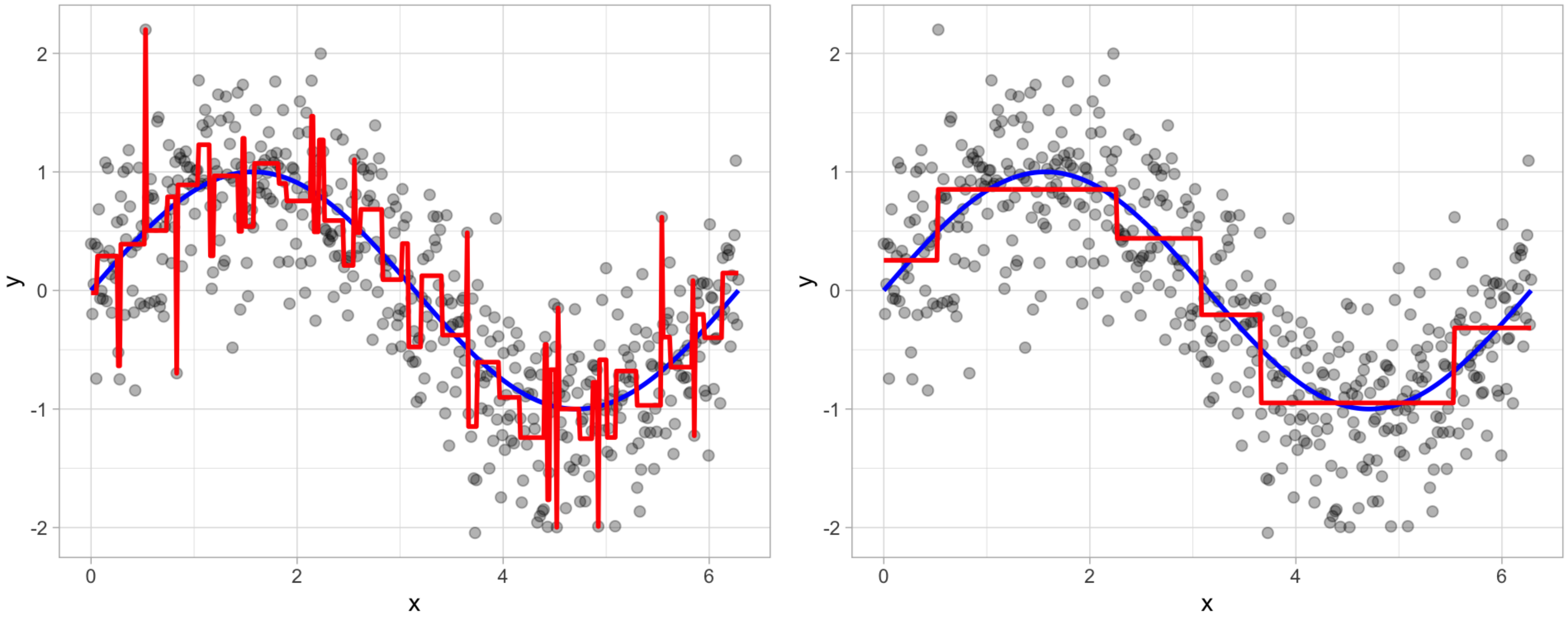
Tune the **maximum tree depth** or **minimum node size**



Prune the tree by tuning **cost complexity**

Can grow a very large complicated tree, and then **prune** back to an optimal **subtree** using a **cost complexity** parameter α (like λ for elastic net)

- α penalizes objective as a function of the number of **terminal nodes**
- e.g., we want to minimize $SSE + \alpha \cdot (\#$ of terminal nodes$)$



Example data: MLB 2021 batting statistics

Downloaded MLB 2021 batting statistics leaderboard from Fangraphs

```
library(tidyverse)
mlb_data <- read_csv("http://www.stat.cmu.edu/cmsac/sure/2021/materials/data/fg_batting_2021.csv'
  janitor::clean_names() %>%
  mutate_at(vars(bb_percent:k_percent), parse_number)
head(mlb_data)
```

```
## # A tibble: 6 × 23
##   name      team      g    pa    hr     r   rbi    sb bb_pe…¹ k_per…²   iso babip
##   <chr>     <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>   <dbl> <dbl> <dbl>
## 1 Vladimi… TOR      82   354    27    66    69     2    14.4    17.2 0.336 0.346
## 2 Fernand… SDP      68   288    27    66    58    18    12.5    28.1 0.395 0.333
## 3 Carlos … HOU      79   347    16    61    52     0    13.5    17   0.231 0.324
## 4 Marcus … TOR      82   372    21    63    54    10     8.9    23.9 0.256 0.329
## 5 Ronald … ATL      78   342    23    67    51    16    13.2    24.3 0.313 0.306
## 6 Shohei … LAA      82   322    31    60    67    12    11.2    28   0.418 0.29
## # … with 11 more variables: avg <dbl>, obp <dbl>, slg <dbl>, w_oba <dbl>,
## #   xw_oba <dbl>, w_rc <dbl>, bs_r <dbl>, off <dbl>, def <dbl>, war <dbl>,
## #   playerid <dbl>, and abbreviated variable names ¹bb_percent, ²k_percent
```

Regression tree example with the `rpart` package

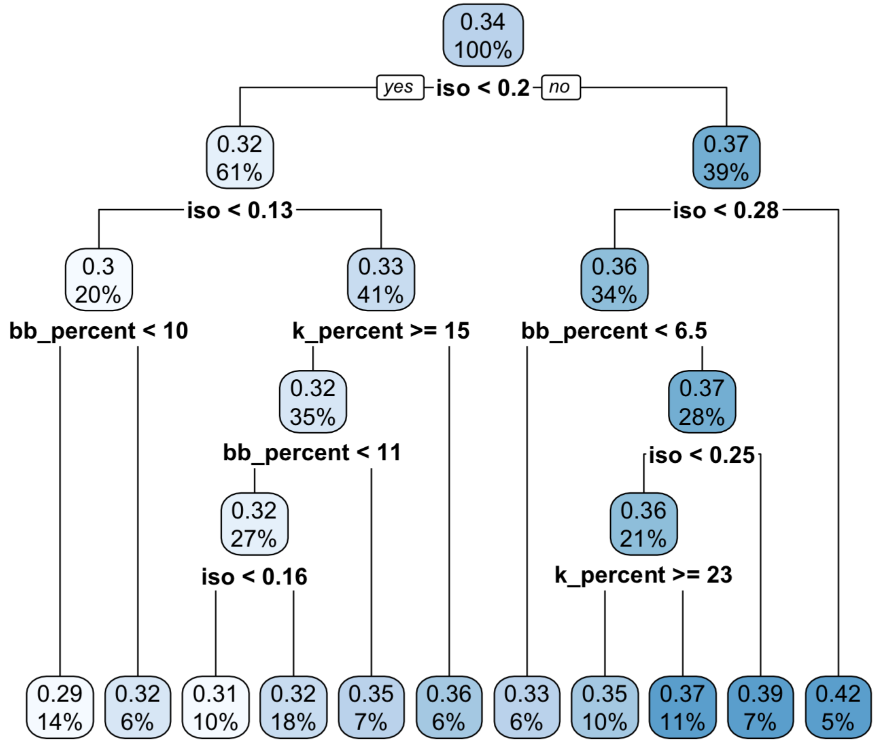
Revisit the modeling of `w_oba` from the KNN slides

```
library(rpart)
init_mlb_tree <- rpart(formula = w_oba ~ bb_percent + k_percent + iso,
                       data = mlb_data, method  = "anova")
init_mlb_tree
```

```
## n= 135
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 135 0.203847700 0.3383259
##    2) iso< 0.2 82 0.069028260 0.3188171
##      4) iso< 0.1315 27 0.021396070 0.2981852
##        8) bb_percent< 10.15 19 0.013706740 0.2894737 *
##        9) bb_percent>=10.15 8 0.002822875 0.3188750 *
##      5) iso>=0.1315 55 0.030496840 0.3289455
##       10) k_percent>=15.15 47 0.020832550 0.3243404
##         20) bb_percent< 11.45 37 0.012078700 0.3186216
##           40) iso< 0.1585 13 0.002205692 0.3071538 *
##           41) iso>=0.1585 24 0.007237333 0.3248333 *
```

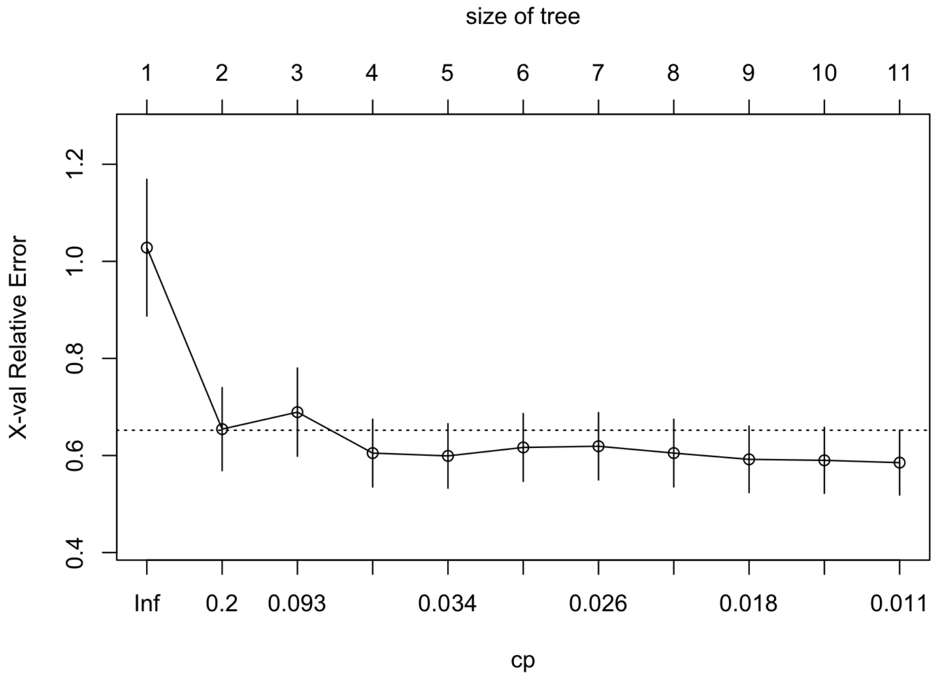
Display the tree with `rpart.plot`

```r
library(rpart.plot)
rpart.plot(init_mlb_tree)
```



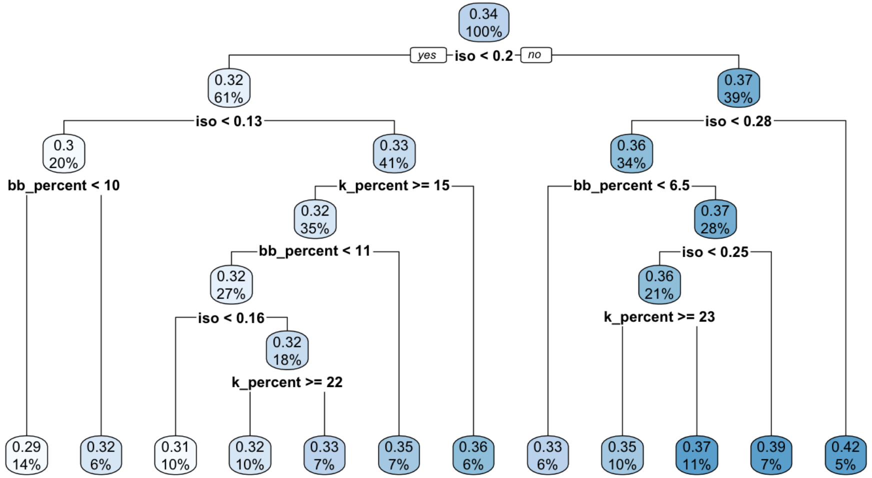
- `rpart()` runs 10-fold CV to tune α for pruning
- Selects # terminal nodes via 1 SE rule

```r
plotcp(init_mlb_tree)
```

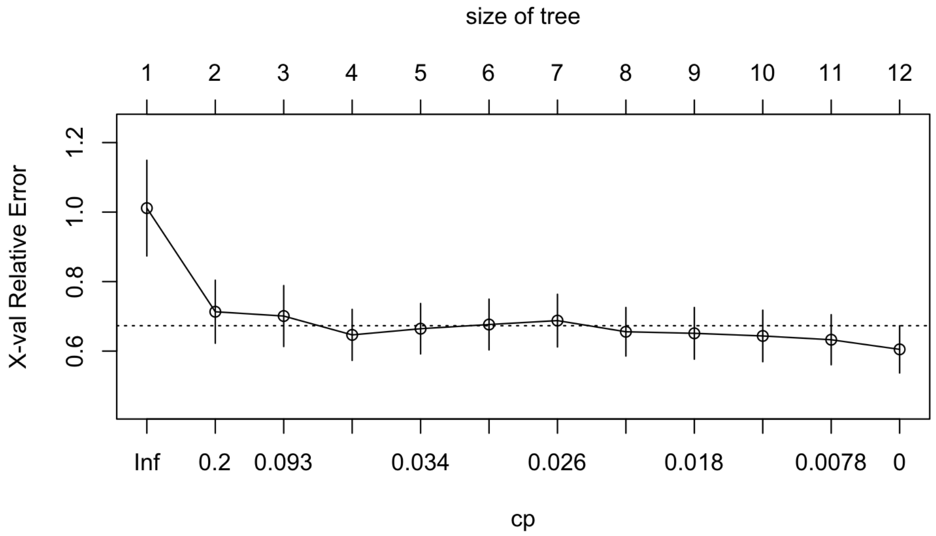


What about the full tree? (check out `rpart.control`)

```
full_mlb_tree <- rpart(formula = w_oba ~ bb_p
                       data = mlb_data, metho
                       control = list(cp = 0,
rpart.plot(full_mlb_tree)
```

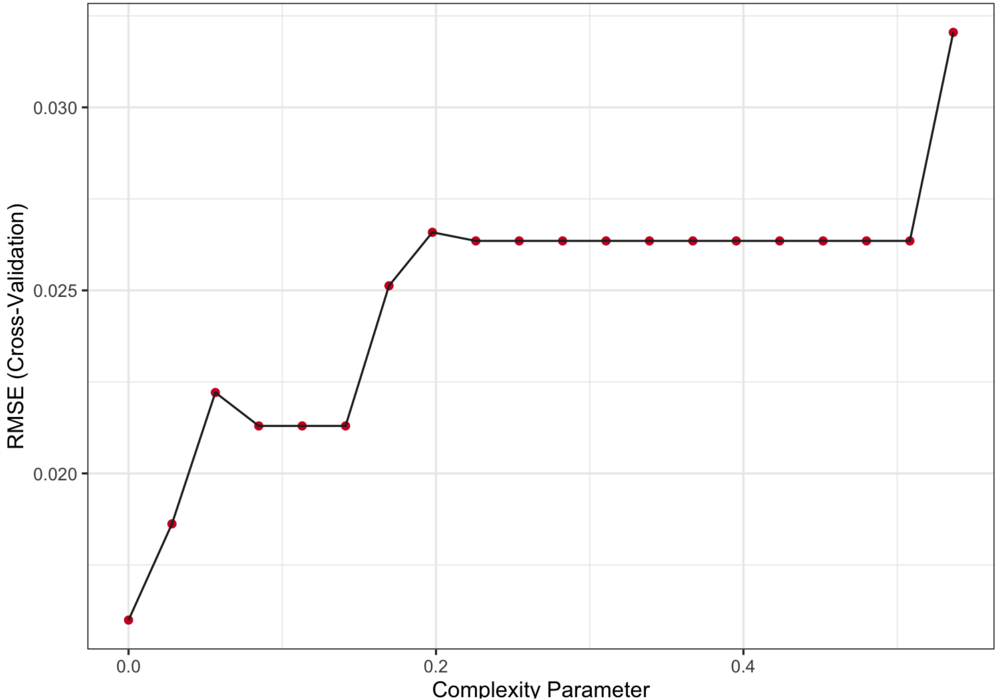



```
plotcp(full_mlb_tree)
```



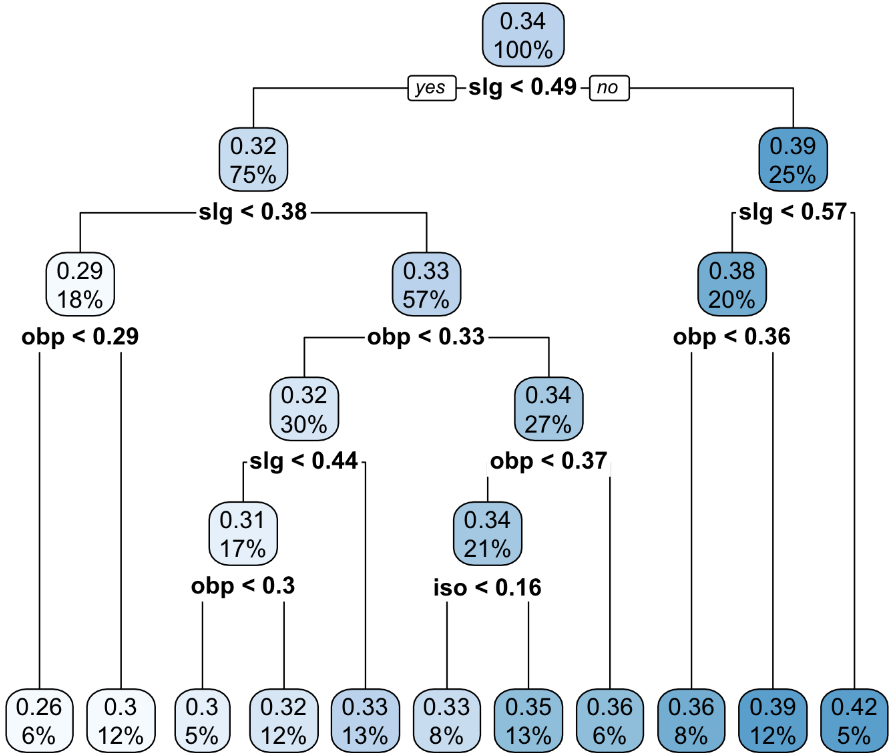
Train with caret

```
library(caret)
caret_mlb_tree <- train(w_oba ~ bb_percent + k_percent + iso + avg + obp + slg + war,
                        data = mlb_data, method = "rpart",
                        trControl = trainControl(method = "cv", number = 10),
                        tuneLength = 20)
ggplot(caret_mlb_tree) + theme_bw()
```



Display the final model

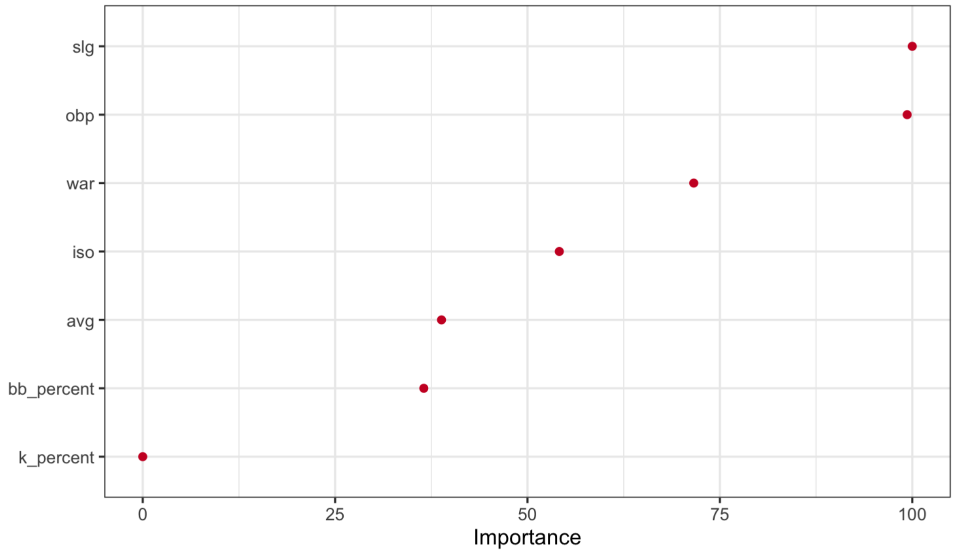
```
rpart.plot(caret_mlb_tree$finalModel)
```



Summarizing variables in tree-based models

Variable importance - based on reduction in SSE (*notice anything odd?*)

```
library(vip)
vip(caret_mlb_tree, geom = "point") + theme_b
```



- Summarize single variable's relationship with **partial dependence plot**

```
library(pdp)
partial(caret_mlb_tree, pred.var = "obp") %>%
```