Machine learning

Decision trees

July 6th, 2023

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.

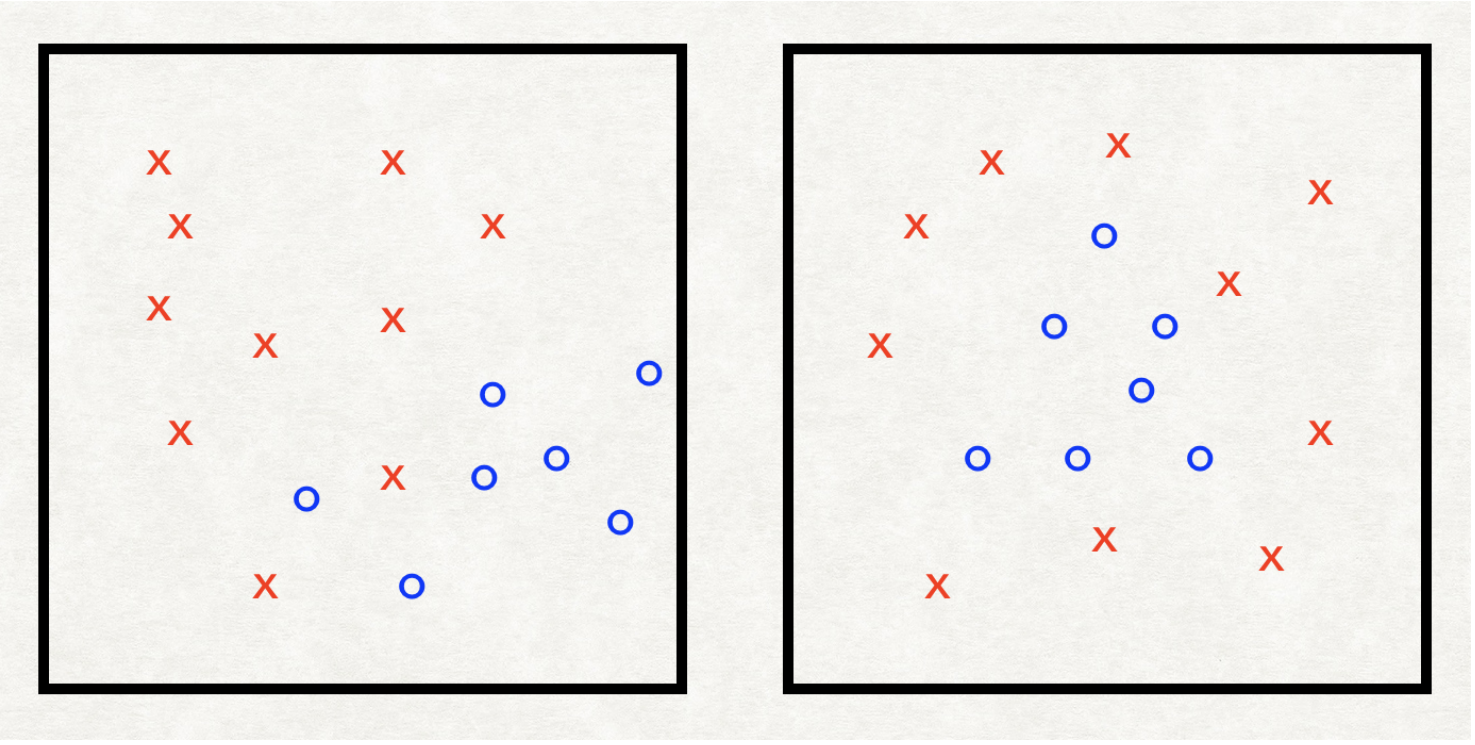
(And the answer is *not* deep learning. Because if the underlying relationship between your predictors and your response is truly linear, *you do not need to apply deep learning*! Just do linear regression. Really. It's OK.)

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

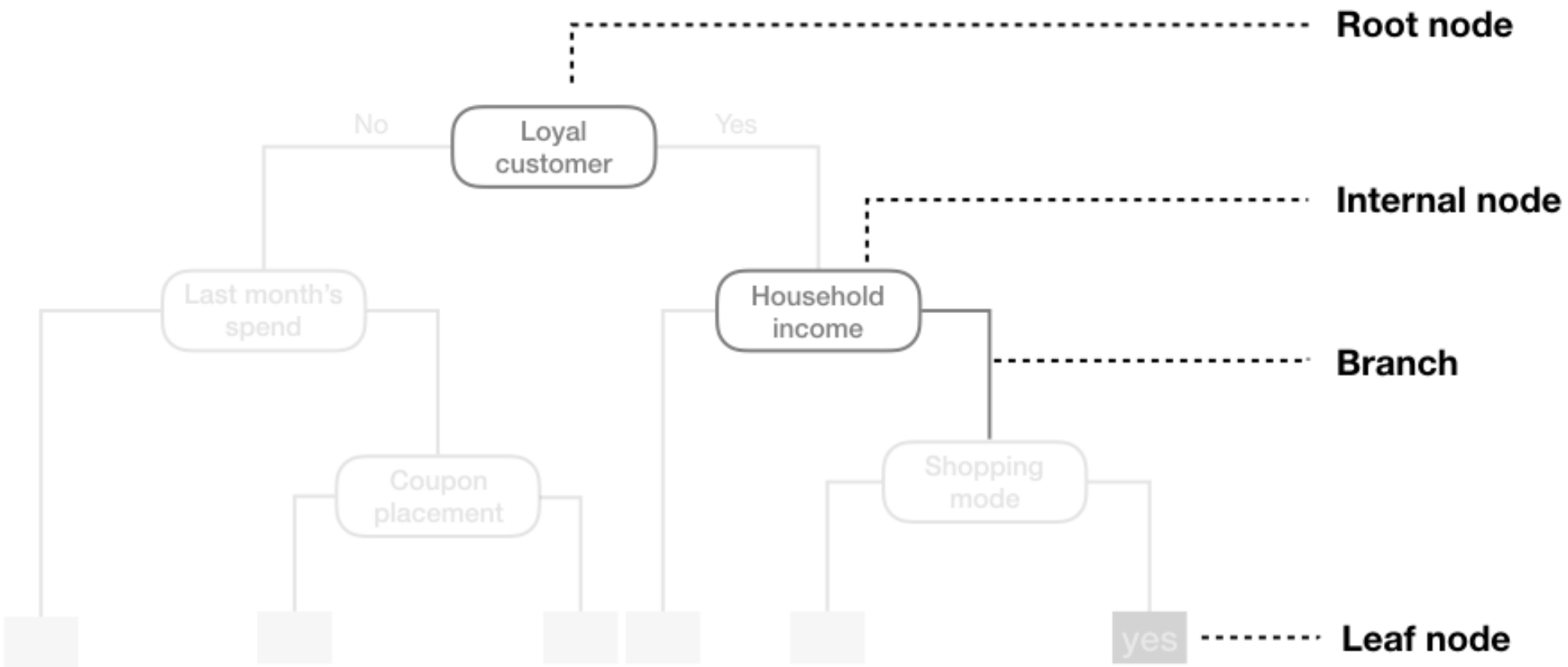
We stop splitting the tree once a **stopping criteria** has been reached (e.g. maximum depth allowed)

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

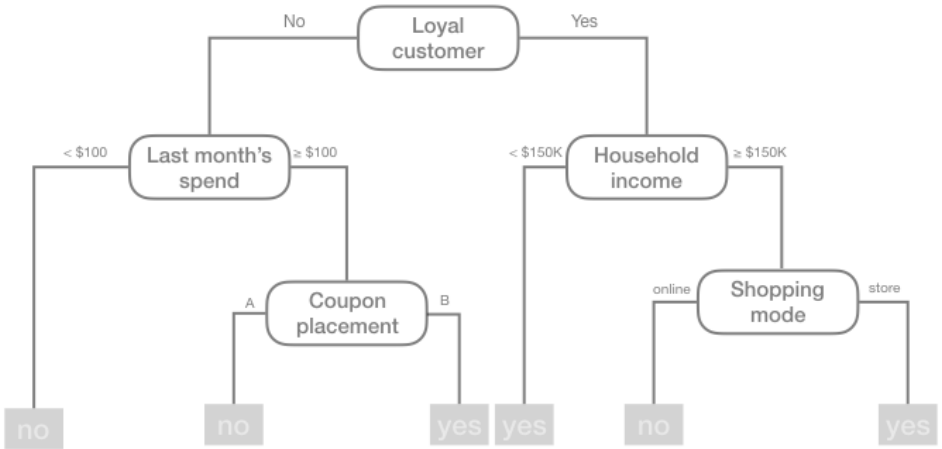
Most popular approach is Leo Breiman's **C**lassification **A**nd **R**egression **T**ree (CART) algorithm

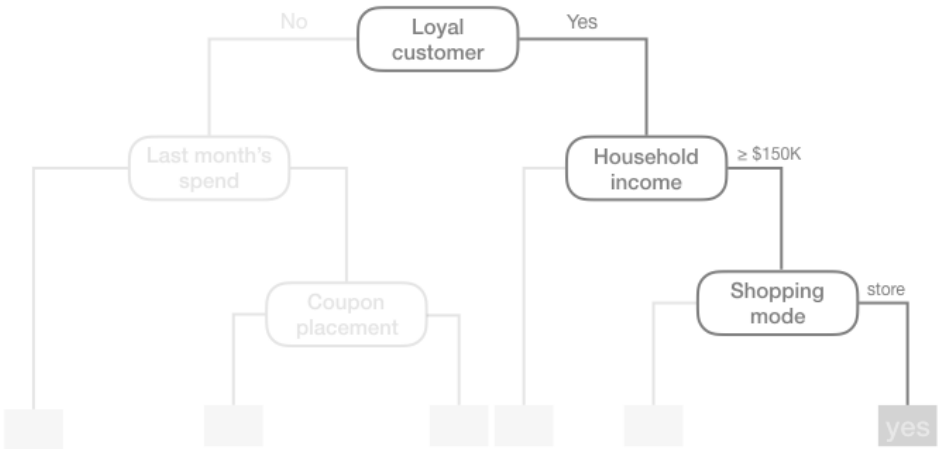
Decision tree structure



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):

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

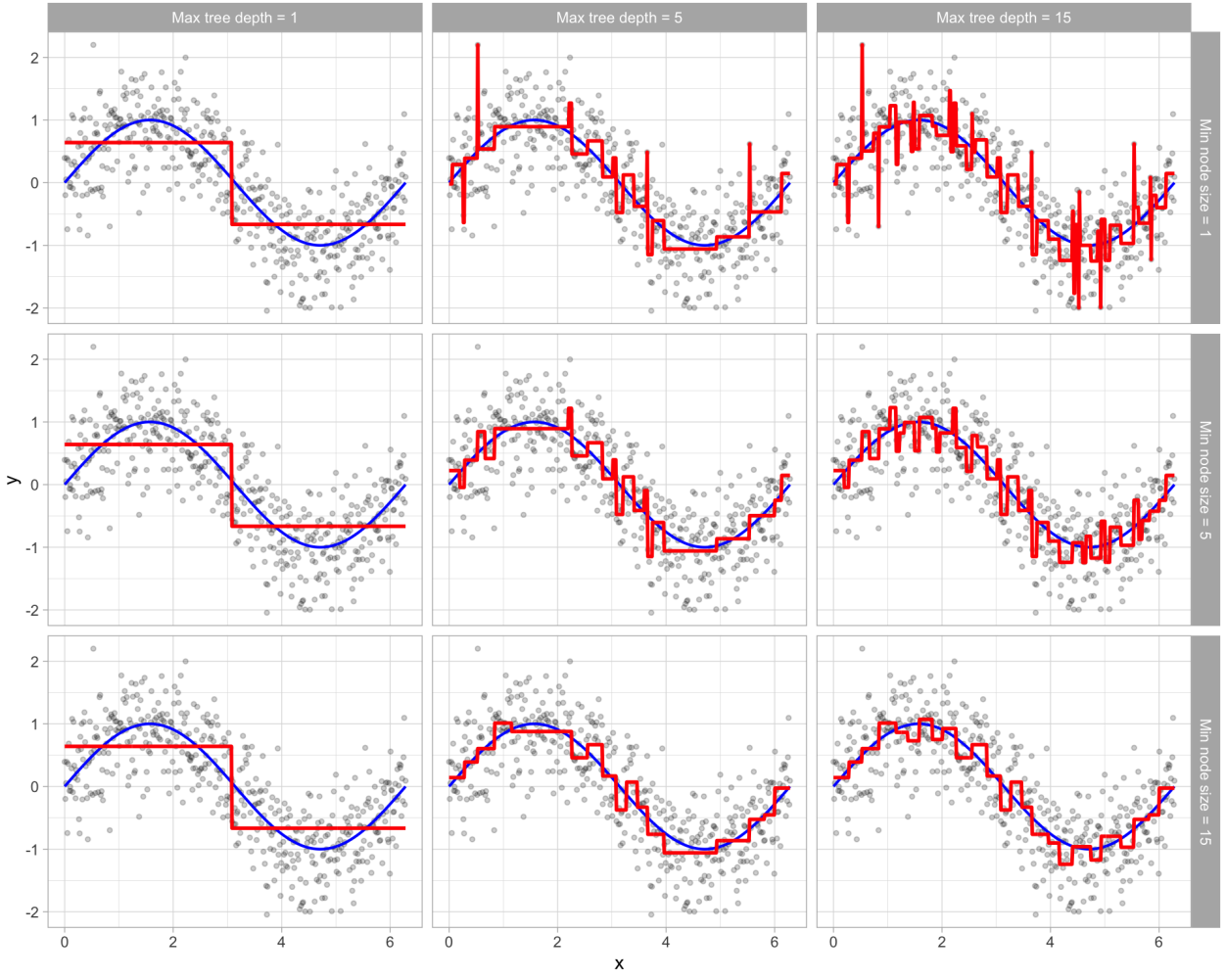
- For classification trees we minimize the node's *impurity* the **Gini index**
  - where p_k is the proportion of observations in the node belonging to class k out of K total classes
  - want to minimize $Gini$: small values indicate a node has primarily one class (*is more pure*)

$$Gini = 1 - \sum_{k}^{K} p_k^2$$

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?

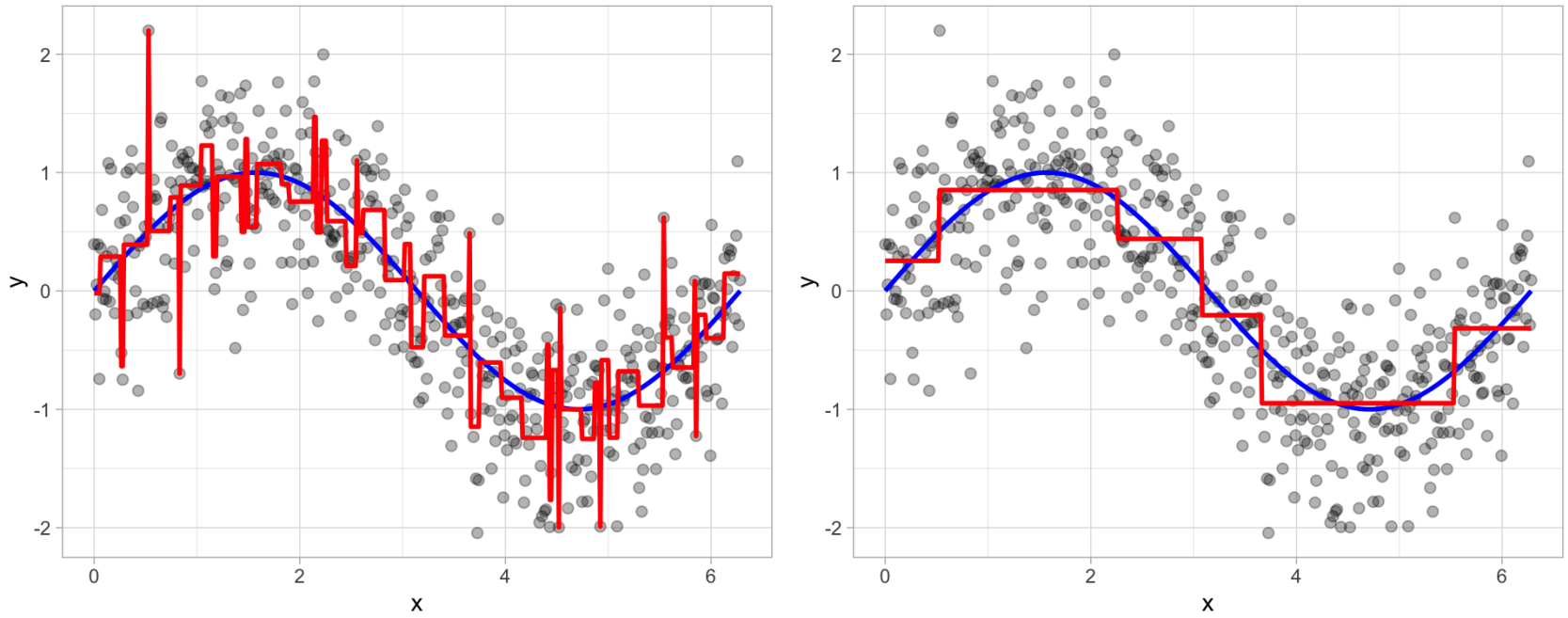
Tune the **maximum tree depth** or **minimum node size**



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 $SSE + \alpha \cdot (\#\text{ of terminal nodes})$



Example data: MLB 2022 batting statistics

Downloaded MLB 2022 batting statistics leaderboard from Fangraphs

```
library(tidyverse)
mlb_data <- read_csv("https://shorturl.at/iCP15") %>%
  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_percent k_percent   iso
##   <chr>     <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>      <dbl>     <dbl> <dbl>
## 1 Rafael D… BOS      85   374    22    62    55     2        6.7      17.6 0.28
## 2 Aaron Ju… NYY      88   385    33    72    69     8       11.4      26   0.337
## 3 Nolan Ar… STL      88   370    18    41    59     1        8.9      13   0.233
## 4 Manny Ma… SDP      82   349    15    56    51     7       10.6      18.6 0.213
## 5 Paul Gol… STL      90   391    20    64    70     5       12        21.2 0.26
## 6 Jose Ram… CLE      87   375    19    54    75    13       10.7       9.9 0.288
## # ℹ 12 more variables: babip <dbl>, 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>
```

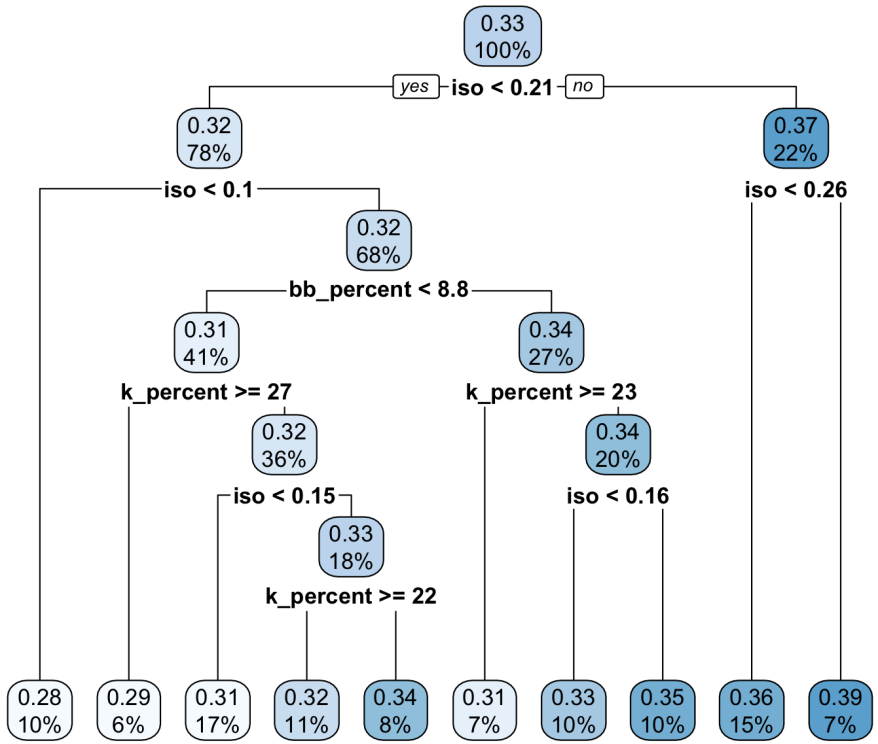
Regression tree example with the `rpart` package

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

```
## n= 157
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 157 0.215948200 0.3291338
##    2) iso< 0.2055 123 0.113126200 0.3175691
##      4) iso< 0.1035 16 0.016633000 0.2837500 *
##      5) iso>=0.1035 107 0.075457050 0.3226262
##       10) bb_percent< 8.75 65 0.039689380 0.3146154
##         20) k_percent>=27.15 9 0.001585556 0.2902222 *
##         21) k_percent< 27.15 56 0.031887930 0.3185357
##           42) iso< 0.152 27 0.010937850 0.3089259 *
##           43) iso>=0.152 29 0.016135240 0.3274828
##             86) k_percent>=21.85 17 0.008568235 0.3194706 *
##             87) k_percent< 21.85 12 0.004929667 0.3388333 *
##       11) bb_percent>=8.75 42 0.025140980 0.3350238
```

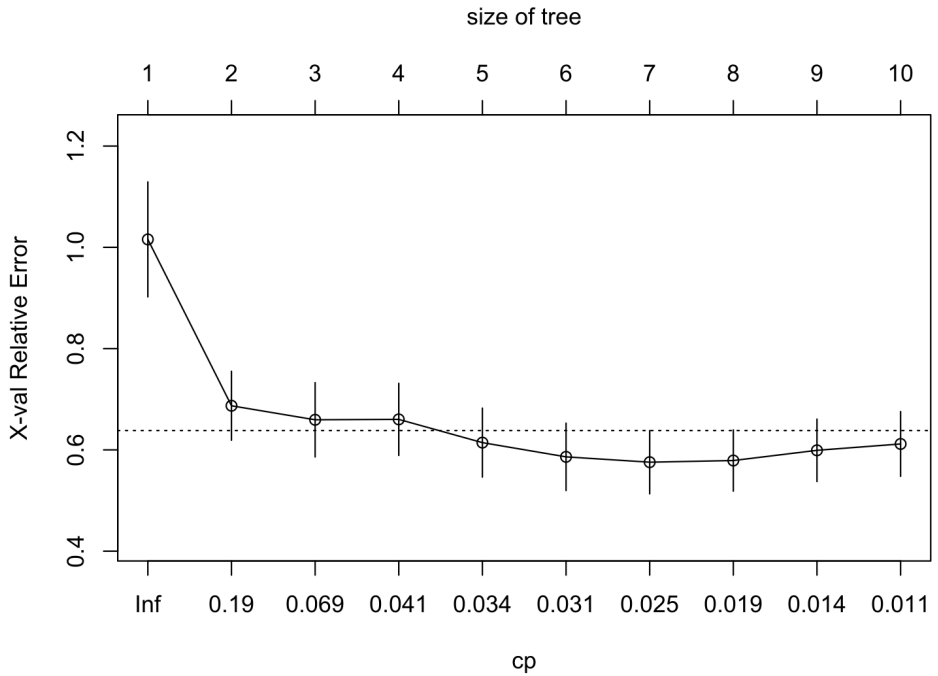
Display the tree with `rpart.plot`

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



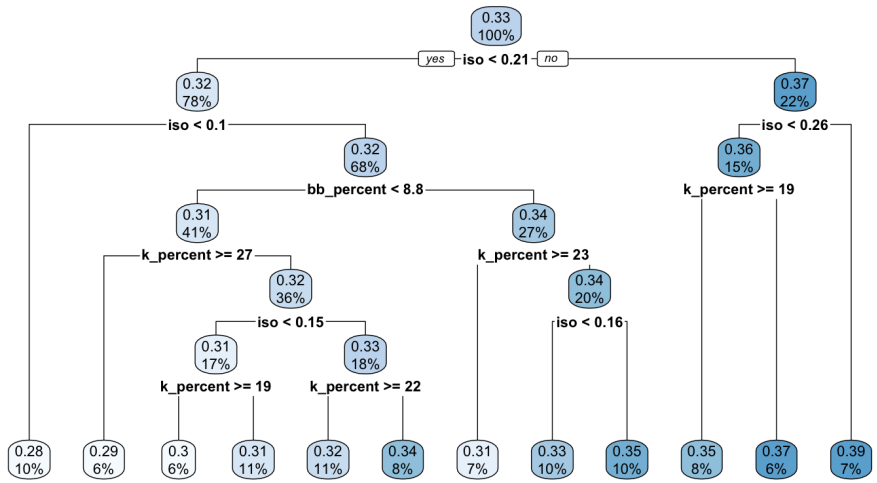
- `rpart()` runs 10-fold CV to tune α for pruning
- Selects # terminal nodes via 1 SE rule

```
plotcp(init_mlb_tree)
```

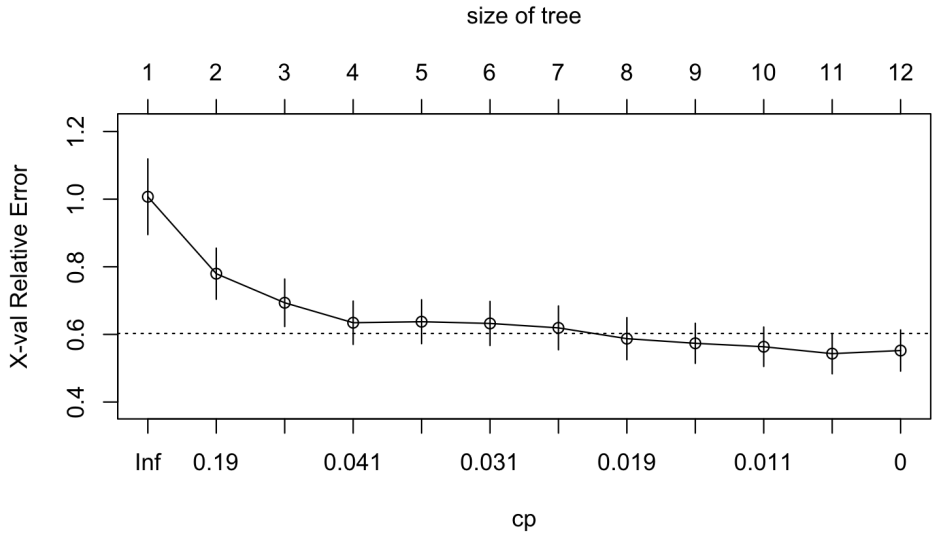


What about the full tree? (check out `rpart.control`)

```
full_mlb_tree <- rpart(formula = w_oba ~
            bb_percent + k_percent + iso,
            data = mlb_data, method = "anova"
            control = list(cp = 0, xval = 10)
rpart.plot(full_mlb_tree)
```

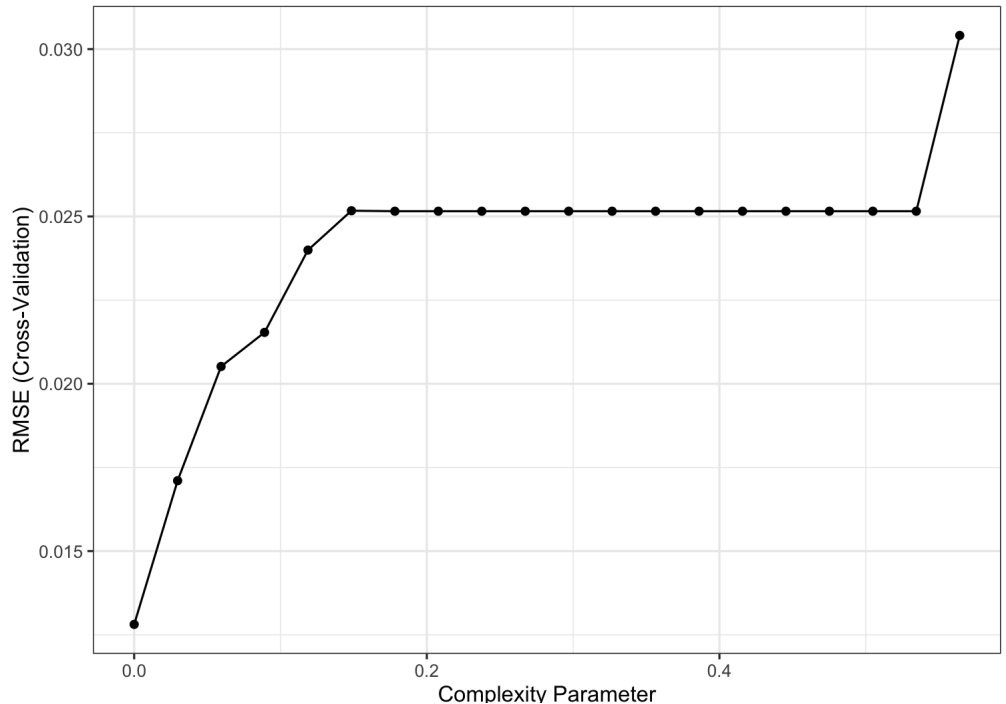



```
plotcp(full_mlb_tree)
```



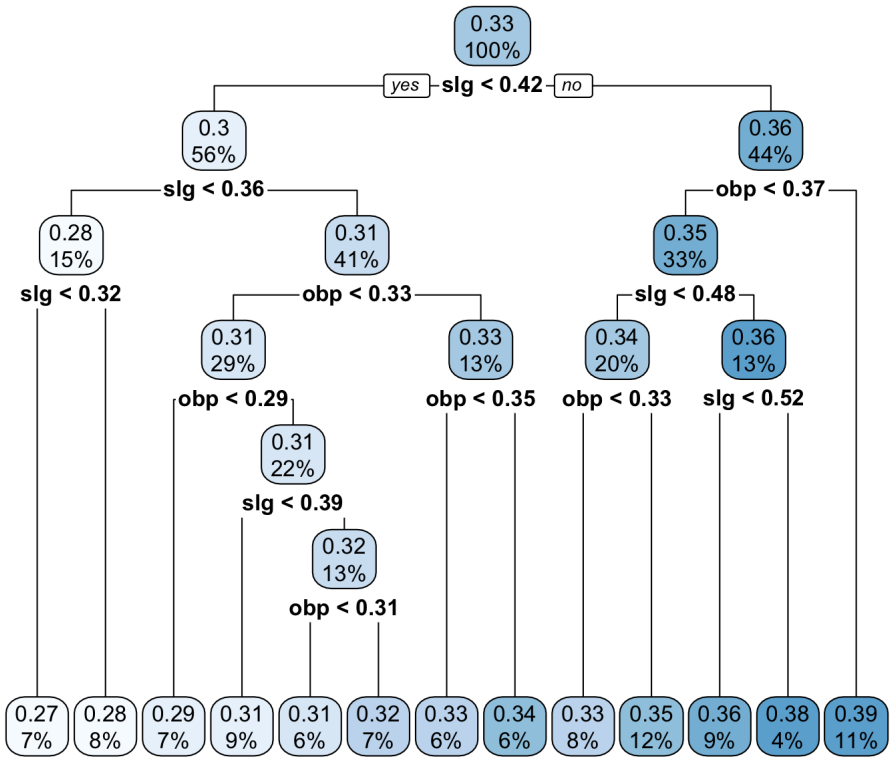
Train with `caret`

```
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

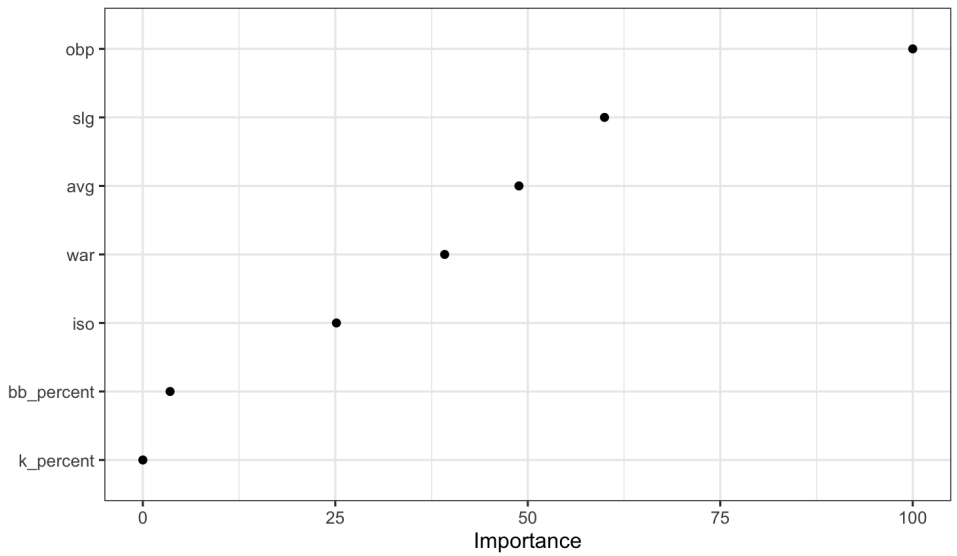
```
rpart.plot(caret_mlb_tree$finalModel)
```



Summarizing variables in tree-based models

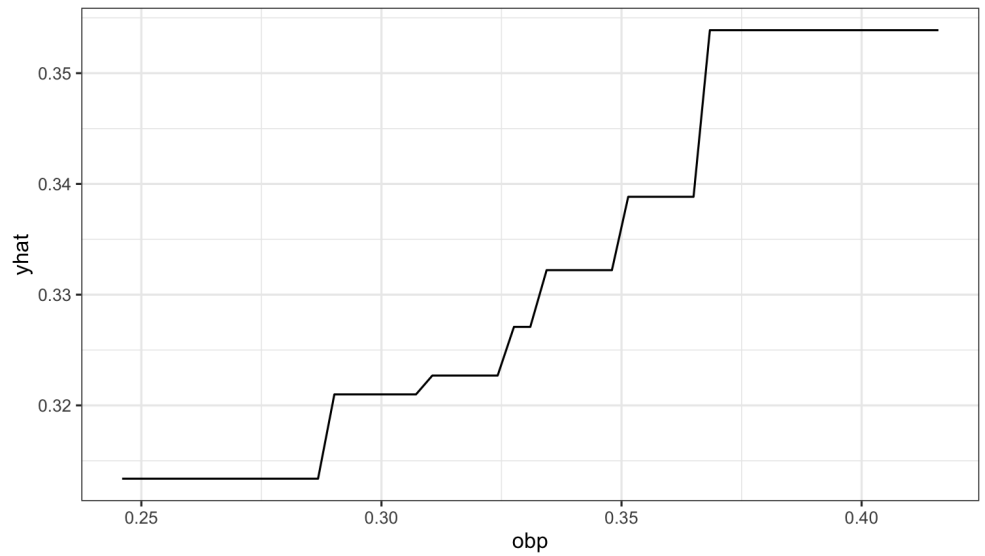
Variable importance - based on reduction in SSE (*notice anything odd?*)

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



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

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



Classification: predicting MLB HRs

Used the `baseballr` package to scrape all batted-balls from 2022 season:

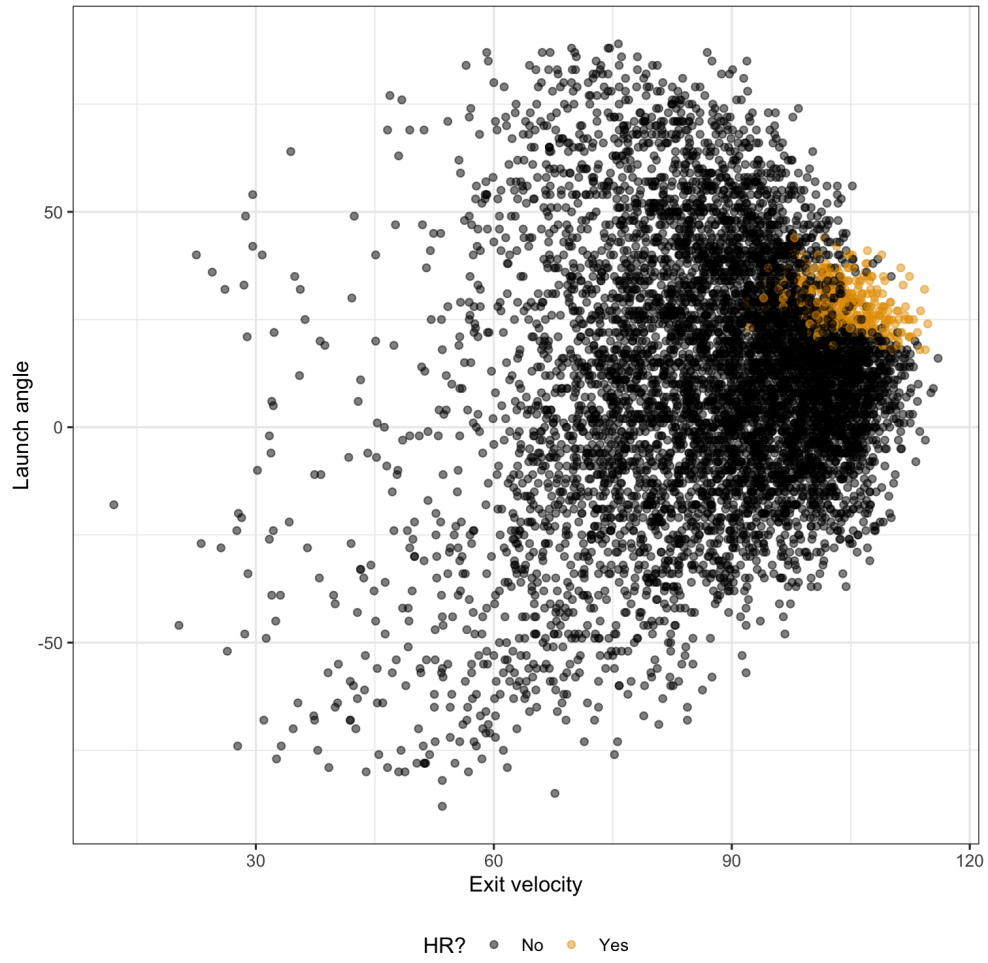
```
library(tidyverse)
batted_ball_data <- read_csv("https://shorturl.at/moty2") %>%
  mutate(is_hr = as.numeric(events == "home_run")) %>%
  filter(!is.na(launch_angle), !is.na(launch_speed),
         !is.na(is_hr))
table(batted_ball_data$is_hr)
```

```
##
##    0    1
## 6702  333
```

Predict HRs with launch angle and exit velocity?

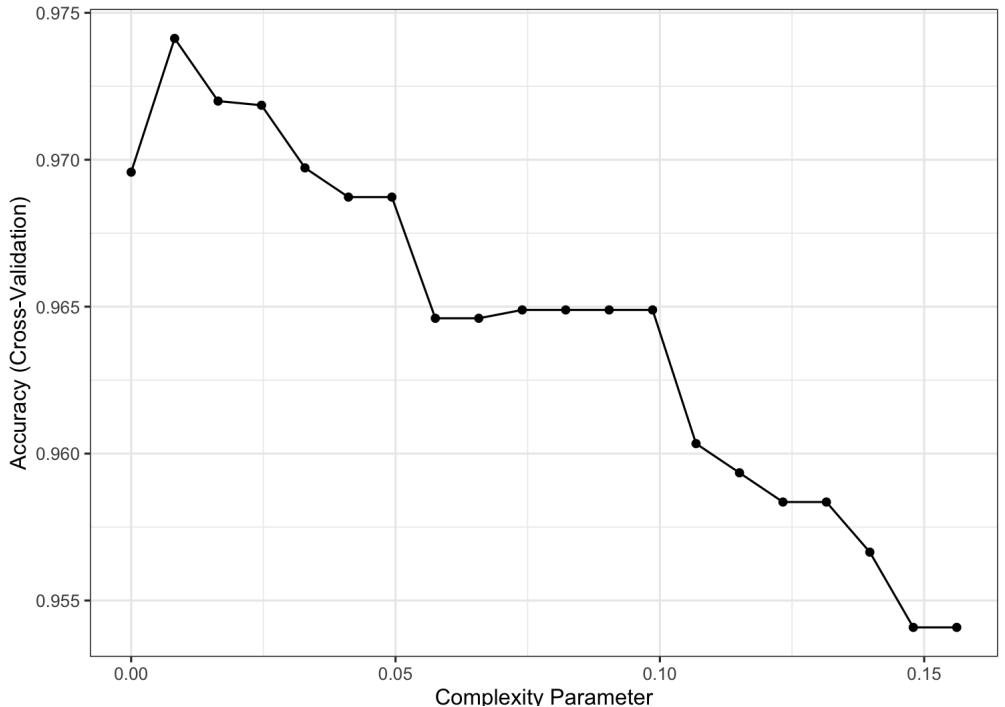
```
batted_ball_data %>%
  ggplot(aes(x = launch_speed,
             y = launch_angle,
             color = as.factor(is_hr))) +
  geom_point(alpha = 0.5) +
  ggthemes::scale_color_colorblind(
    labels = c("No", "Yes")) +
  labs(x = "Exit velocity",
       y = "Launch angle",
       color = "HR?") +
  theme_bw() +
  theme(legend.position = "bottom")
```

- HRs are relatively rare and confined to one area of this plot



Train with `caret`

```
library(caret)
caret_hr_tree <- train(as.factor(is_hr) ~ launch_speed + launch_angle,
                       data = batted_ball_data, method = "rpart",
                       trControl = trainControl(method = "cv", number = 10),
                       tuneLength = 20)
ggplot(caret_hr_tree) + theme_bw()
```



Display the final model

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