07 - Trees and Forests

ml4econ, HUJI 2021

Itamar Caspi
April 25, 2021 (updated: 2021-04-25)

Packages and setup

Use the {pacman} package that automatically loads and installs packages if necessary:

```
if (!require("pacman")) install.packages("pacman")

pacman::p_load(
  tidyverse,   # for data wrangling and visualization
  broom,       # for tidy model output
  rpart,       # for estimating CART
  rpart.plot,  # for plotting rpart objects
  ranger,      # for estimating random forests
  vip,         # for variable importance plots
  knitr,       # for displaying nice tables
  here         # for referencing folders and files
)

# remotes::install_github("grantmcdermott/parttree")
library(parttree)
```

Set a theme for `ggplot` (Relevant only for the presentation), and set a seed for replication

```
theme_set(theme_grey(20))
set.seed(1203)
```

Outline

- Stratification
- Regression Trees
- Classification Trees
- Random Forests
- Other Ensemble Methods

Types of desicion tree applications

- Decision trees can be applied to both regression and classification tasks.
- We first consider regression problems using the Boston dataset, and then move on to classification using the Titanic dataset.

Stratification

Boston housing (again)

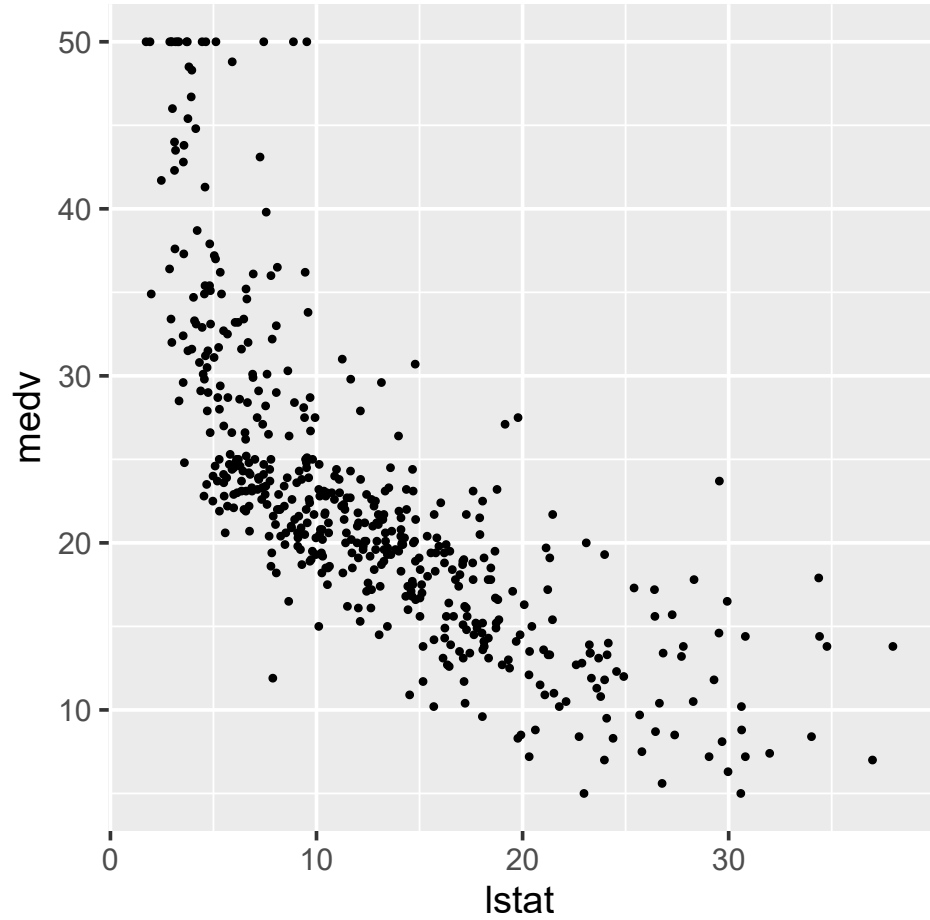
Load the data

```
boston <-
  here("04-ml-workflow/data","BostonHousing.csv") %>%
  read_csv()
```

```
## Parsed with column specification:
## cols(
##   crim = col_double(),
##   zn = col_double(),
##   indus = col_double(),
##   chas = col_double(),
##   nox = col_double(),
##   rm = col_double(),
##   age = col_double(),
##   dis = col_double(),
##   rad = col_double(),
##   tax = col_double(),
##   ptratio = col_double(),
##   b = col_double(),
##   lstat = col_double(),
##   medv = col_double()
## )
```

Recall the nonlinear association between `lstat` and `medv`

```
boston %>%
  ggplot(aes(lstat, medv)) +
  geom_point()
```



Two-way split of `lstat`

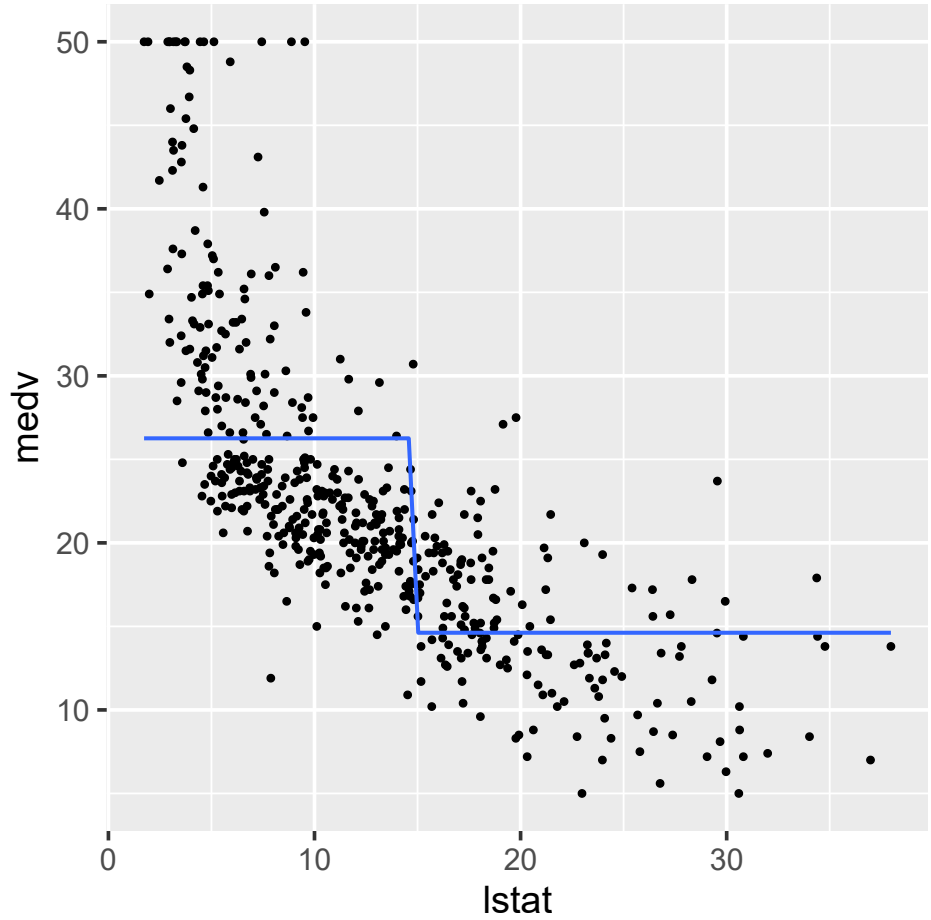
Let D_i denote an arbitrary two-way "split" dummy variable such that:

$$D_i = \begin{cases} 1 & \text{if } lstat_i > 15 \\ 0 & \text{otherwise}, \end{cases}$$

On the left, the blue step-function is the fitted value from running

$$medv_i = \beta_0 + \beta_1 D_i + \varepsilon_i$$

Note that the prediction is given by the average of $medv_i$ within each "region".



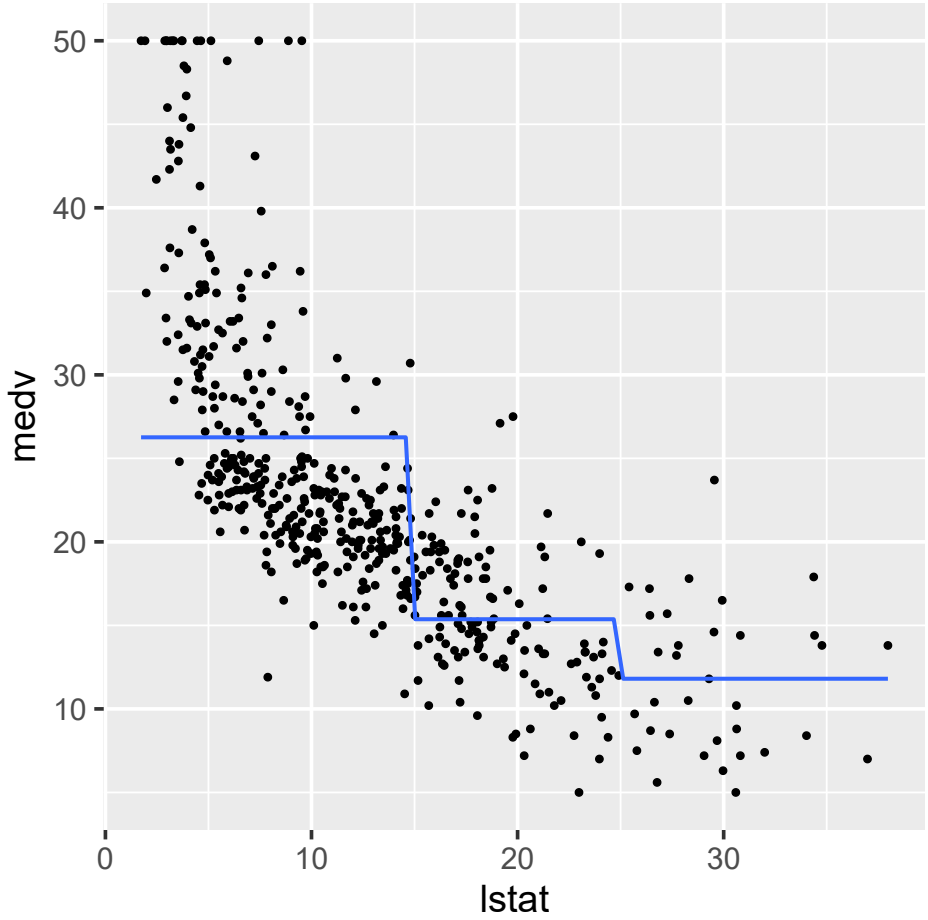
Three-way split

Now, let's try a three-way split:

$$D_{1i} = \begin{cases} 1 & \text{if } lstat_i > 25 \\ 0 & \text{otherwise,} \end{cases} \qquad D_{1i} = \begin{cases} 1 & \text{if } lstat_i < 5 \\ 0 & \text{otherwise ,} \end{cases}$$

```
boston %>%
  ggplot(aes(lstat, medv)) +
  geom_point() +
  geom_smooth(
    method = lm,
    se = FALSE,
    formula = y ~ (x>25) + (x<=25 & x>=15) + (x<15)
  )
```

Again, the prediction is given by the average of $medv_i$ within each "region".

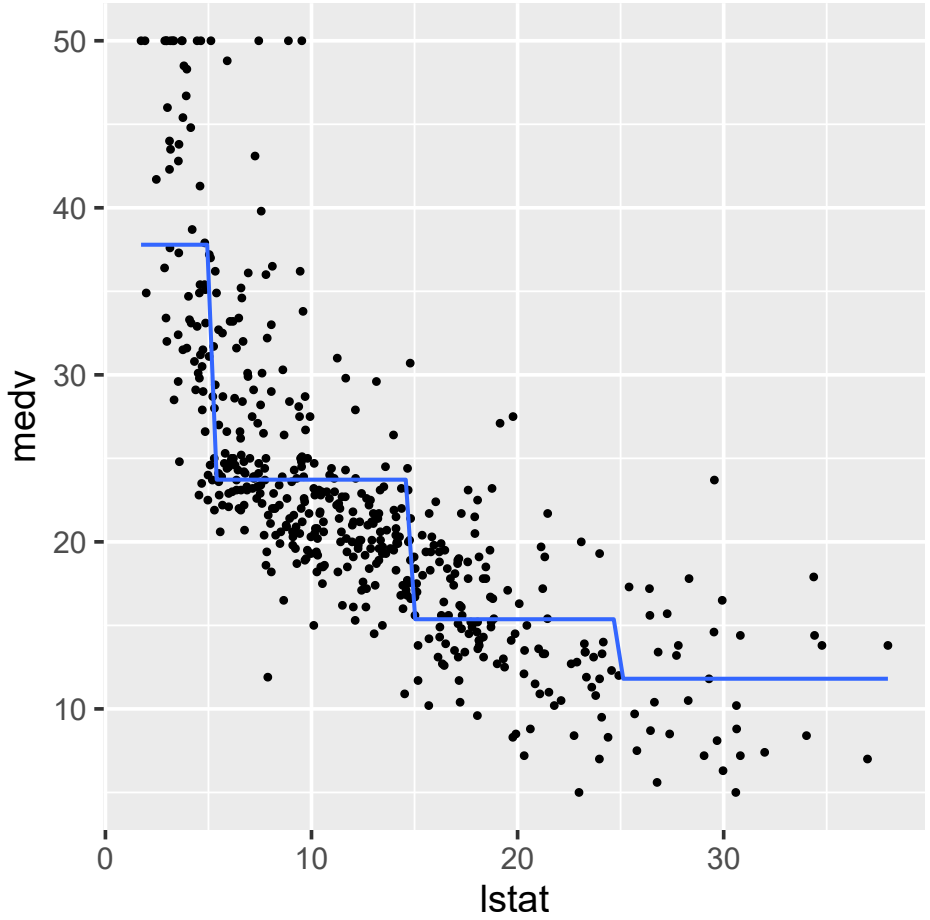


Four-way split

You get the point

```
boston %>%
  ggplot(aes(lstat, medv)) +
  geom_point() +
  geom_smooth(
    method = lm,
    se = FALSE,
    formula = y ~ (x>25) + (x<=25 & x>=15) + (x<15 &
  )
```

- The more splits we have, the better the fit. (What about prediction?)



Issues

In general using splits involve three main issues:

1. Where to split?
2. How many splits?
3. How to predict within each node?

The answers to the these questions are related to the decision trees framework.

Regression Trees

Classification and regression trees (CART)

Basic idea (Breiman et al., 1984):

1. Split the features space $x_1, x_2, \ldots, x_p$ - into M distinct and non-overlapping regions (rectangles), $R_1, R_2, \ldots, R_M$.

2. For every observation that falls into the region R_j, we make the same prediction (regression or classification). For example, for a continuous y,

$$\hat{y}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

   where y_m is a test observation that belongs to region R_m.

How to split?

- Going over every possible partitions of the feature space is infeasible. (Why?)
- Instead, the CART algorithm follows a **greedy** approach.
- Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j, s) = \{x | x_j \leq s\}, \qquad R_2(j, s) = \{x | x_j > s\}$$

- find the predictor j^* and split s^* that partitions the data into two regions $R_1(j^*, s^*)$ and $R_2(j^*, s^*)$ such that the overall sums of squares error are minimized:

$$\text{RSS} = \sum_{i \in R_1(j^*, s^*)} (y_i - \bar{y}_1)^2 + \sum_{i \in R_2(j^*, s^*)} (y_i - \bar{y}_2)^2$$

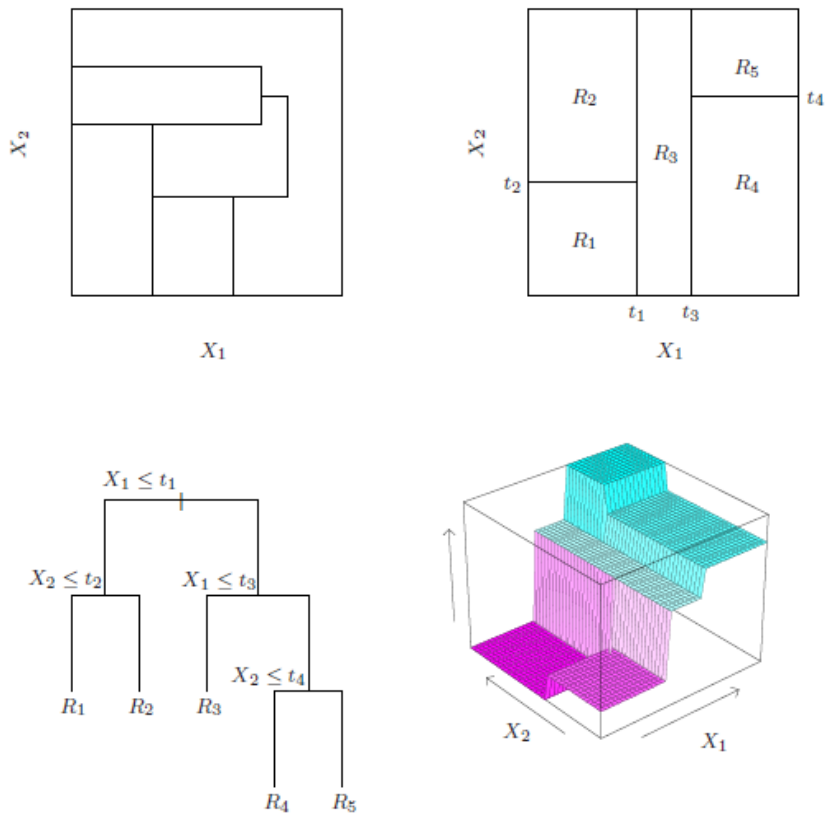
where $\bar{y}_1$ and $\bar{y}_2$ are the averages of the training set outcomes within each group.

The CART Algorithm

For each node, beginning with the root containing the full sample:

1. Determine the single RSS minimizing split for this node.
2. Split this parent node into the left and right node.
3. Apply steps 1 and 2 to each child node.
4. Continue until you reach a leaf node of some prespecified minimum size (e.g., stop splitting when there are fewer than, say, 10 observations in each leaf).

Feature space partitioning



Source: ESL, pp. 308.

- **Top right:** partition of a 2-D feature space by CART.
- **Top left:** general partition that cannot be obtained from CART.
- **Bottom left:** the tree corresponding to the partition in the top right.
- **Bottom right:** prediction surface.

How large should we grow the tree?

- Large tree - overfit. Small tree - high variance.
- The tree's level of *expressiveness* is captured by its size (the number of terminal nodes).
- Common practice: Build a large tree and **prune** the tree backwards using *cost-complexity pruning*.

Cost-complexity pruning

The cost complexity criterion associated with a tree T is given by

$$\text{RSS}_{cp}(T) = \text{RSS}(T) + cp|T|$$

where

- RSS is the sum of squared error for tree T.
- $|T|$ is the number of terminal nodes in tree T.
- cp is the complexity parameter.

Hence, for CART, the penalty is a function of the number of terminal nodes.

NOTE: cp and $|T|$ are analogous to λ and $\|\boldsymbol{\beta}\|_1$ in the lasso.

cp

The complexity parameter is unit free and ranges from 0 to 1:

- When $cp = 0$, we have a fully saturated tree.
- When $cp = 1$, there are no splits, i.e, we predict the unconditional mean.

Boston with $cp = 1$

The R implementation of the CART algorithm is called `{rpart}`. Estimating a tree is straightforward using the `rpart()` function:

```
tree_fit <- rpart(
  medv ~ lstat,
  data = boston,
  control = rpart.control(cp = 1)
)
```

Recall that setting $cp = 1$ enforces no splits.

Plotting a tree is done using the `{rpart.plot}` package

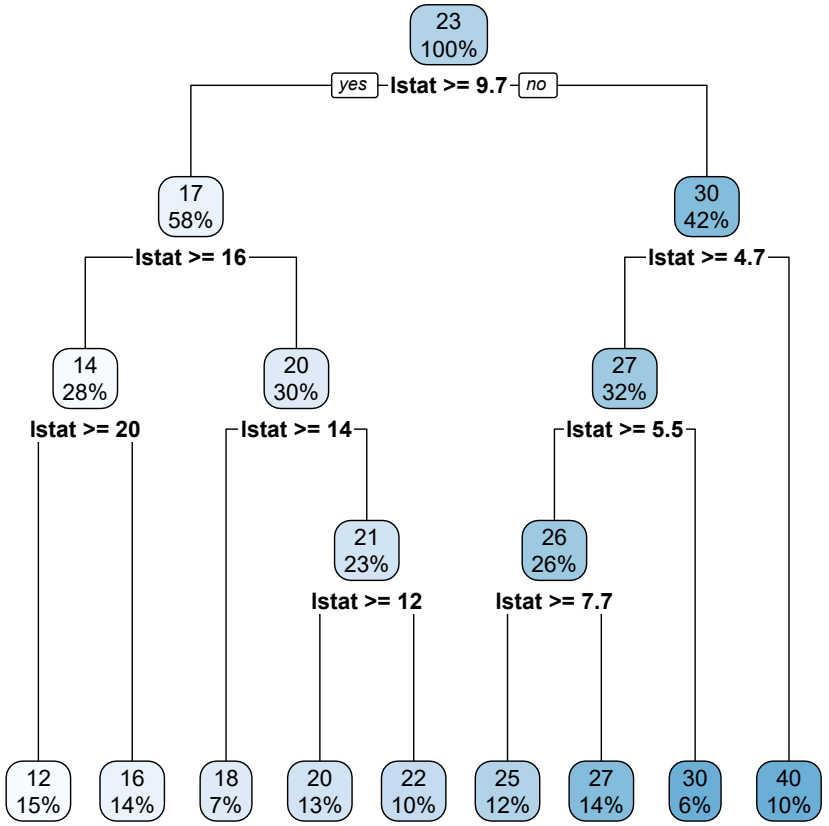
```
rpart.plot(tree_fit, cex = 2)
```

Boston with $cp = 0$

```
tree_fit <- rpart(
  medv ~ lstat,
  data = boston,
  control = rpart.control(cp = 0, minsplit = 80)
)
rpart.plot(tree_fit, cex = 1)
```

Setting $cp = 0$ results in a saturated tree.

Note we've set the minimum split criterion to 80 just to avoid clutter (too many splits) in the figure on the right.



Tuning cp

Breiman et al. (1984) suggest using a cross-validation approach to find the optimal* cp:

- For any value of the cp there is a unique, subtree T_{cp} that minimizes cost complexity $\text{RSS}_{cp}(T)$.
- To find the best subtree, we evaluate the data across a sequence of cp values. This process generates a (finite) sequence of subtrees which contains T_{cp}.
- Estimation of cp is achieved by cross-validation: we choose the value $\hat{cp}$ that minimizes the cross-validated RSS. Our final tree is $T_{\hat{cp}}$

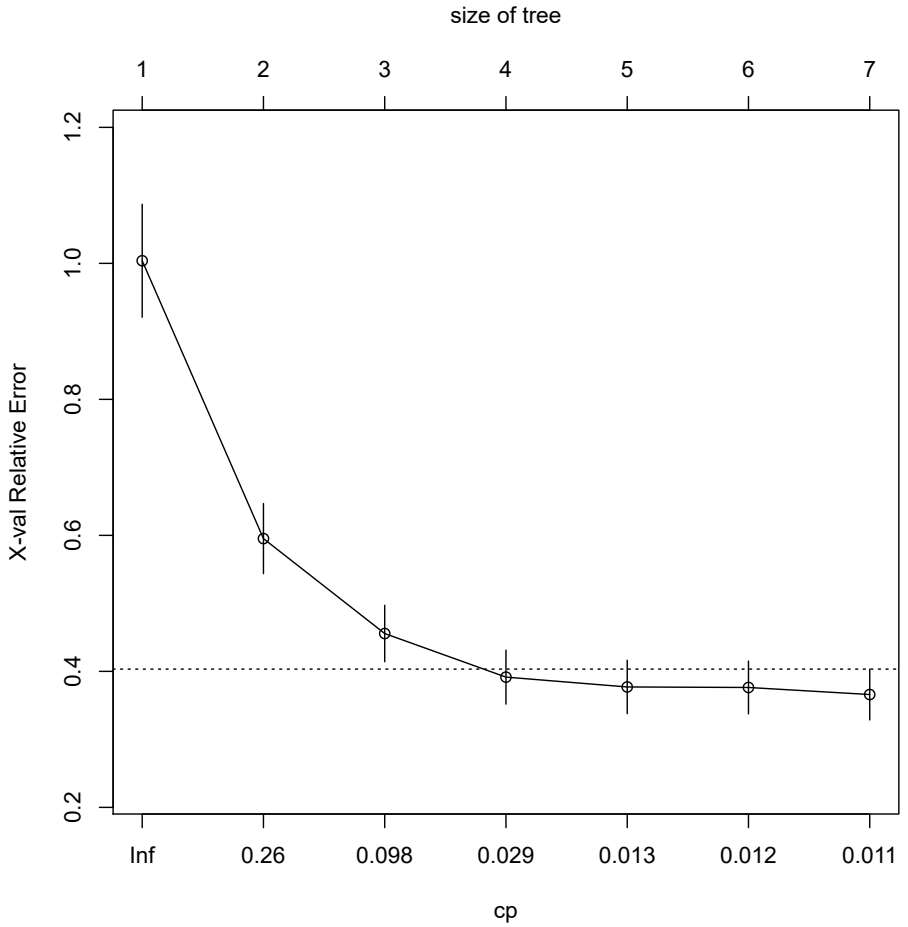
[*] Breiman et al. (1984) also propose using the 1se heuristic, i.e., find the smallest tree that is within one standard error of the tree with smallest absolute error.

Boston tree cross validation

The `plotcp()` function from the `{rpart}` package gives a visual representation of the cross-validation results in an `rpart` object:

```
tree_fit <- rpart(
  medv ~ lstat,
  data = boston
)
plotcp(tree_fit)
```

$cp = 0.029$ is the 1se optimal cp.



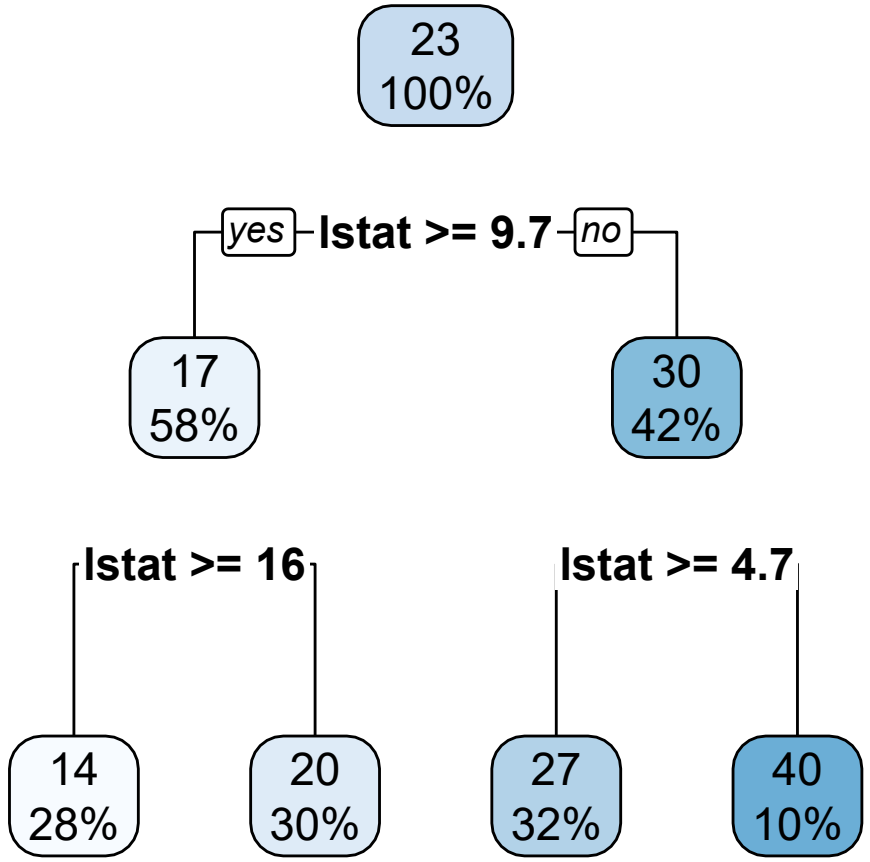
Boston pruned tree

We now proceed to pruning the tree using the `prune()` function (also from `{rpart}`), where we set `cp = 0.029`:

```
tree_prune <- prune(tree_fit, cp = 0.029)
```

And now we can plot the pruned tree:

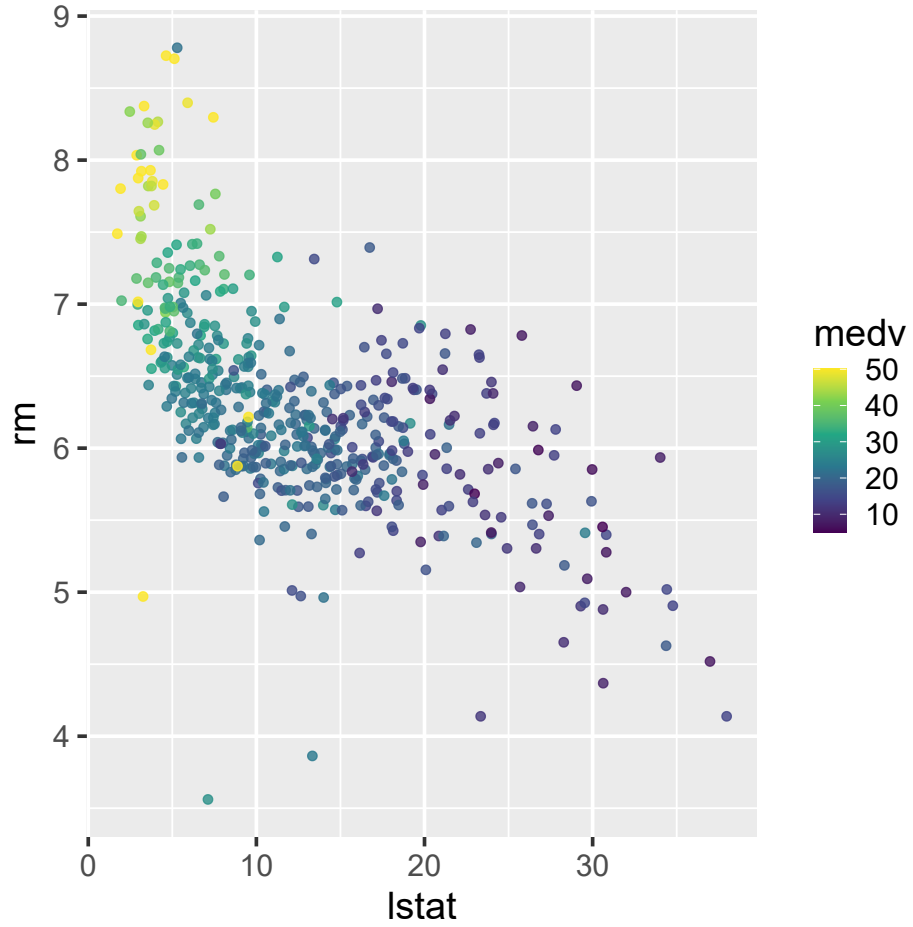
```
rpart.plot(tree_prune, cex = 2)
```



Trees with multiple features

How would you partition the data?

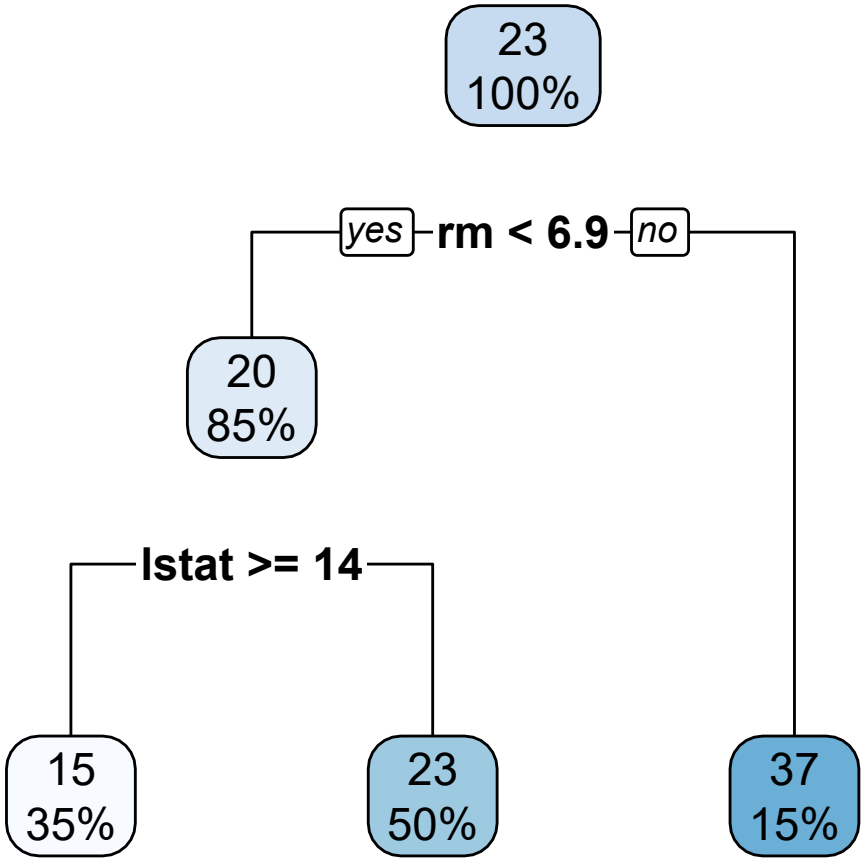
```
boston %>%
  ggplot(aes(lstat, rm, color = medv)) +
  scale_color_viridis_c() +
  geom_point(size = 2, alpha = 0.8)
```



`rpart` with many features

```
tree_fit <- rpart(
  medv ~ lstat + rm,
  data = boston,
  control = rpart.control(cp = 0.15),
  method = "anova"
)
rpart.plot(tree_fit, cex = 2)
```

For now, we will ignore the `cp = 0.15` argument.



Variable importance

- Once a tree has been estimated, it is common practice to assess the relative importance of the features to the prediction.
- A popular measure of variable (feature) importance is the total amount that the RSS is decreased due to splits over a given variable (Breiman et al., 1984.)
- Variables that appear higher or multiple times are more important than variable that appear lower in the tree or less frequently.

Boston variable importance

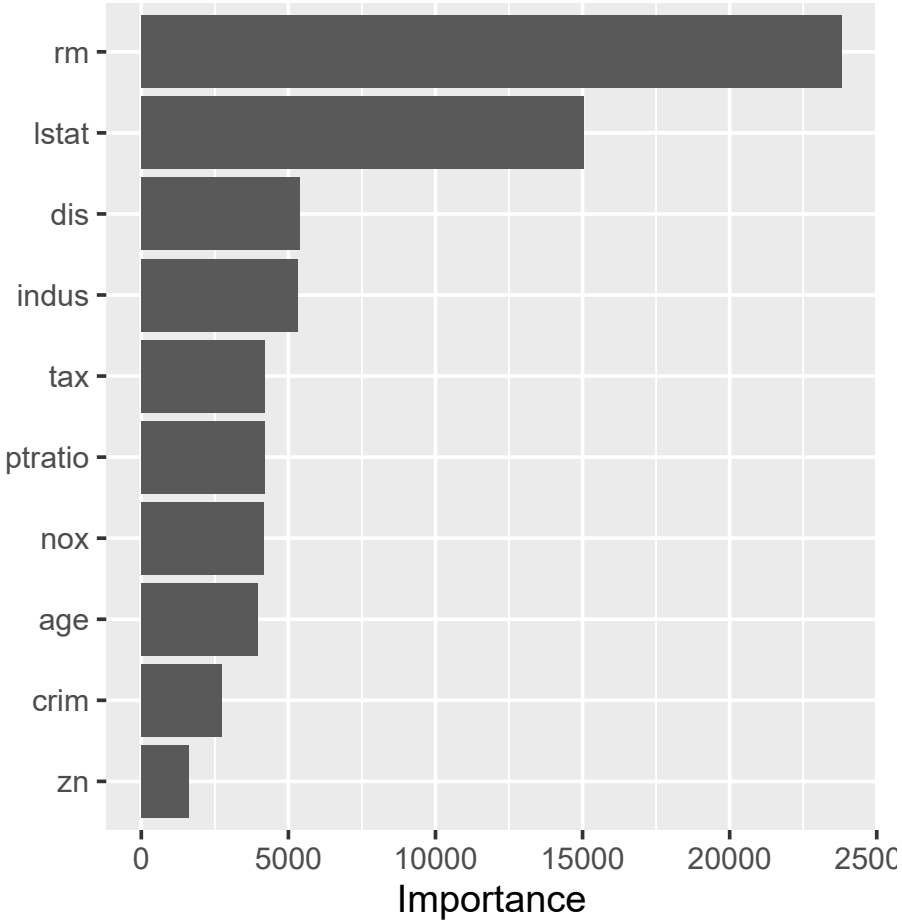
We now fit a tree with the entire set of features in the Boston dataset:

```
tree_all_vars <- rpart(medv ~ ., data = boston)
```

We can easily show variable importance for the fitted tree with the help of the {vip} package:

```
vip(tree_all_vars)
```

`rm` (the number of rooms) is clearly the most important feature in predicting `medv`.



Classification Trees

Adjustment to classification tasks: Splits

- Instead of RSS, splits are typically based on the *Gini index* (a.k.a *node purity*), defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk} \left(1 - \hat{p}_{mk}\right),$$

  a measure of total variance across the total classes (this is `rpart`'s default.)

- An alternative to the Gini index is *cross-entropy*, given by

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

Adjustment to classification: Prediction

- Instead of predicting based on the average y in region R_m, prediction is based on a majority rule: each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.

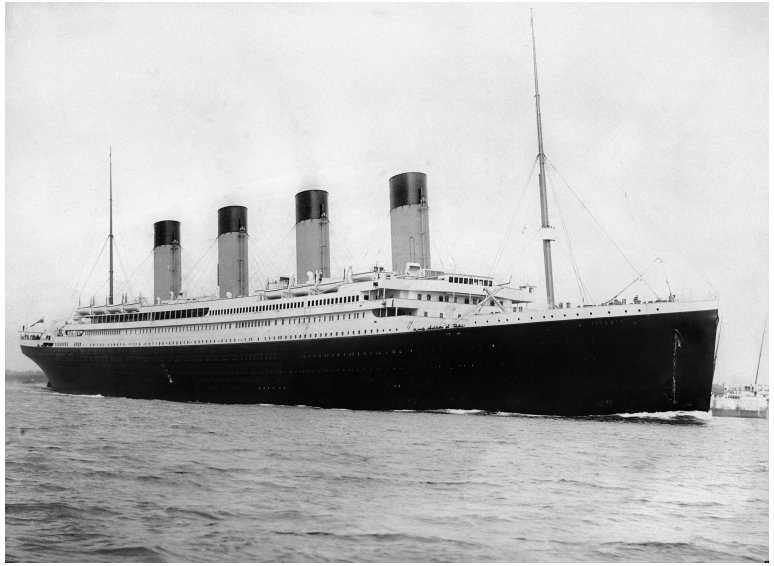
Adjustment to classification: Variable importance

- Variable importance is determined based on the amount that the Gini index/cross-entropy is decreased by splits over a given variable.

Classification trees example: The Titanic

"The RMS Titanic was a British passenger liner that sank in the North Atlantic Ocean in the early morning hours of 15 April 1912, after it collided with an iceberg during its maiden voyage from Southampton to New York City. There were an estimated 2,224 passengers and crew aboard the ship, and more than 1,500 died, making it one of the deadliest commercial peacetime maritime disasters in modern history."

— Wikipedia



Load the data

We'll replicate the results in Varian (2014) "Big data: New tricks for econometrics":

```
titanic_raw <-
  here("07-trees-forests/data", "titanic_varian.csv") %>%
  read_csv()
```

```
titanic_raw %>% glimpse()
```

```
## Rows: 1,309
## Columns: 14
## $ pclass    <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ survived  <dbl> 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1~
## $ name      <chr> "Allen, Miss. Elisabeth Walton", "Allison, Master. Hudson Trevor~
## $ sex       <chr> "female", "male", "female", "male", "female", "male", "female", ~
## $ age       <dbl> 29.00, 0.92, 2.00, 30.00, 25.00, 48.00, 63.00, 39.00, 53.00, 71.~
## $ sibsp     <dbl> 0, 1, 1, 1, 1, 0, 1, 0, 2, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1~
## $ parch     <dbl> 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1~
## $ ticket    <chr> "24160", "113781", "113781", "113781", "113781", "19952", "13502~
## $ fare      <dbl> 211.3375, 151.5500, 151.5500, 151.5500, 151.5500, 26.5500, 77.95~
## $ cabin     <chr> "B5", "C22 C26", "C22 C26", "C22 C26", "C22 C26", "E12", "D7", "~
## $ embarked  <chr> "S", "S", "S", "S", "S", "S", "S", "S", "S", "C", "C", "C", "C",~
## $ boat      <chr> "2", "11", NA, NA, NA, "3", "10", NA, "D", NA, NA, "4", "9", "6"~
## $ body      <dbl> NA, NA, NA, 135, NA, NA, NA, NA, NA, 22, 124, NA, NA, NA, NA, NA~
## $ home.dest <chr> "St Louis, MO", "Montreal, PQ / Chesterville, ON", "Montreal, PQ~
```

Data details

In this lecture, we will focus on a single outcome and two features:

Variable	Role	Definition	Values
`survived`	Outcome	Survival	0 = No, 1 = Yes
`age`	Feature	Age in years	
`pclass`	Feature	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd

Our goal: Predict which passengers survived based on their age and status.

Preprocessing

For what will follow, it would be useful to remove `NA`s and define `survived` as a factor:

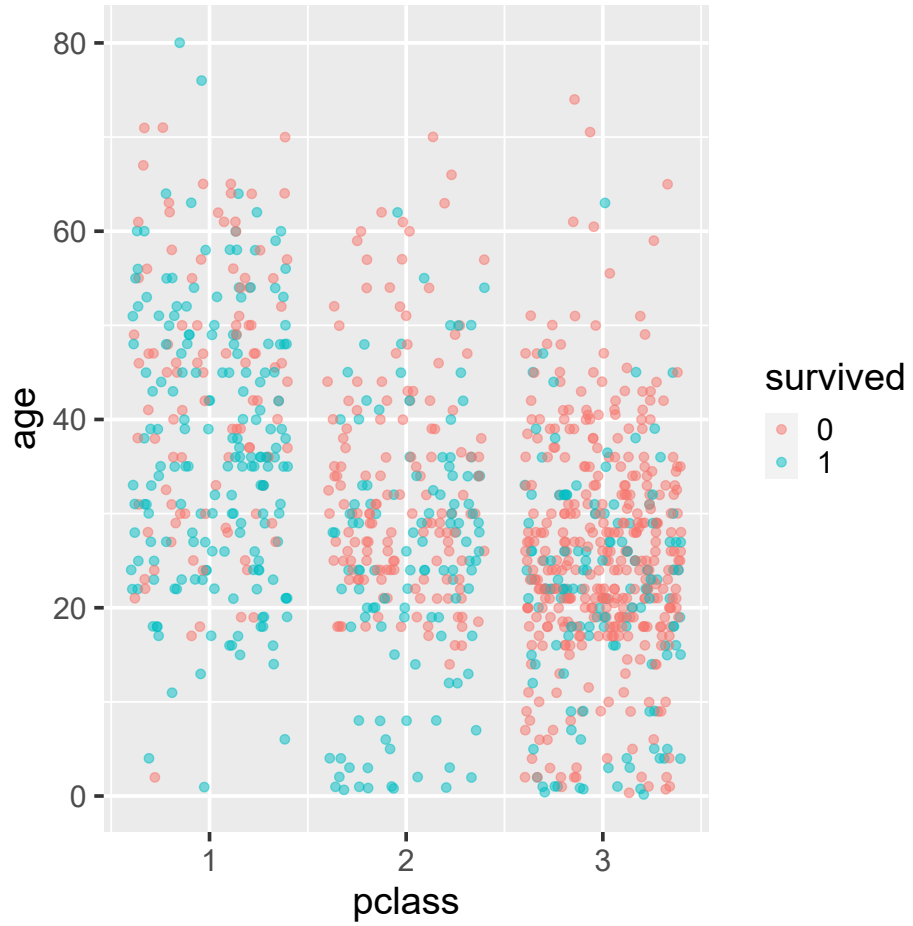
```
titanic <-titanic_raw %>%
  select(survived, age, pclass) %>%
  drop_na() %>%
  mutate(
    survived = as_factor(survived),
  )

titanic
```

```
## # A tibble: 1,046 x 3
##    survived   age pclass
##    <fct>    <dbl>  <dbl>
##  1 1        29         1
##  2 1         0.92      1
##  3 0         2         1
##  4 0        30         1
##  5 0        25         1
##  6 1        48         1
##  7 1        63         1
##  8 0        39         1
##  9 1        53         1
## 10 0        71         1
## # ... with 1,036 more rows
```

Partition

How would you stratify the data?

```
titanic %>%
  ggplot(aes(pclass, age, color = survived)) +
  geom_jitter(alpha = 0.5, size = 2)
```



Estimate , prune, and plot the tree

Fit the tree

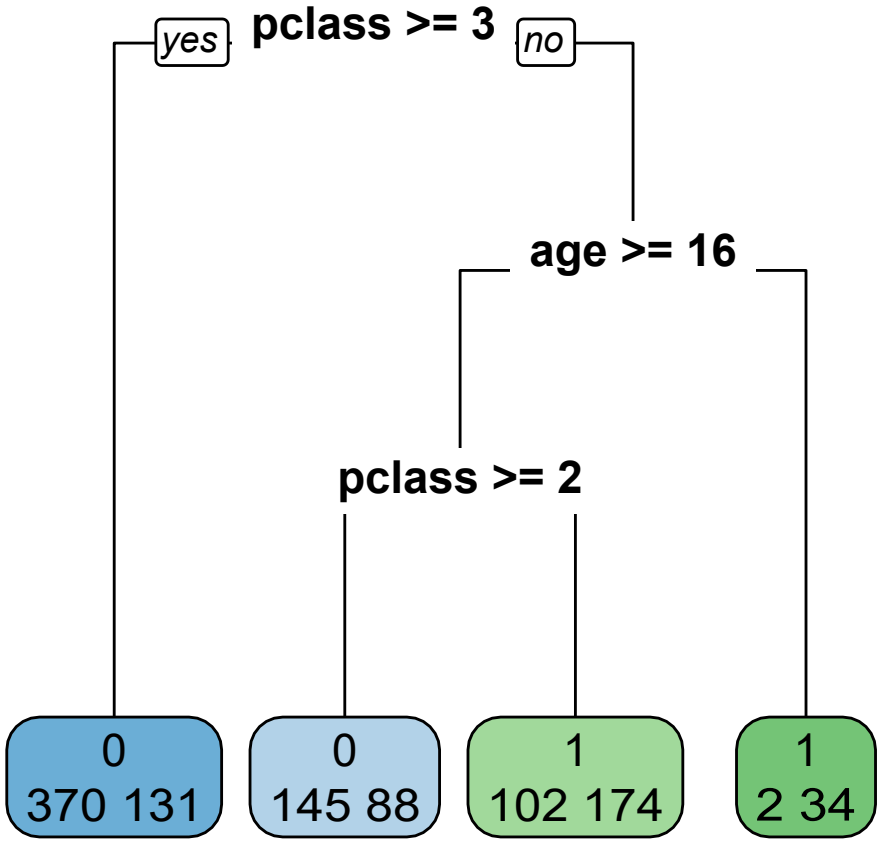
```
rpart_model <- rpart(
  survived ~ pclass + age,
  data = titanic,
  method = "class"
)
```

Prune

```
rpart_prune <- prune(rpart_model, cp = .038)
```

Plot

```
rpart.plot(rpart_prune, type=0, extra=1, cex = 2)
```

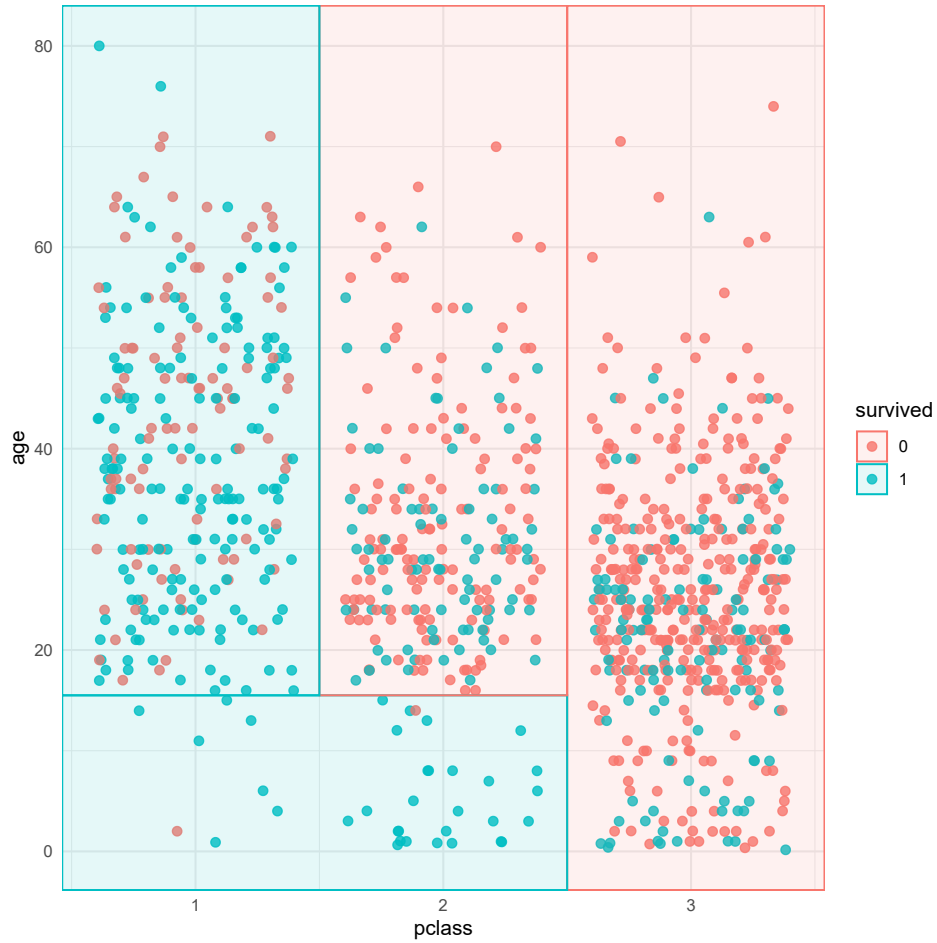


Partitioning a 2-D feature space

This is how the tree partitions the data

```
titanic %>%
  ggplot(aes(pclass, age, color = survived)) +
  geom_jitter(alpha = 0.8, size = 2) +
  geom_parttree(data = rpart_prune, aes(fill=survive
  theme_minimal()
```

where I've used the `geom_parttree()` function from the {parttree} package (in development.)



Recall: trees stratify the features space

Let's generate partition dummies that correspond to our tree's partitioning:

```
titanic_lm <-
  titanic %>%
  mutate(
    survived = as.numeric(survived) - 1,
    class_3 = if_else(pclass == 3, 1, 0),
    class_1_or_2_age_below_16 = if_else(pclass %in% c(1,2) & age < 16, 1, 0),
    class_1_age_above_16 = if_else(pclass == 1 & age >=16, 1, 0),
    class_2_age_above_16 = if_else(pclass == 2 & age >=16, 1, 0),
  ) %>%
  select(survived, starts_with("class_"))

titanic_lm %>% glimpse()
```

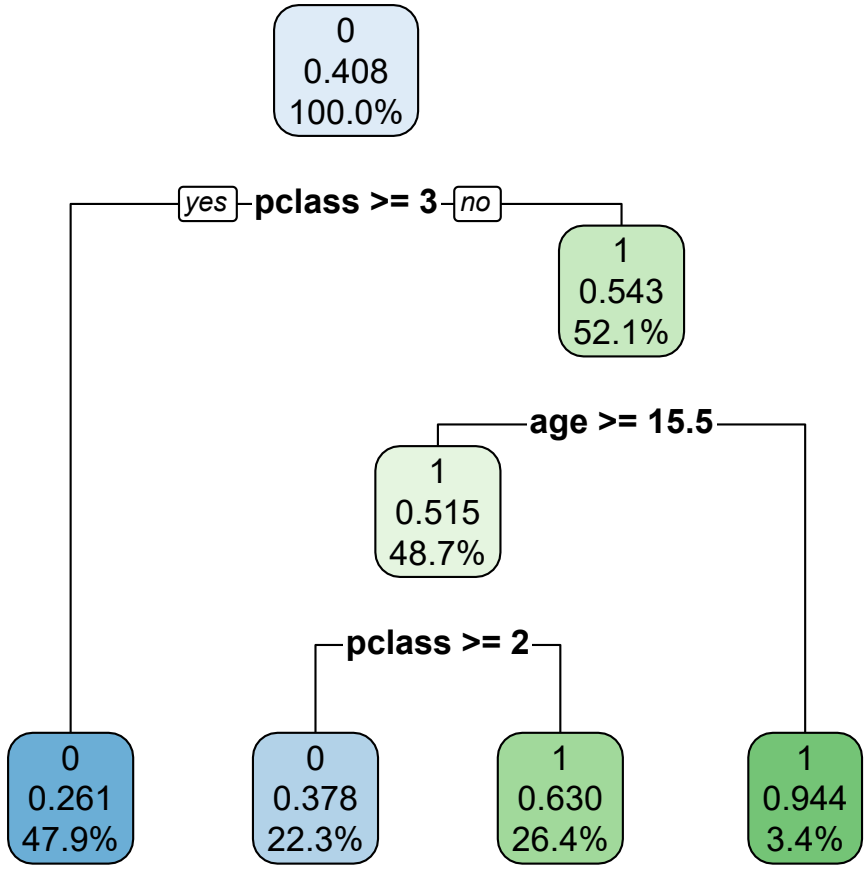
```
## Rows: 1,046
## Columns: 5
## $ survived                 <dbl> 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, ~
## $ class_3                  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ class_1_or_2_age_below_16 <dbl> 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ class_1_age_above_16      <dbl> 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ class_2_age_above_16      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

Representing trees as linear regressions

Now, we can estimate a linear model using our set of partition dummies and compare the results to our tree.

```
lm(survived ~ . -1, data = titanic_lm) %>%
  tidy() %>%
  select(term, estimate, std.error)
```

```
## # A tibble: 4 x 3
##   term                        estimate std.error
##   <chr>                          <dbl>     <dbl>
## 1 class_3                        0.261    0.0204
## 2 class_1_or_2_age_below_16      0.944    0.0762
## 3 class_1_age_above_16           0.630    0.0275
## 4 class_2_age_above_16           0.378    0.0299
```



Random Forests

Trees: pros and cons

Pros:

- Intuitive (more than regression?)
- Interpretable
- Nonparametric (no bookkeeping)

Cons:

- Overfit
- Poor predictive performance (typically)

Random forests: Basic Idea

- Breiman (2001): Instead of using a single tree, average the predictions of several trees, fitted to bootstrapped training samples + use a subset of the feature space for each split.
- Intuition: Reduce variance (overfit) by averaging multiple noisy and weakly-correlated predictions.

The random forests algorithm

Suppose B is the number of bootstrapped samples, i.e., the number of trees in the forest (typically thousands.)

For $b = 1 \dots B$:

1. Sample with-replacement n observations from the data.

2. Grow a tree T_b-, where for each split, draw a subset of m features (a common choice is $m \approx \sqrt{p}$, where p is the dimension of x.)

3. Use typical tree model stopping criteria to determine when a tree is complete (but do not prune.)

Making predictions

Regression forests:

- For each observation, predict based on the average of B predictions, i.e.,

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

where $\hat{f}^{*b}(x)$ is the prediction based on sample b.

Classification forests:

- For each test observation, record the class predicted by each of the B trees, and take a majority vote.

Out-of-bag error estimation

- Because pf bootstrap, on average, each bagged tree makes use of around two-thirds of the observations.
- Thus, we can use the remaining on-third observations as "out-of-bag" (OOB) validation set.
- This will yield, on average, B predictions for the i^{th} observation that belong to the OOB set.

Fitting forests using `ranger`

Fitting forests in R is easy with the {ranger} package, which uses the same syntax as {rpart}.

```
rf_fit <- ranger(
  formula = medv ~ .,
  data = boston,
  mtry = 3,
  num.trees = 1000,
  importance = "impurity"
)
```

where

- `num.trees` is the argument for B, the number of trees.
- `mtry` is the argument for m, the number of features drawn before each split.

Note that the `importance` argument will be latter used to construct variable importance measures.

The output of the model

```
rf_fit
```

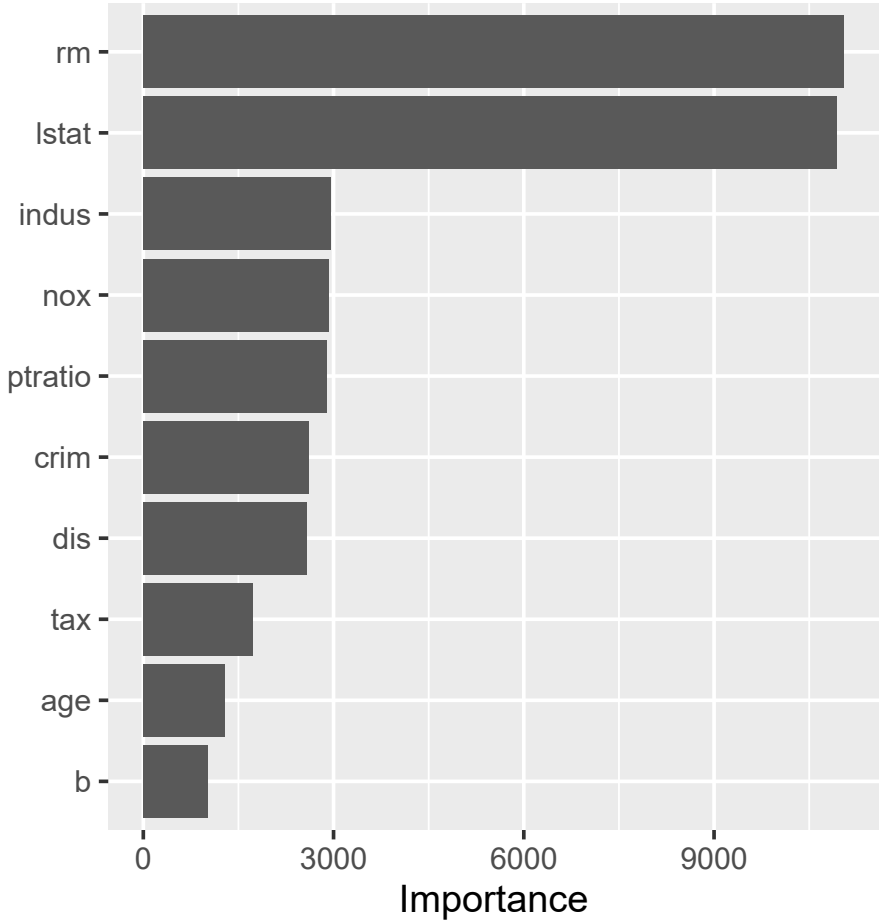
```
## Ranger result
##
## Call:
##  ranger(formula = medv ~ ., data = boston, mtry = 3, num.trees = 1000,      importance = "impurity")
##
## Type:                             Regression
## Number of trees:                  1000
## Sample size:                      506
## Number of independent variables:  13
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       10.3489
## R squared (OOB):                  0.8776533
```

Variable importance

The idea is the same as in trees, only now we average the effect of a variable over the B trees.

```
rf_fit %>%
  vip()
```

According to our forest, both `lstat` and `rm` outperform the other features.



Other Ensemble Methods

Bagging and Boosting

- Bagging (Bootstrap Aggregating): same as random forest, only $m = p$. Inferior to random forests since trees are correlated.

- Boosting: This is an example of a *slow learner* where each tree is grown using information from previously grown trees. (can be estimated using the `{gbm}` package.)

Note: Boosting algorithms in general are very popular and are on the high-end of predictive performance.

`slides::end()`

Source code

References

Breiman L, Friedman J, Olshen R, Stone C (1984). *Classification and Regression Trees*. Chapman and Hall, New York.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5-32.

Varian, H. R. (2014). Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2), 3-28.