Data Science for Economics

Lecture 3: Tree based methods

Dawie van Lill

Packages and topics

Here are the packages that we will be using for this session.

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(dplyr, ggplot2, rpart, caret, rpart.plot,
               vip, pdp, doParallel, foreach,
               ipred, ranger, gbm, xgboost, AmesHousing)
```

Many new packages -> several new topics covered.

1. Decision trees
2. Bagging
3. Random forests
4. Gradient boosting

We will probably take **two lecture slots** to get through all these topics.

Important that you go and read on these specific topics if they interest you.

Decision trees

Partition feature space into smaller regions with similar response values.

Use a set of **splitting rules** to decide regions.

Referred to as divide-and-conquer methods.

Rules can be easily interpreted and visualised with *tree diagrams*.

Decision trees become more powerful when combined with ensemble algorithms.

We will first develop strong foundation in decision trees.

CART

Classification and regression tree (**CART**) algorithm.

Decision tree partitions the training data into homogeneous subgroups and then fits a simple constant in each subgroup.

The subgroups (also called *nodes*) are formed recursively using binary partitions formed by asking simple yes-or-no questions about each feature.

Done a number of times until a suitable stopping criteria is satisfied.

After partitioning, model predicts output based on

1. Average response values for all observations that fall in that subgroup
2. Class that has majority representation

First is regression problem, second is classification.

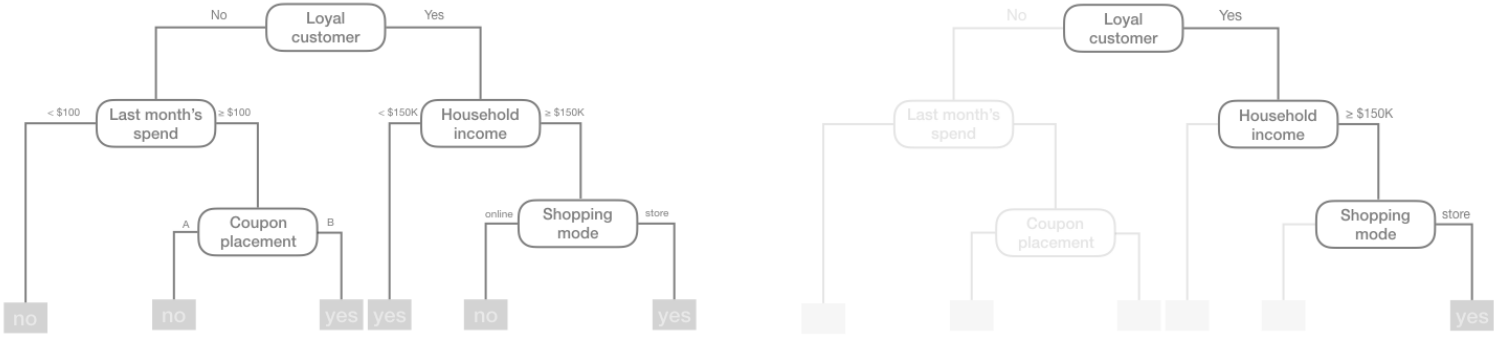


Figure 1: Decision tree depicting whether consumer will redeem a coupon.

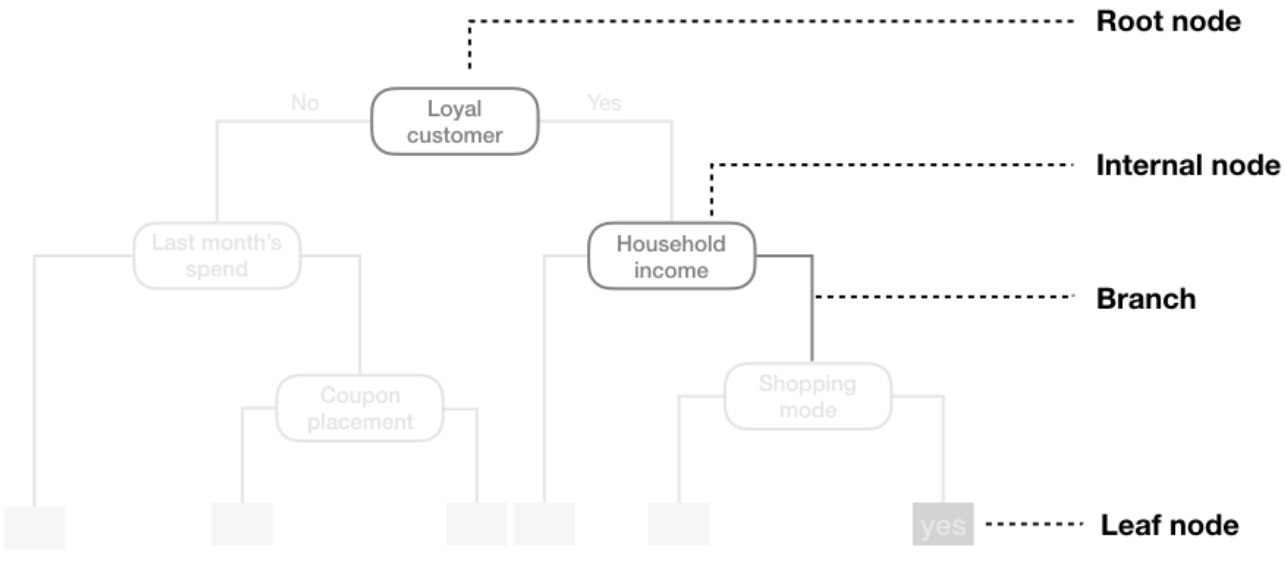


Figure 2: Terminology of a decision tree.

Partitioning

CART uses binary recursive partitioning -- each split depends on the split above it.

Objective is to find the best feature to partition the remaining data into two regions (R_1 and R_2).

Want to minimise overall error between response and predicted constant.

In regression problems, minimise total SSE as follows

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

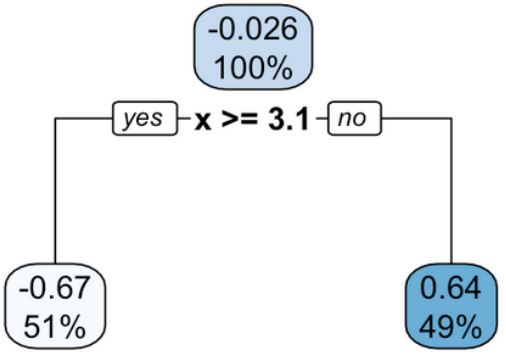
Having found best feature/split combination, data are partitioned again into two regions and the splitting process is repeated.

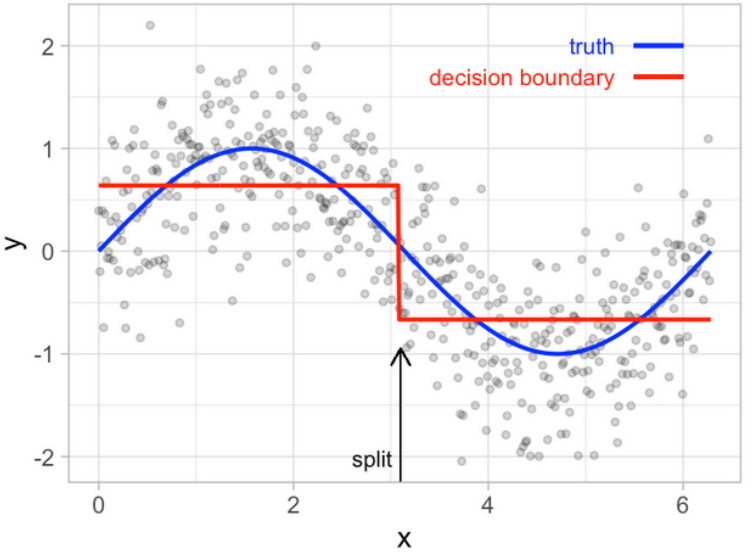
Process is continued until stopping criterion is reached (max depth).

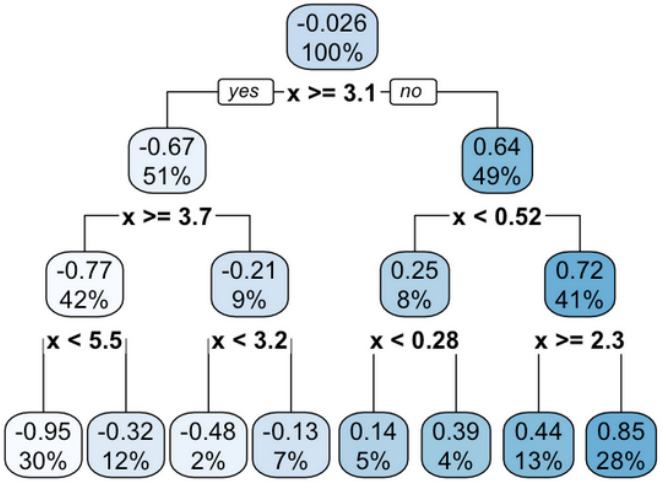
Creating a dataset

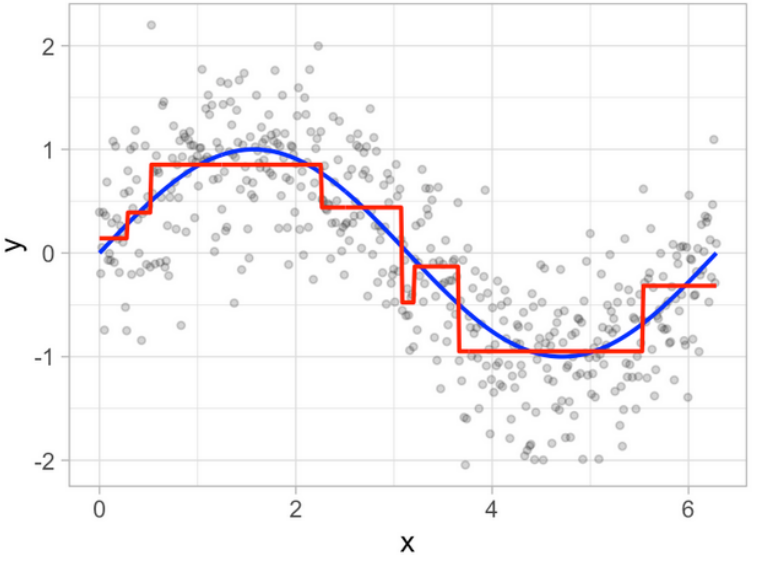
```
# create data
set.seed(1112)  # for reproducibility
df <- tibble::tibble(
  x = seq(from = 0, to = 2 * pi, length = 500),
  y = sin(x) + rnorm(length(x), sd = 0.5),
  truth = sin(x)
)

# run decision stump model
ctrl <- list(cp = 0, minbucket = 5, maxdepth = 1)
fit <- rpart(y ~ x, data = df, control = ctrl)
```

Here we have data generated from simple $\sin$ function with Gaussian noise.


-0.026
100%
yes
x >= 3.1
no
-0.67
51%
0.64
49%


truth
decision boundary
split
y
x
2
1
0
-1
-2
0
2
4
6


-0.026
100%
yes
x >= 3.1
no
-0.67
51%
0.64
49%
x >= 3.7
x < 0.52
-0.77
42%
-0.21
9%
0.25
8%
0.72
41%
x < 5.5
x < 3.2
x < 0.28
x >= 2.3
-0.95
30%
-0.32
12%
-0.48
2%
-0.13
7%
0.14
5%
0.39
4%
0.44
13%
0.85
28%

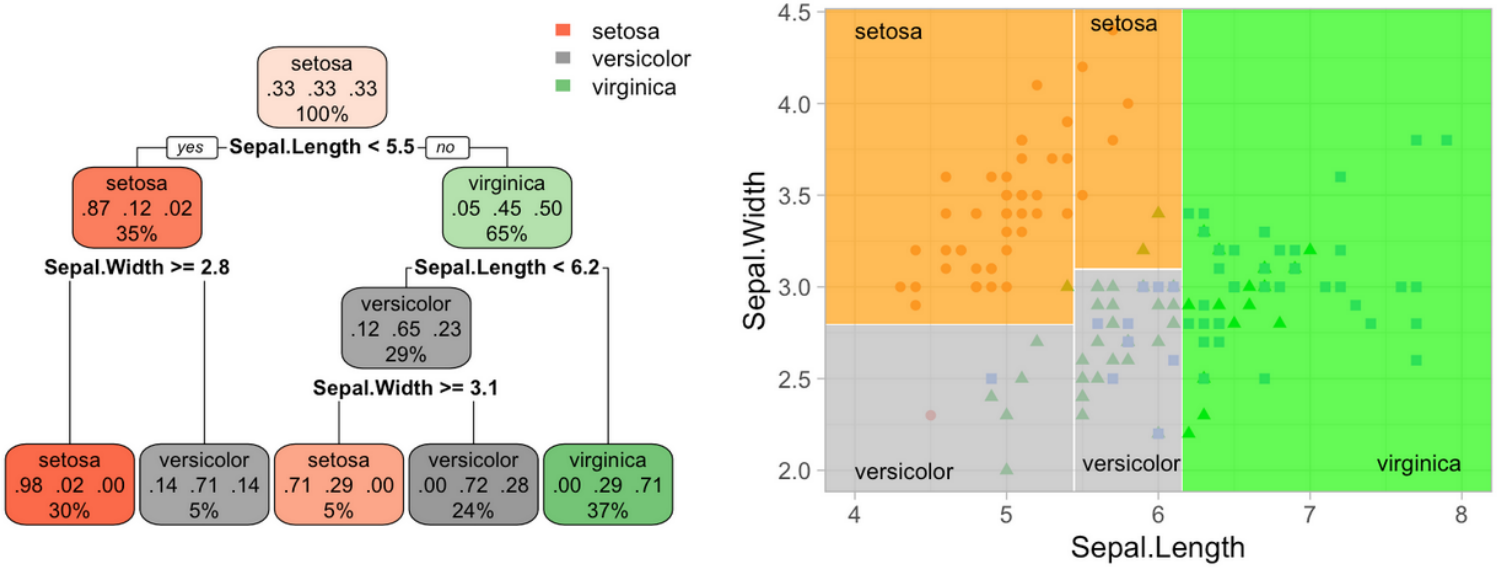

y
x
2
1
0
-1
-2
0
2
4
6

Classification problem

Decision tree applied to iris data set.

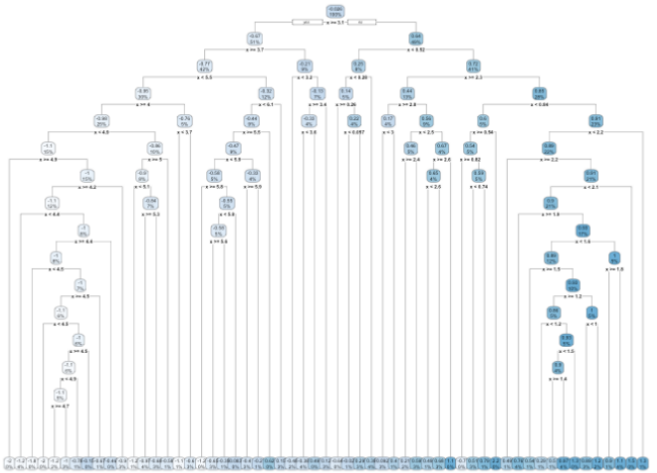
Species of flower predicted based on two features (sepal width and sepal length).

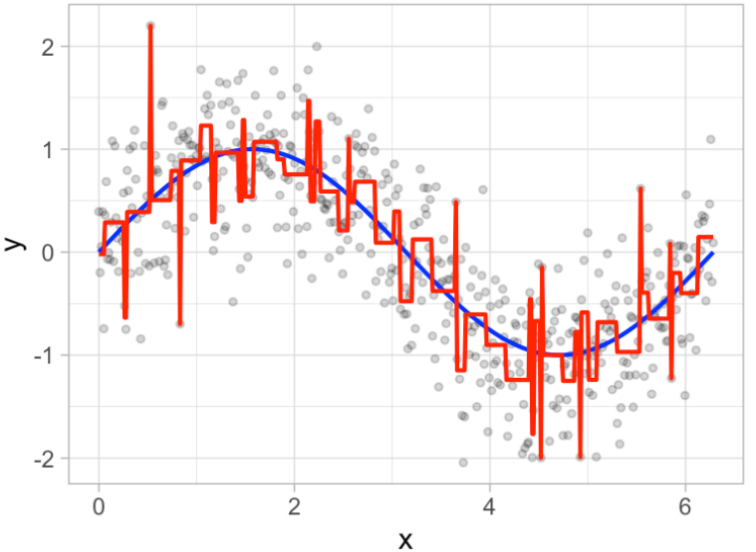
Optimal decision tree with two splits in each feature.



Predicted value is response class with greatest proportion within enclosed region.

How complex should tree be?





Don't want to overfit the data, so there is some balance we need to maintain.

Two approaches to find this balance.

1. Early stopping
2. Pruning

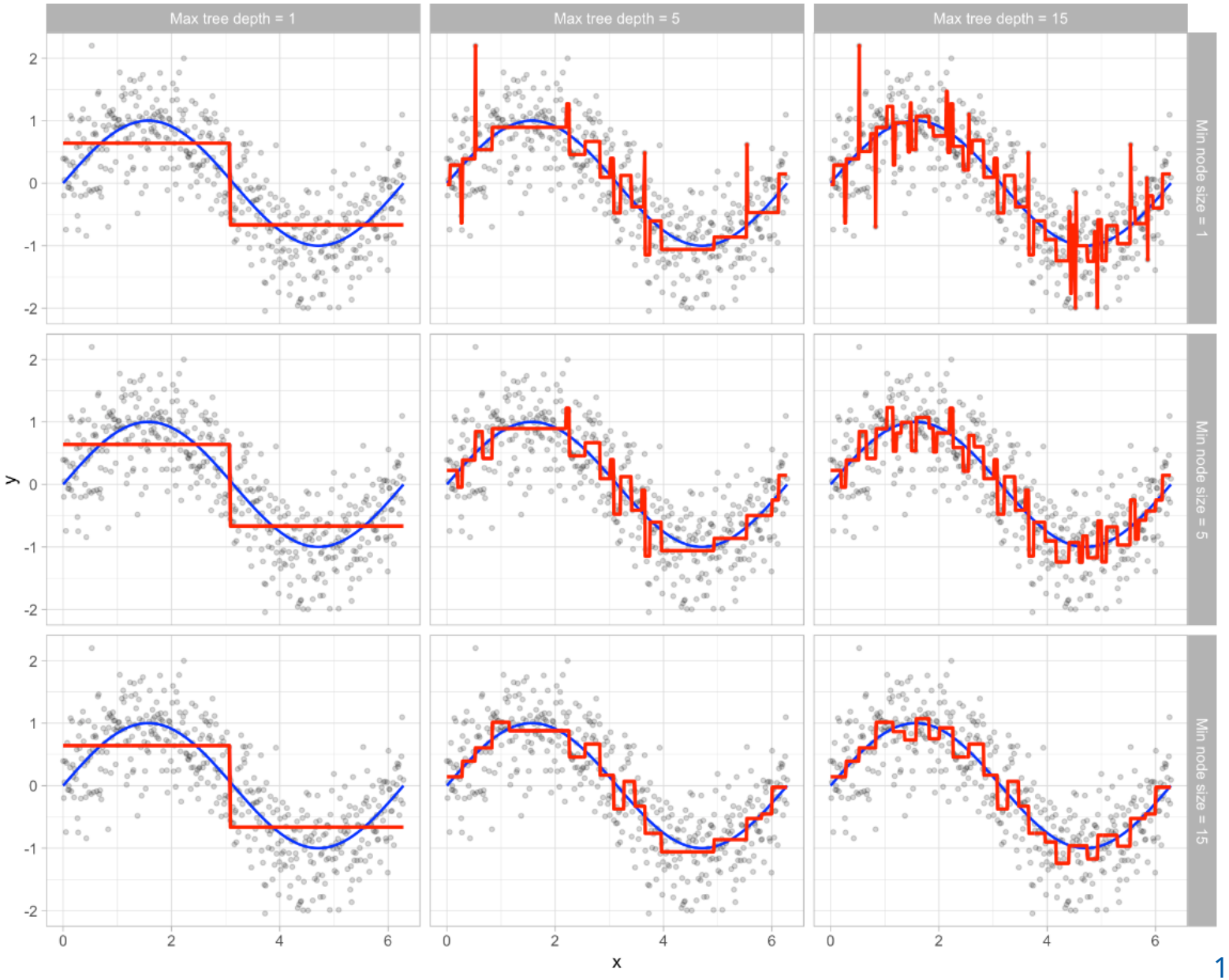
Early stopping

Early stopping explicitly restricts the growth of the tree.

Two most common approaches are

1. Restrict the tree depth to a certain level
2. Restrict the minimum number of observations allowed in any terminal node

Two methods can be operated independently or jointly (see figure on next slide).


Max tree depth = 1
Max tree depth = 5
Max tree depth = 15
Min node size = 1
Min node size = 5
Min node size = 15
y
x
2
1
0
-1
-2
0
2
4
6

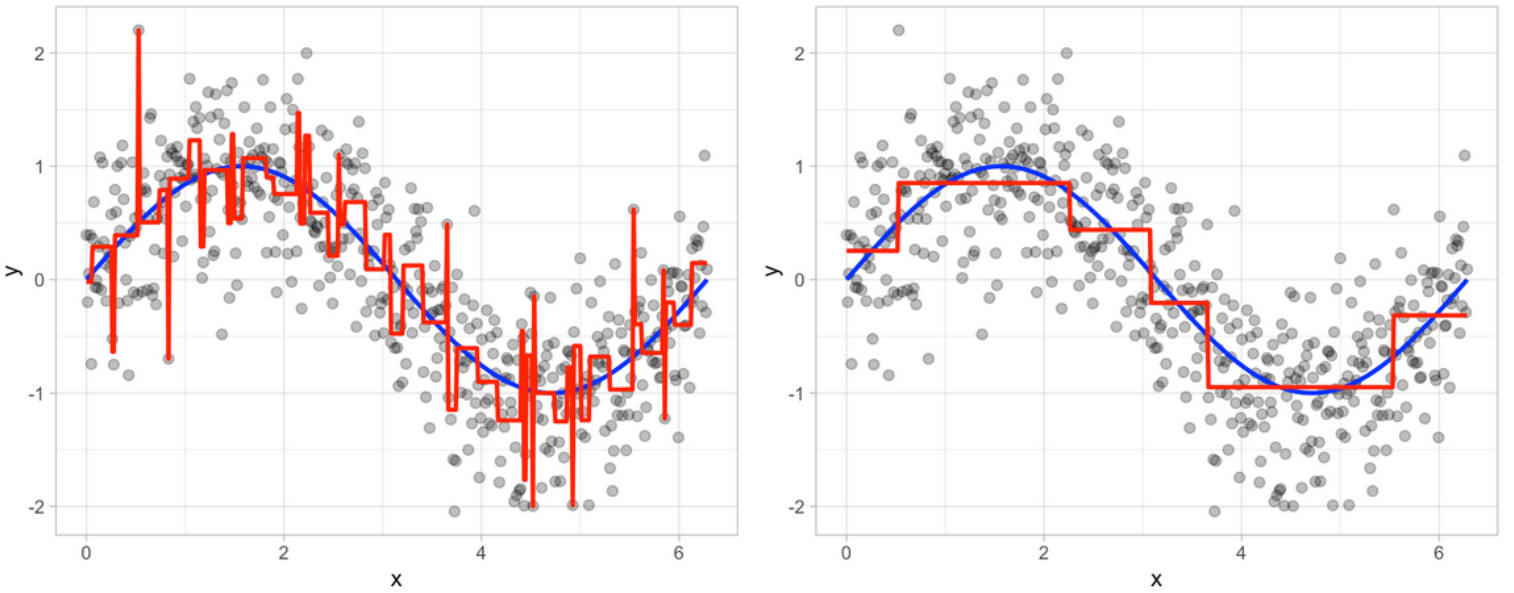
Pruning

Allow tree to grow large and then prune back to find optimal subtree.

Penalises objective function for number of terminal nodes in the tree.

$$\min(\text{SSE} + \alpha|T|)$$

Shares some similarity with the **lasso penalty** we discussed in the previous lecture.



Ames housing example

```
# Create training (70%) set for the Ames housing data.
set.seed(123)

ames <- AmesHousing::make_ames()
split  <- rsample::initial_split(ames, prop = 0.7,
                                 strata = "Sale_Price")
ames_train  <- rsample::training(split)

ames_dt1 <- rpart(
  formula = Sale_Price ~ .,
  data    = ames_train,
  method  = "anova"
)
```

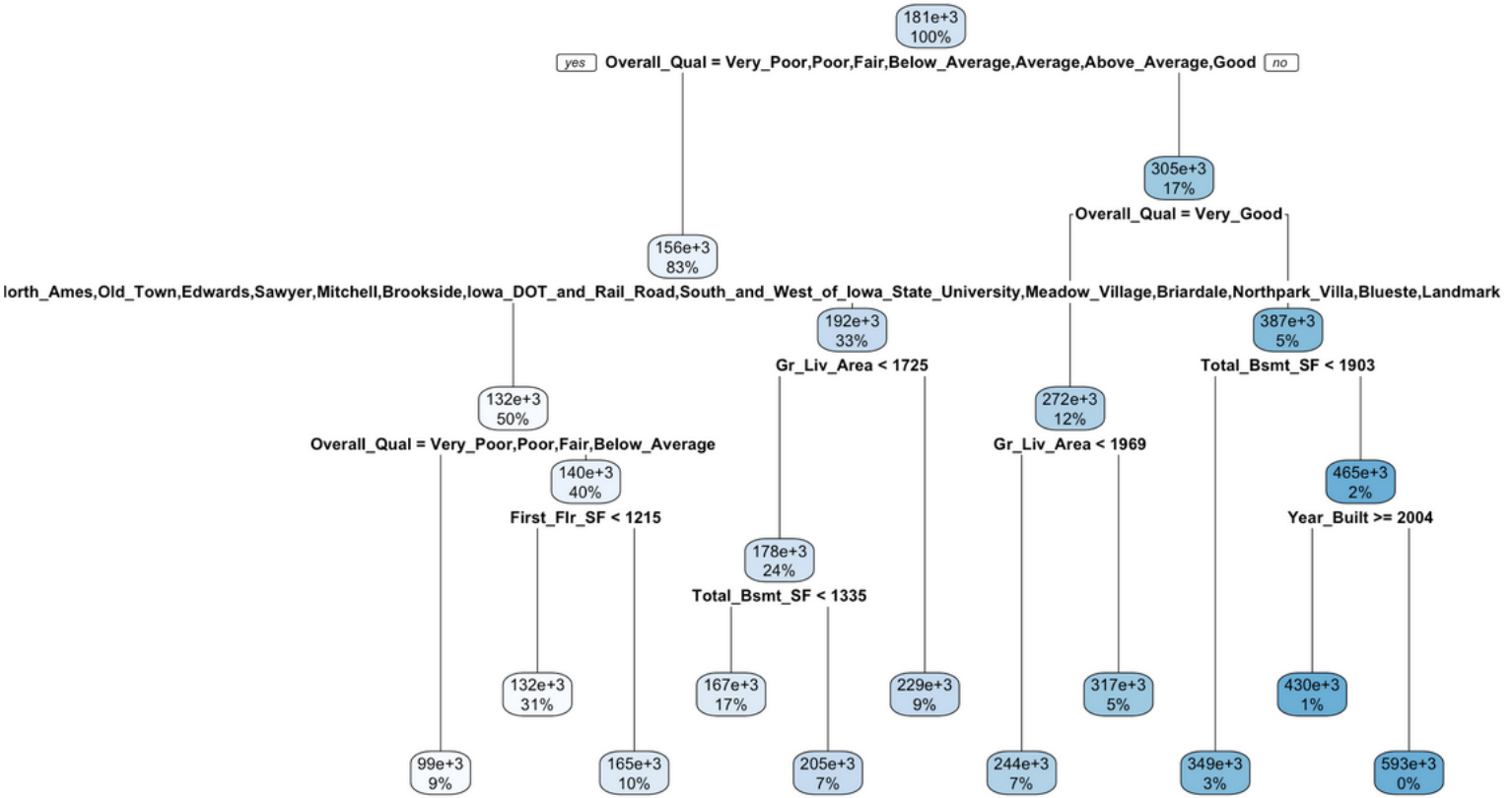
For the decision tree we want to use the `anova` method, instead of `lm`.

`rpart` will make an educated guess as to the method, but we can specify if we want.

```
ames_dt1
```

```
## n= 2053
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 2053 1.321794e+13 180996.30
##    2) Overall_Qual=Very_Poor,Poor,Fair,Below_Average,Average,Above_Average,
##      4) Neighborhood=North_Ames,Old_Town,Edwards,Sawyer,Mitchell,Brookside,
##        8) Overall_Qual=Very_Poor,Poor,Fair,Below_Average 199 1.792954e+11
##        9) Overall_Qual=Average,Above_Average,Good 823 8.762399e+11 140409.0
##         18) First_Flr_SF< 1089 517 2.905312e+11 129244.00 *
##         19) First_Flr_SF>=1089 306 4.123757e+11 159272.60 *
##      5) Neighborhood=College_Creek,Somerset,Northridge_Heights,Gilbert,Nort
##       10) Gr_Liv_Area< 1477 287 2.508268e+11 165395.90 *
##       11) Gr_Liv_Area>=1477 413 6.302277e+11 212054.00
##         22) Total_Bsmt_SF< 959.5 199 1.390877e+11 192493.10 *
##         23) Total_Bsmt_SF>=959.5 214 3.441912e+11 230243.70 *
##    3) Overall_Qual=Very_Good,Excellent,Very_Excellent 331 2.936700e+12 3060
##      6) Overall_Qual=Very_Good 231 9.469746e+11 270626.10
##       12) Gr_Liv_Area< 1919 142 3.349783e+11 244016.60 *
##       13) Gr_Liv_Area>=1919 89 3.510308e+11 313081.60 *
##      7) Overall_Qual=Excellent,Very_Excellent 100 1.029126e+12 387948.00
##       14) Total_Bsmt_SF< 1907.5 72 3.149859e+11 350532.40 *
##       15) Total_Bsmt_SF>=1907.5 28 3.541599e+11 484159.40 *
```

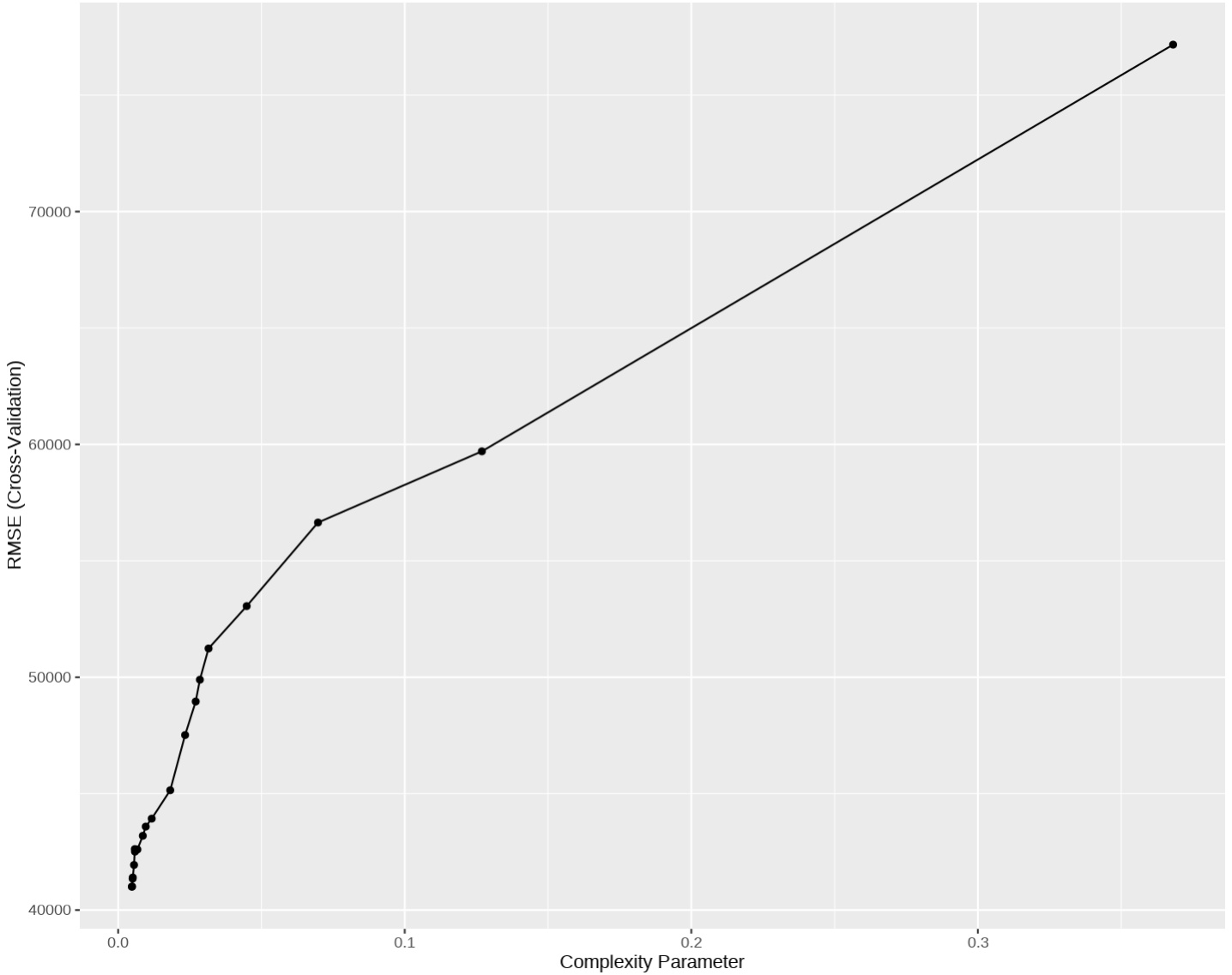
Pruned decision tree (Ames)



```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainIn
## There were missing values in resampled performance measures.
```

```
ames_dt2
```

```
## CART
##
## 2053 samples
##   80 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1848, 1849, 1848, 1848, 1847, 1848, ...
## Resampling results across tuning parameters:
##
##   cp           RMSE      Rsquared   MAE
##   0.004769378  40999.96  0.7471767  27853.78
##   0.004857895  41009.43  0.7472357  27880.91
##   0.005044889  41349.04  0.7429694  28146.47
##   0.005101876  41398.15  0.7423978  28191.61
##   0.005534858  41937.05  0.7356224  28625.93
##   0.005816673  42618.42  0.7284863  29133.73
##   0.005867977  42508.86  0.7287411  29157.66
##   0.006705522  42599.97  0.7259067  29512.07
##   0.008595390  43188.78  0.7191325  29899.85
##   0.009626139  43584.13  0.7144605  30349.24
##   0.011710056  43926.06  0.7085772  31102.29
```


RMSE (Cross-Validation)
70000
60000
50000
40000
0.0
0.1
0.2
0.3
Complexity Parameter

Bagging

Bootstrap aggregating (bagging) will be our first ensemble algorithm.

Model averaging is applied to reduce variance and minimise overfitting.

Usually applied to **decision trees**!

With bagging b bootstrap copies of the original training data are created.

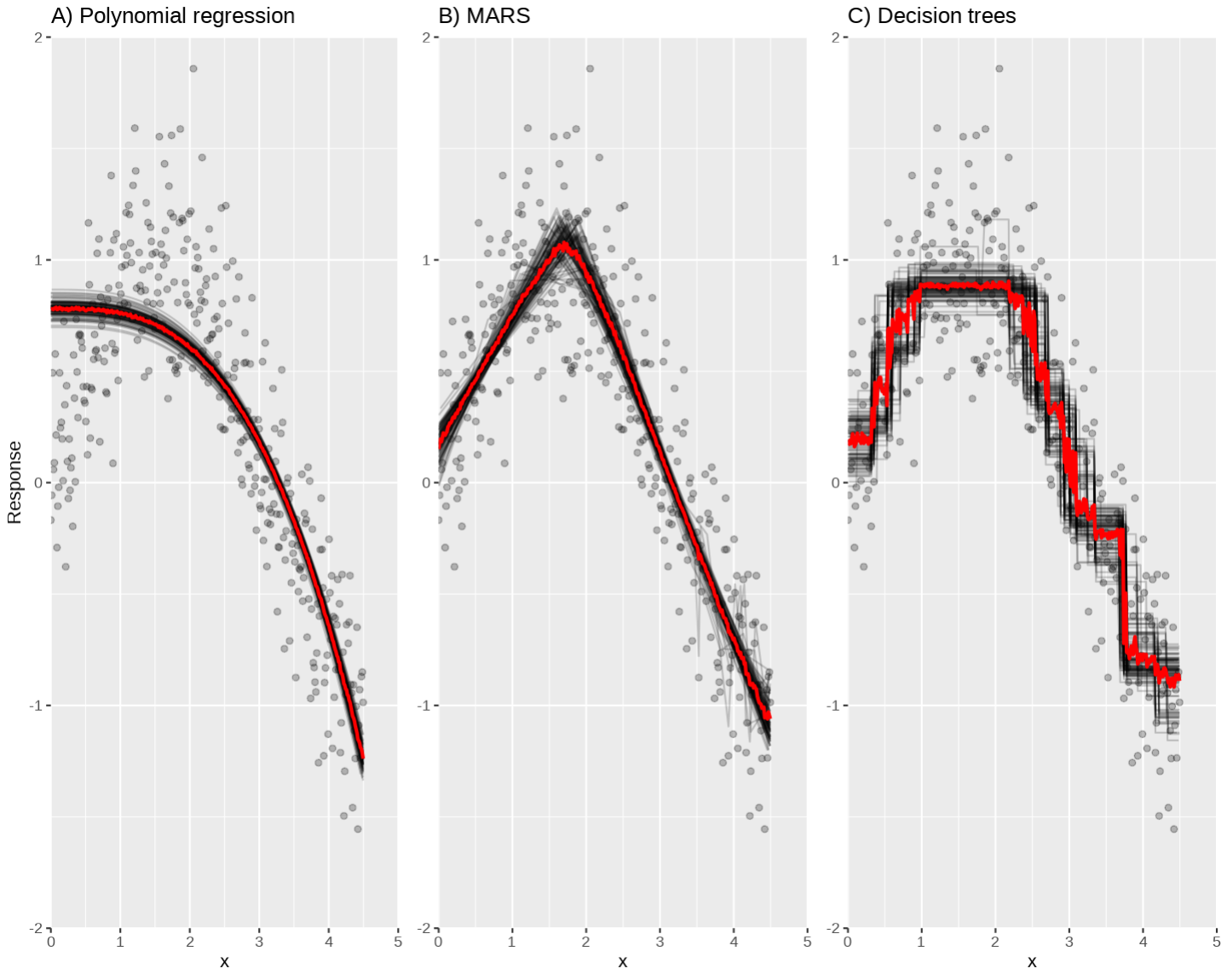
Regression or classification algorithm is applied to each bootstrap sample.

- For regression, new predictions are made by averaging predictions
- For classification, predictions combined by averaging estimated class.

Equation below shows bagged prediction and prediction from regression or classification algorithms.

$$\widehat{f_{bag}} = \widehat{f_1}(X) + \widehat{f_2}(X) + \cdots + \widehat{f_b}(X)$$

Bagging works well for high variance algorithms such as decision trees and KNN.


A) Polynomial regression
B) MARS
C) Decision trees
Response
x
2
1
0
-1
-2
0
1
2
3
4
5

Housing example

Results from decision trees on our Ames example were not too encouraging.

Even after tuning to find best pruned tree, performance not great.

Move from using single pruned decision tree to 100 bagged unpruned trees.

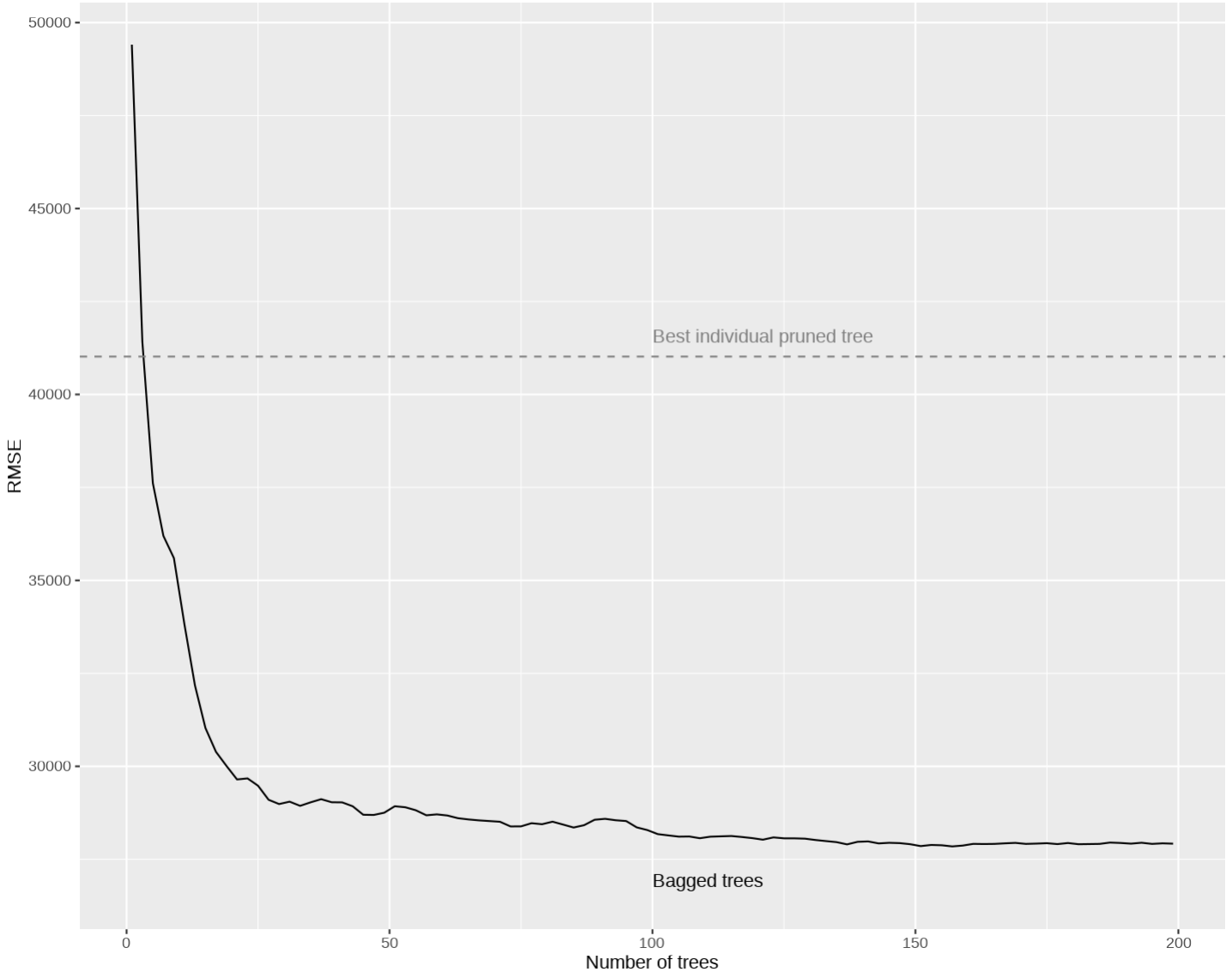
Not pruning keeps variance high and bias low.

```
# make bootstrapping reproducible
set.seed(123)

# train bagged model
ames_bag1 <- bagging(
  formula = Sale_Price ~ .,
  data = ames_train,
  nbagg = 100,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0)
)
```

Housing example -- contd.

```
ames_bag1
```

```
##
## Bagging regression trees with 100 bootstrap replications
##
## Call: bagging.data.frame(formula = Sale_Price ~ ., data = ames_train,
##     nbagg = 100, coob = TRUE, control = rpart.control(minsplit = 2,
##         cp = 0))
##
## Out-of-bag estimate of root mean squared error:  27767.13
```


50000
45000
40000
35000
30000
RMSE
Best individual pruned tree
Bagged trees
0
50
100
150
200
Number of trees

Next time

Next class will be our last lecture focused exclusively on machine learning.

We might mention a model or two as we progress, but most of our focus will be on **data science** topics.

In the next lecture we cover two topics.

1. Random forests
2. Gradient boosting

It would be great to cover support vector machines and neural networks at some stage. Let us see how far we get with the other sections though.