Machine Learning in R

Lecture 5 – Tree-based methods

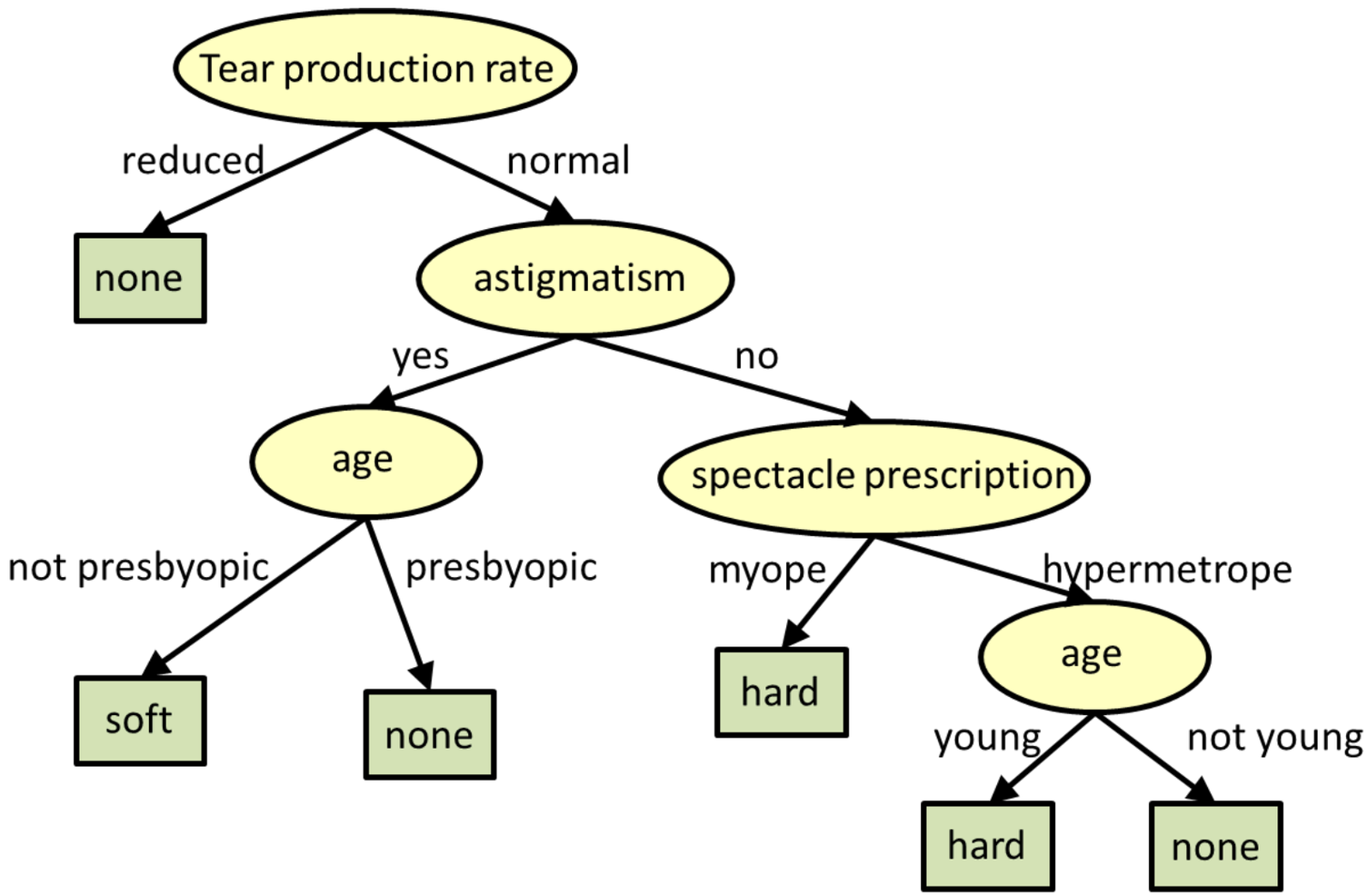
Jean Feng – Epidemiology and Biostatistics

Today's Lecture

- Decision Trees
- Bagging
- Random Forests
- Variable Importance

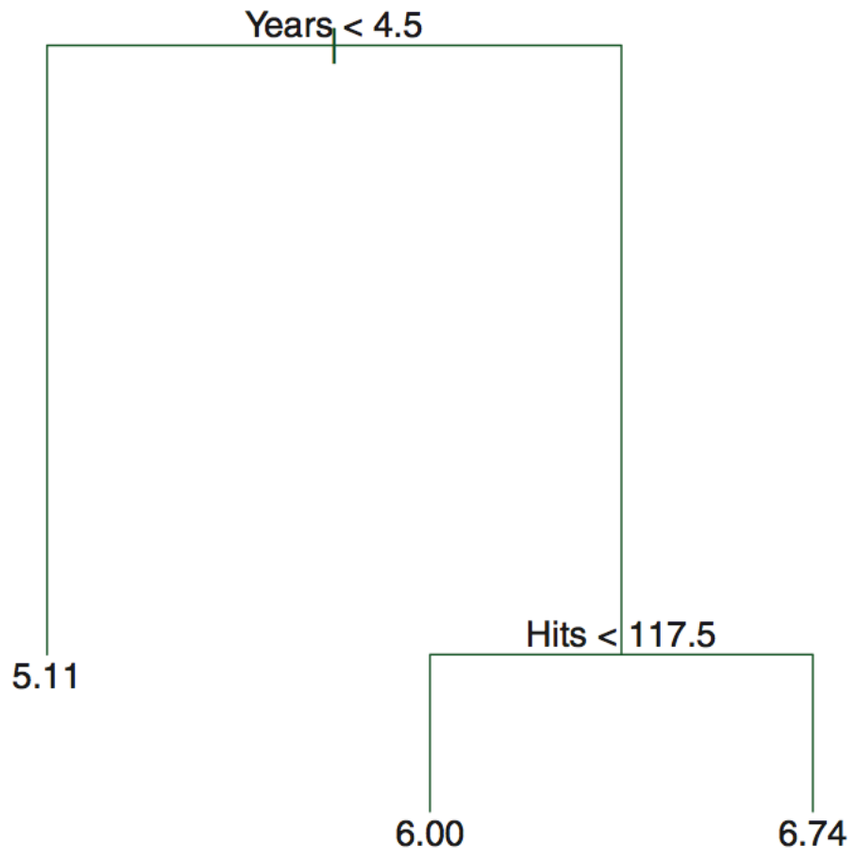
Tree-based methods

- An example tree for deciding what kind of contact lens a person should wear:



Toy Example: Predicting salaries of baseball players

- Outcome: Salary of baseball players (in millions)
- Two predictors: years of experience in MLB (Years) and number of hits made in the previous year (Hits)



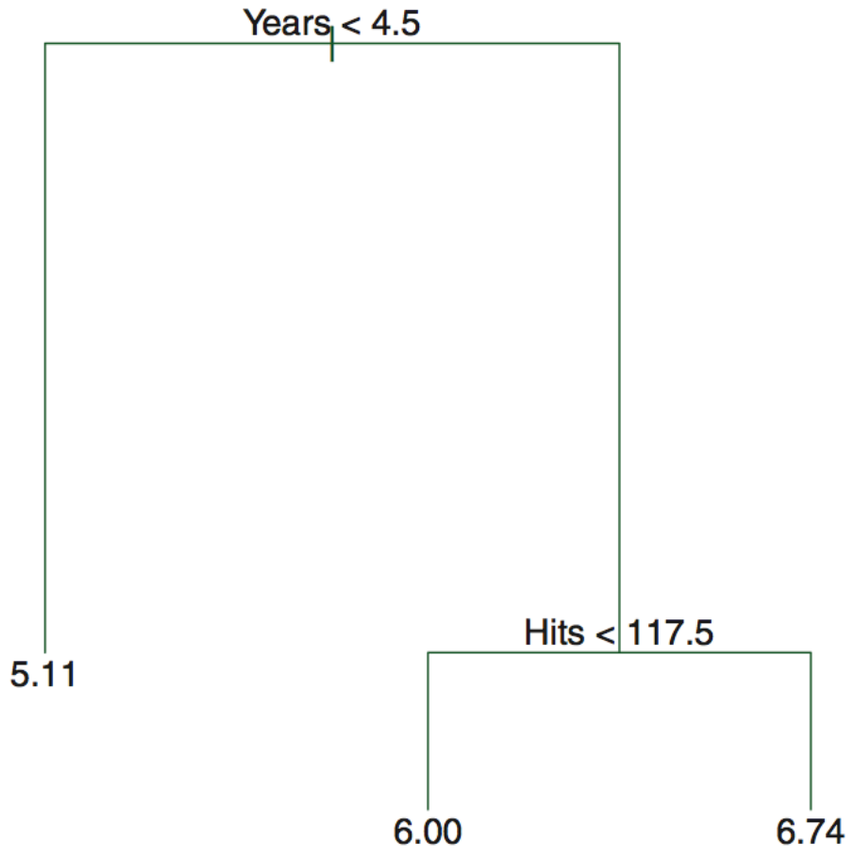
Toy Example: Predicting salaries of baseball players

- The top split predicts that baseball players having Years < 4.5 earn $5.11 million.

- Among players with > 4.5 Years of experience...

- For those with < 117.5 Hits, the predicted salary is $6.00 million.

- For those with > 117.5 Hits, the predicted salary is $6.74 million.



Decision Trees

- Decision trees — also known as Classification and Regression Trees (CART) — can be used in both regression and classification tasks.

    - They have also been extended for survival outcomes.

- Trees are able to model non-linearities and interactions between variables very naturally.

- Trees are probably an oversimplification of the true data, but they are very intuitive and interpretable.

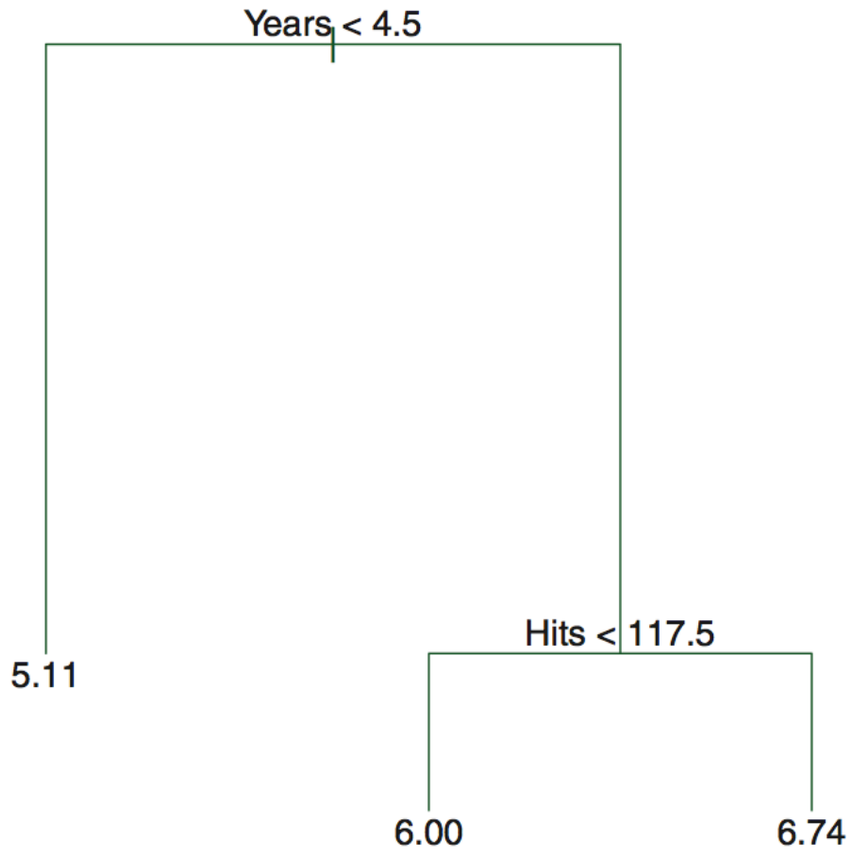
A region-based view

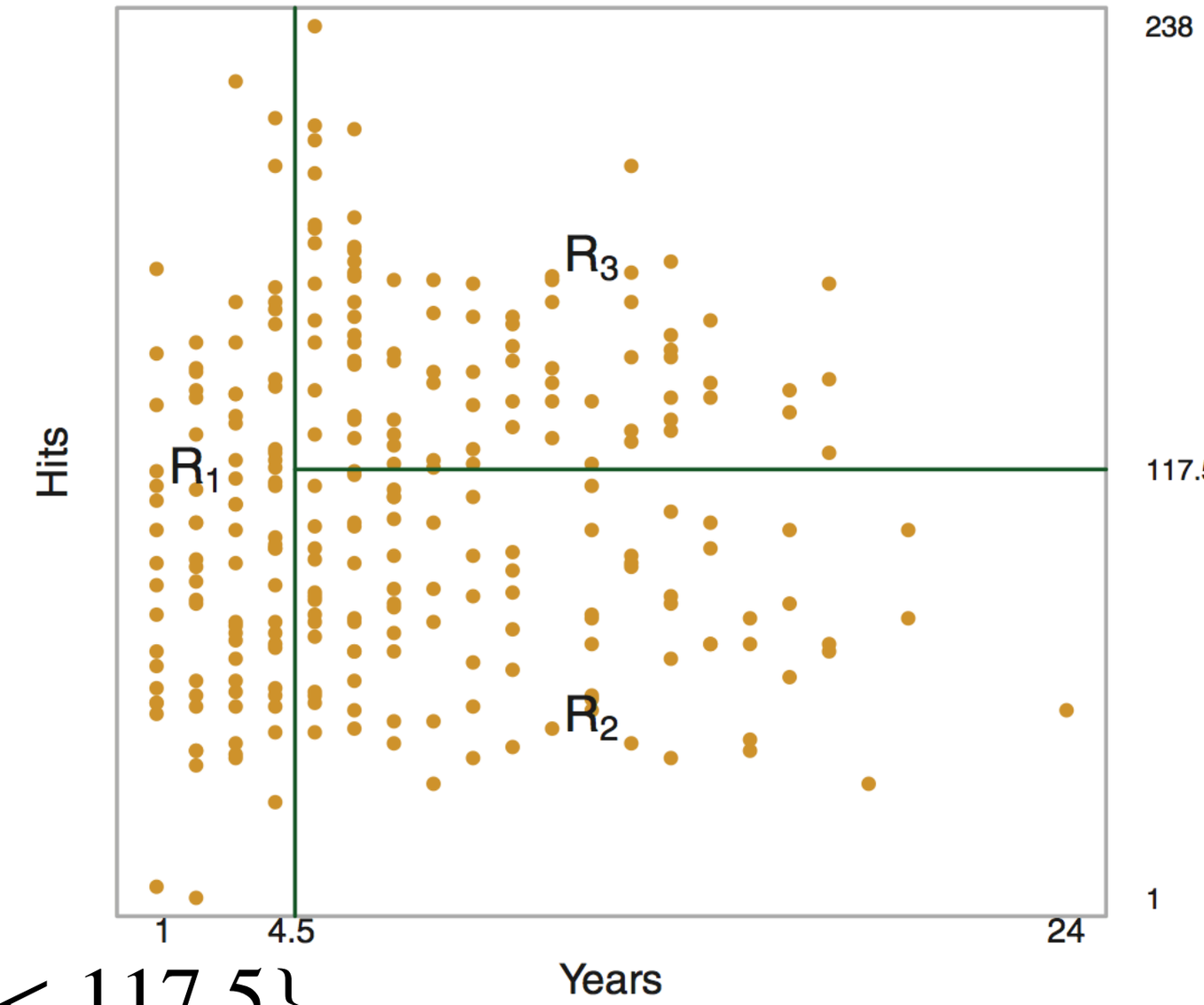
- The basic steps are:

    1. Given training data, **stratify** or **segment** the predictor space.
    2. Given a test observation, determine which segment it falls into and predict the average/most common value observed in that segment.

Toy Example: Predicting salaries of baseball players

The tree **divides the predictor space into 3 regions**





$$R_1 = \{X \mid Years < 4.5\}$$
$$R_2 = \{X \mid Years \geq 4.5, Hits < 117.5\}$$
$$R_3 = \{X \mid Years \geq 4.5, Hits \geq 117.5\}$$

A region-based view

1. **Stratify/Segment**: Partition the predictor space into *J* disjoint regions $R_1, \ldots, R_J$.

   - How should we construct the regions $R_1, \ldots, R_J$?
   - How many regions should there be? (How big should *J* be?)

2. **Prediction**: For observation X in region R_j, output the mean (regression) or mode (classification) of the observed outcomes for the training observations in region R_j

A region-based view

1. **Stratify/Segment**: Partition the predictor space into *J* disjoint regions $R_1, \ldots, R_J$.

   - **How should we construct the regions $R_1, \ldots, R_J$?**
   - How many regions should there be? (How big should *J* be?)

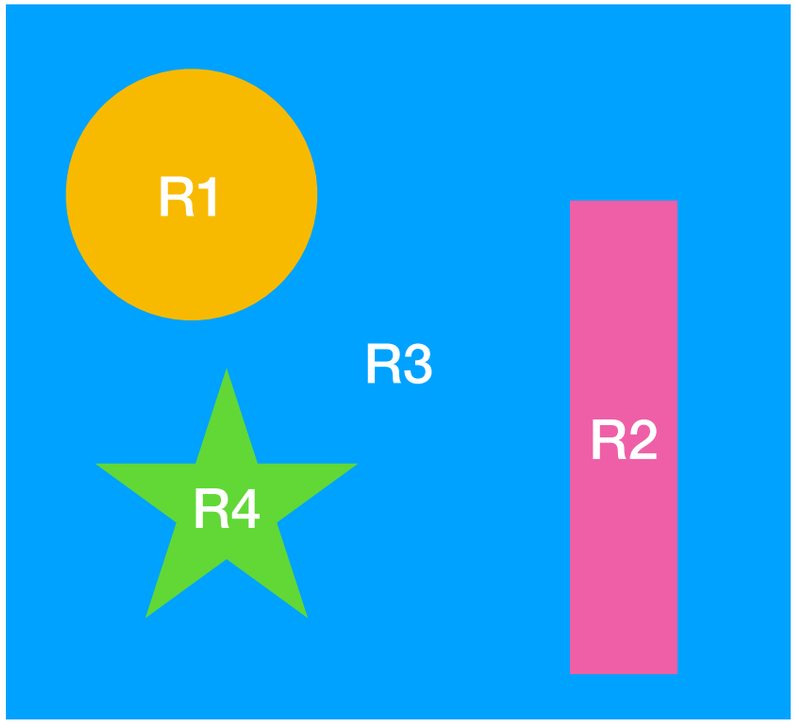
2. **Prediction**: For observation X in region R_j, output the mean (regression) or mode (classification) of the observed outcomes for the training observations in region R_j

Partitioning the Predictor Space

- For now assume *J* is known.

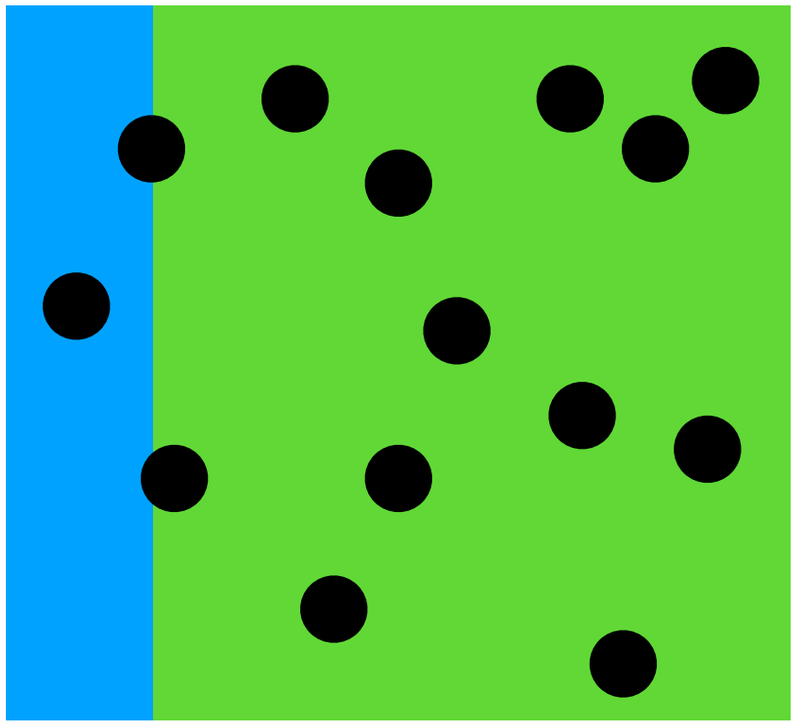
- Unfortunately, it is not possible to consider every possible partition of the space into *J* regions.

- We introduce two simplifications:
  1. We will only split the region into rectangles/boxes in a hierarchical manner.
  2. We'll create splits in a greedy fashion known as **recursive binary splitting**.



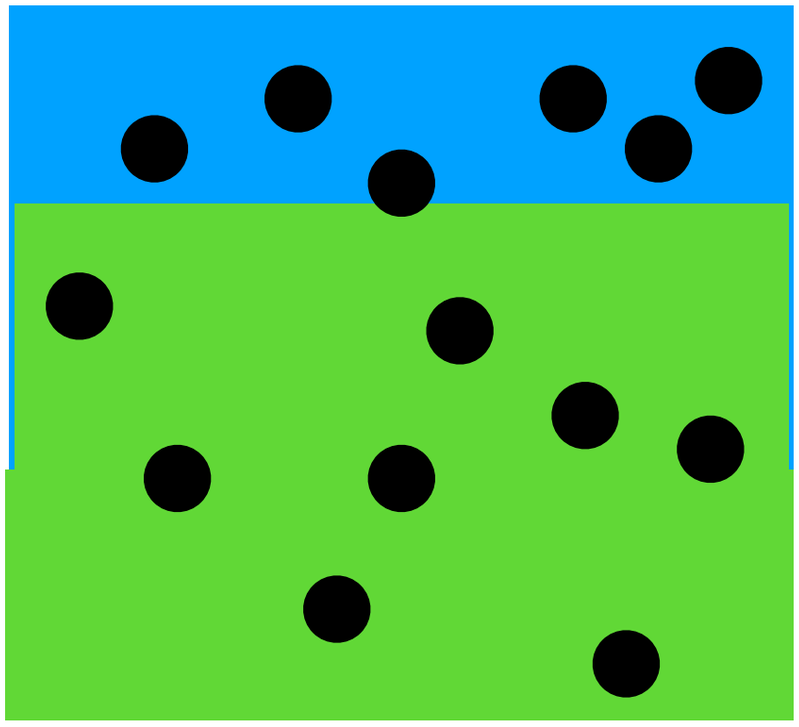
Recursive Binary Splitting

- Start with all the data.
- For all predictors X_j and cut points t:
  - Consider splitting the space into:
    $R_1(j,t) = \{X | X_j < t\}, R_2(j,t) = \{X | X_j \geq t\}$
    - Evaluate the quality of this candidate split according to some **split criterion function** (e.g. drop in mean squared error).
- Choose the best split according to this criterion function.
- Rinse and repeat the above process for each new region until there are J regions



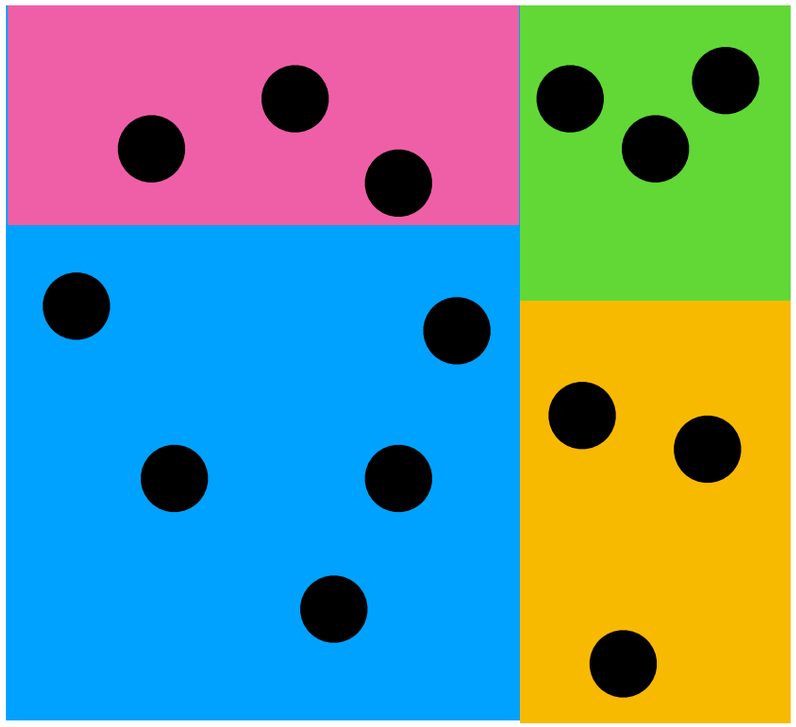
Recursive Binary Splitting

- Start with all the data.
- For all predictors X_j and cut points t:
  - Consider splitting the space into:
    $R_1(j,t) = \{X | X_j < t\}, R_2(j,t) = \{X | X_j \geq t\}$
  - Evaluate the quality of this candidate split according to some **split criterion function** (e.g. drop in mean squared error).
- Choose the best split according to this criterion function.
- Rinse and repeat the above process for each new region until there are J regions



Recursive Binary Splitting

- Start with all the data.
- For all predictors X_j and cut points t:
  - Consider splitting the space into:
    $R_1(j,t) = \{X | X_j < t\}, R_2(j,t) = \{X | X_j \geq t\}$
    - Evaluate the quality of this candidate split according to some **split criterion function** (e.g. drop in mean squared error).
- Choose the best split according to this criterion function.
- Rinse and repeat the above process for each new region until there are J regions

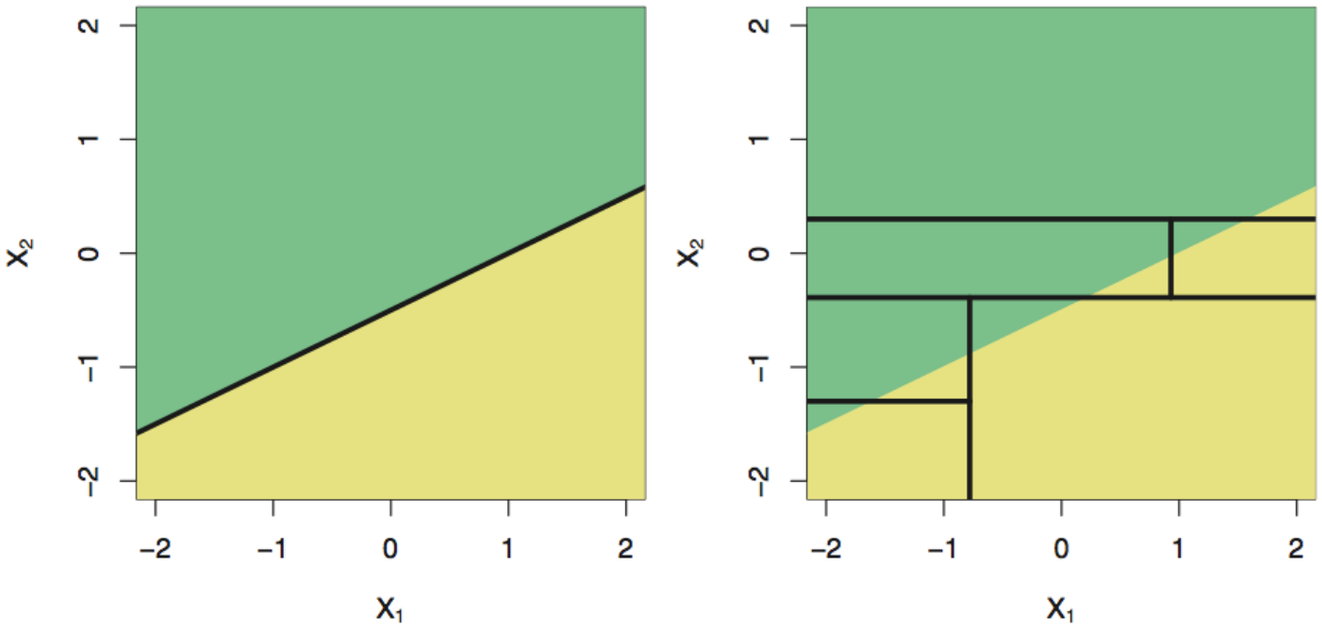


Tree versus Linear Models

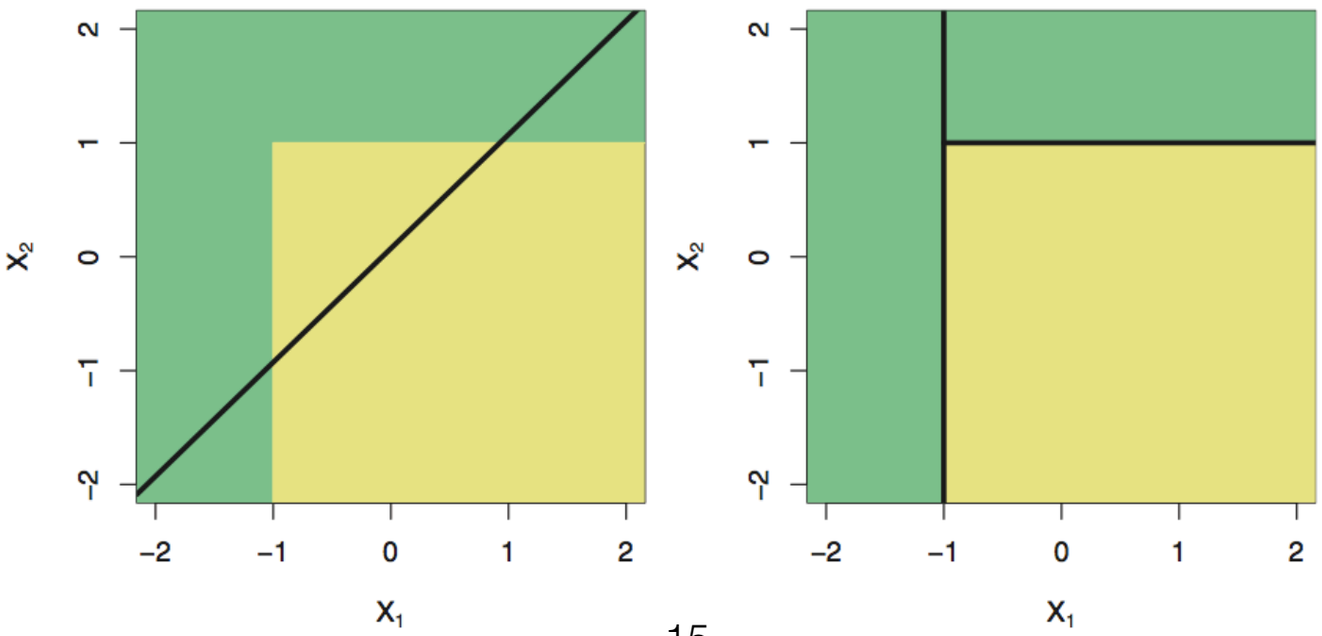
Linear model

Tree-based model

True linear boundary

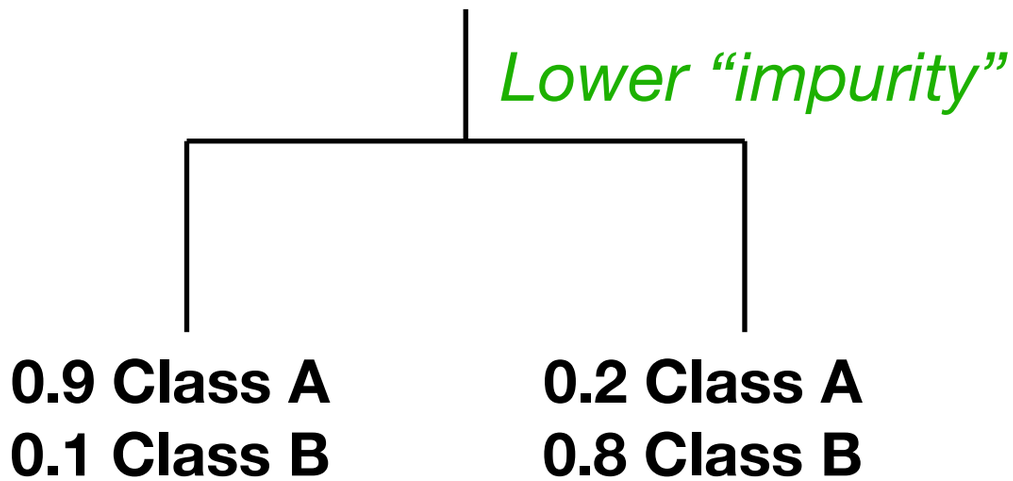


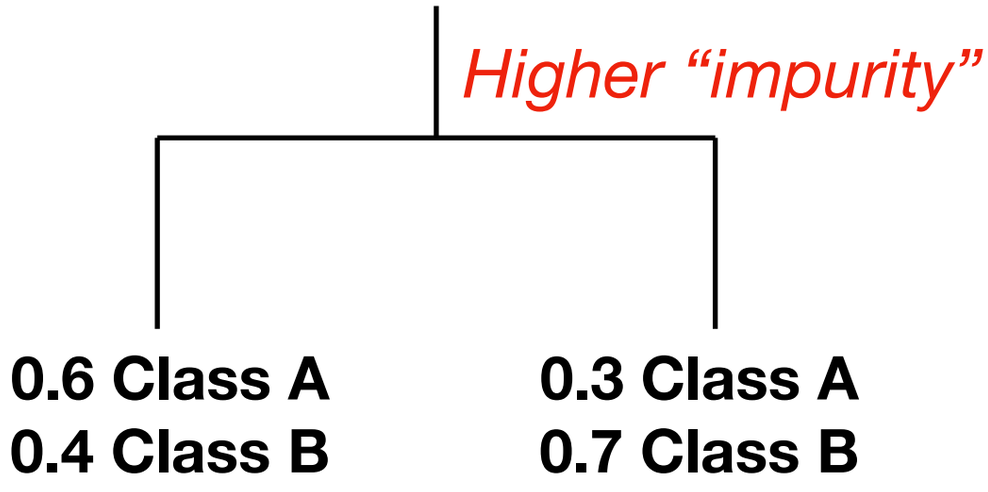
True non-linear boundary



Split Criterion Functions

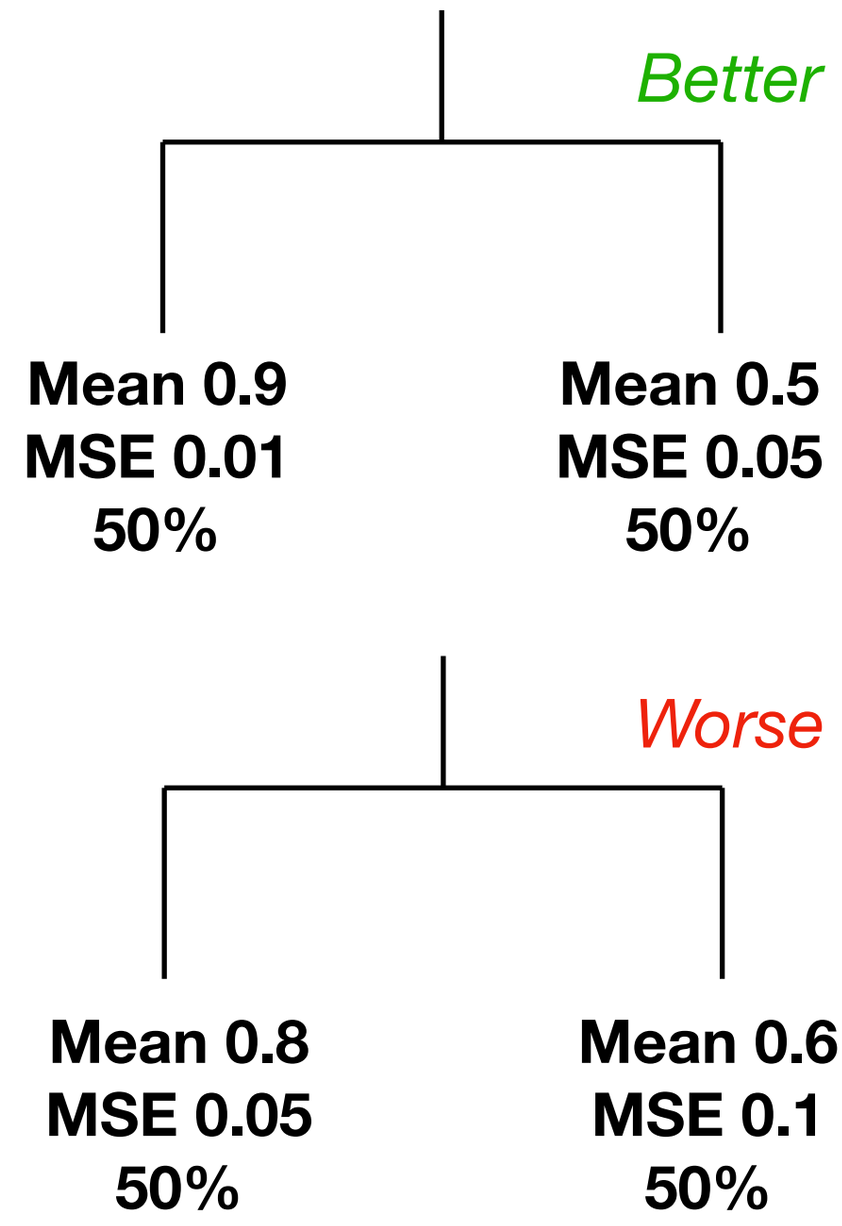
- **Classification**: Evaluate split by decrease in impurity. How often do outcomes differ from the mode for its split.
  - Gini index
  - Cross-entropy





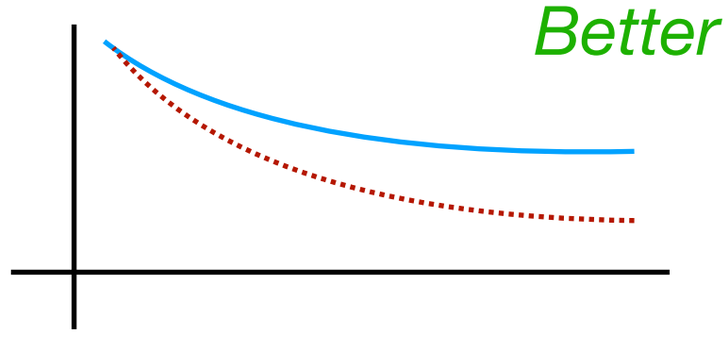
Split Criterion Functions

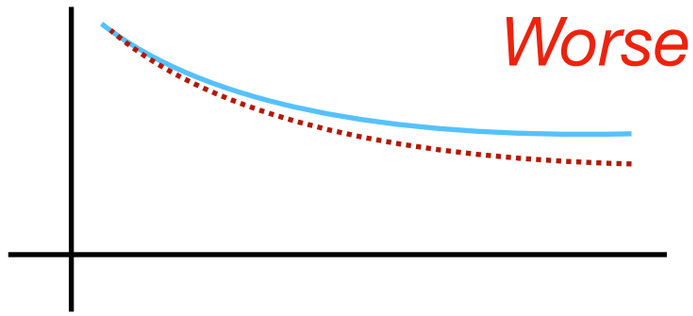
- **Regression:** Evaluate split by decrease in average deviation from the mean. How different are the outcomes from the mean for that split.
  - Squared-error deviations
  - Absolute deviations



Split Criterion Functions

- **Survival:** Evaluate split by how much we can separate the survival curves.
  - Log-rank statistic





How many candidate splits?

Q: Suppose we have two binary predictors $X_1 = \{0,1\}$ and $X_2 = \{A, B\}$. How many candidate splits do we need to search over for the first split?

A. 2 splits

B. 4 splits

C. 8 splits

Candidate splits:

$$\{X_1 < 0.5\}, \{X_1 \geq 0.5\}$$
$$\{X_2 = A\}, \{X_2 = B\}$$

How many candidate splits?

Q: Suppose we have a binary predictor $X_1 = \{0,1\}$ and a categorical predictor $X_2 = \{A, B, C\}$. How many candidate splits do we need to consider?

A. 2 splits

B. 4 splits

C. 8 splits

Candidate splits:

$$\{X_1 < 0.5\}, \{X_1 \geq 0.5\}$$
$$\{X_2 = A\}, \{X_2 \neq A\}$$
$$\{X_2 = B\}, \{X_2 \neq B\}$$
$$\{X_2 = C\}, \{X_2 \neq C\}$$

Categorical predictors

- If we have a categorical predictor, the splits have the form:

$R_1 = \{X : X_j \in S\}, R_2 = \{X : X_j \notin S\}$ where $S \subset \{1,...,q\}$.

- The software will search over all possible splits for binary or continuous predictors. For multi-class predictors, the software will do its best.

- **Note**: Recursive binary splitting will generally favor predictors with many levels because there are more chances for it to find a good split of the data.

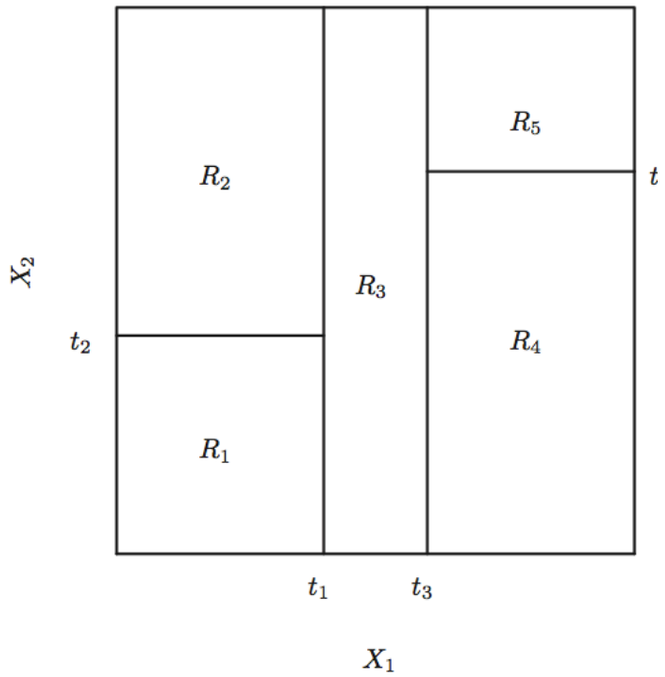
A region-based view

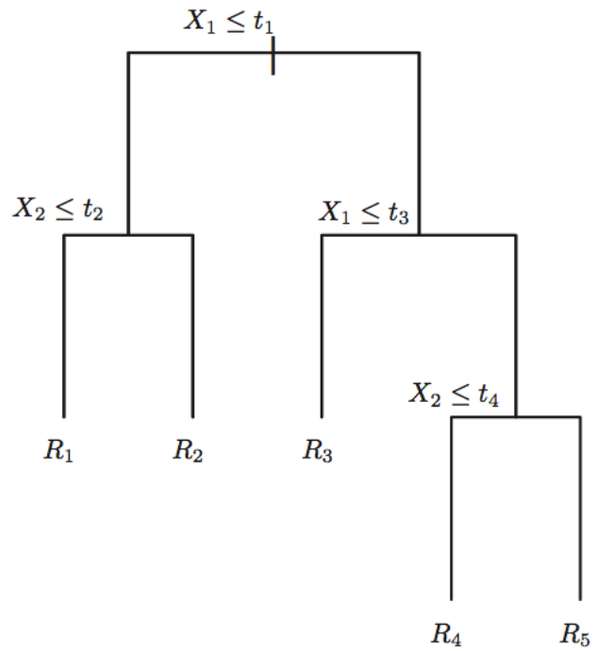
1. **Stratify/Segment**: Partition the predictor space into *J* disjoint regions $R_1, \ldots, R_J$.

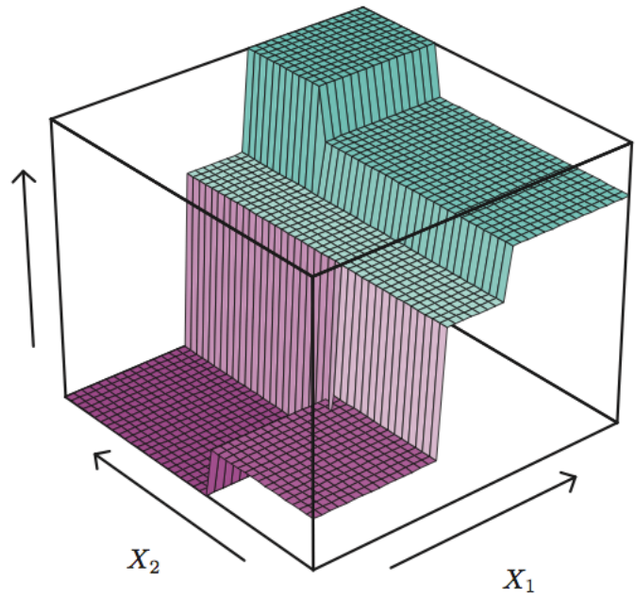
   - How should we construct the regions $R_1, \ldots, R_J$?
   - **How many regions should there be? (How big should *J* be?)**

2. **Prediction**: For observation X in region R_j, output the mean (regression) or mode (classification) of the observed outcomes for the training observations in region R_j

Interactions in a tree







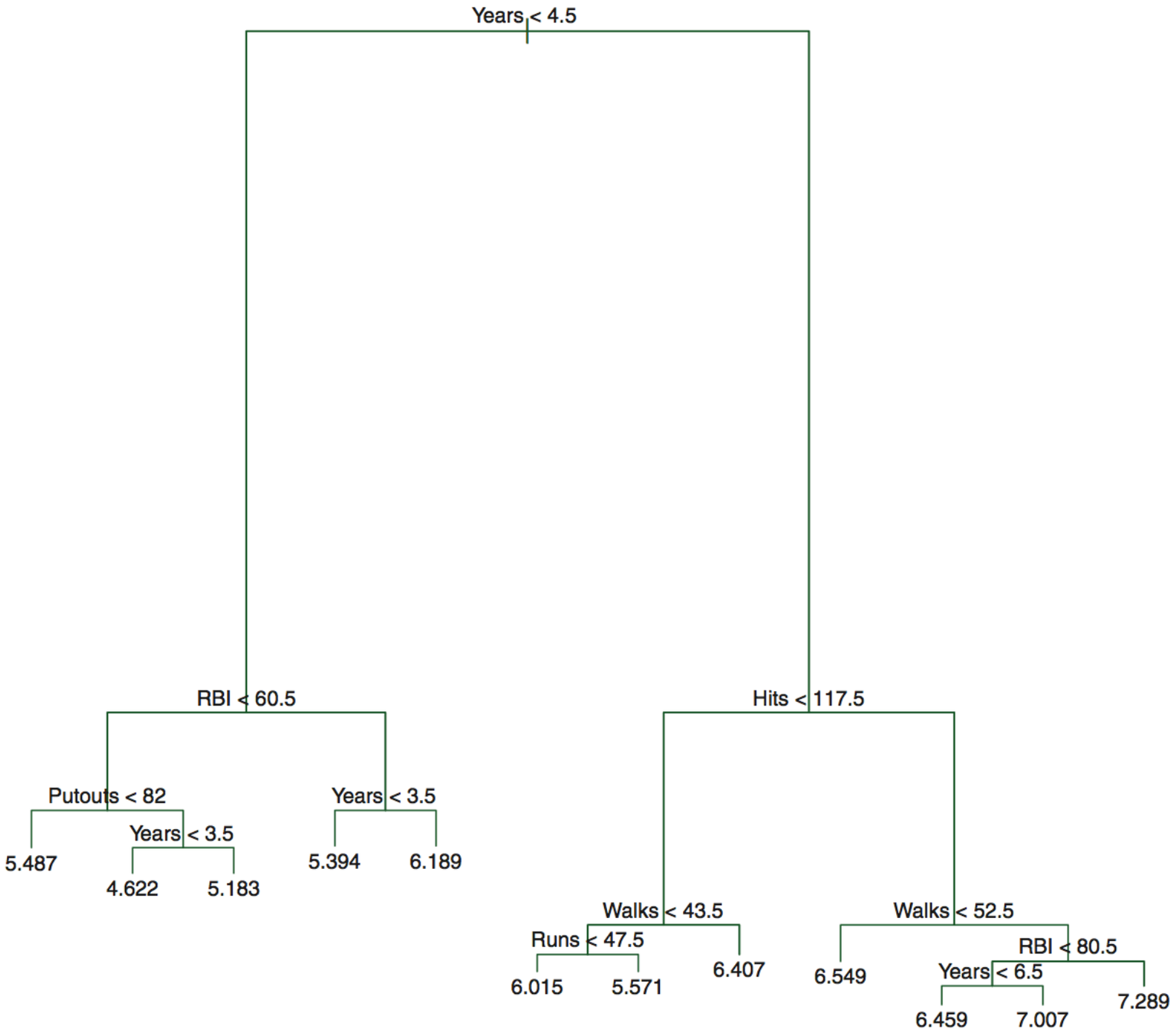
Question: The level of interaction between variables is controlled by the tree depth. For a tree of depth *J*, what is the maximum level of interaction between variables?

Answer: For tree depth *J*, there will be no interaction effects of level greater than *J*. In this example, the tree has depth *J*=3 and the level of interaction is 2.

Tuning the number of regions/tree size

- The **complexity** of the tree model is determined by the number of regions J.
- A tree with large J might overfit to the training data!
- A smaller tree with fewer splits might have lower **variance** (and better interpretability), at the cost of a little **bias**.
- To find a good small tree, a common approach is to:
    1. Grow a large tree, e.g. until no region has > 5 observations.
    2. Consider different subtrees by **pruning** the tree to different sizes.
    3. Use **cross-validation** to select J.

Full Tree for the baseball dataset

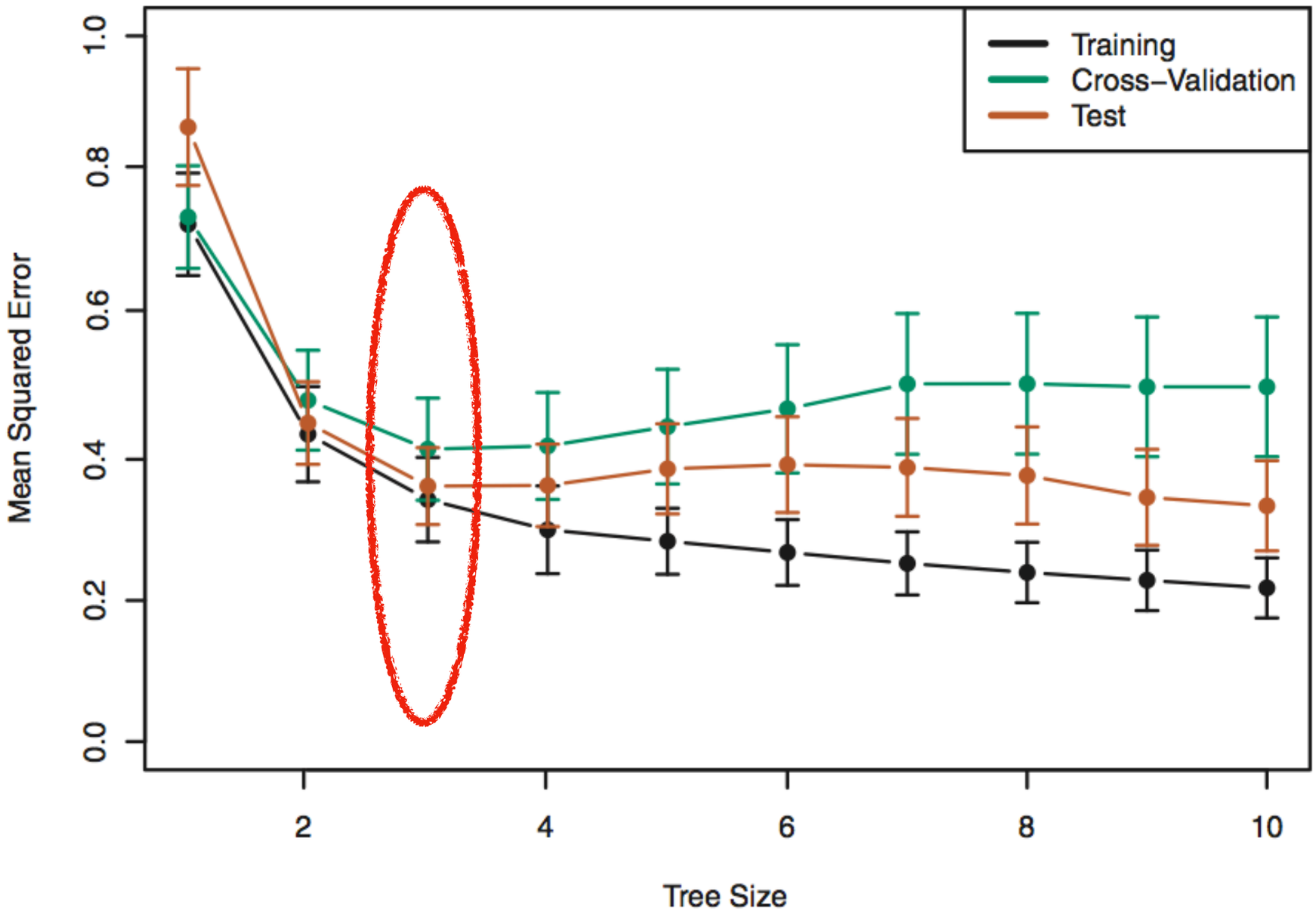


Cross-validation to determine tree size

- Split the data into K folds and fit K trees by holding out each of the folds for the candidate tree sizes.

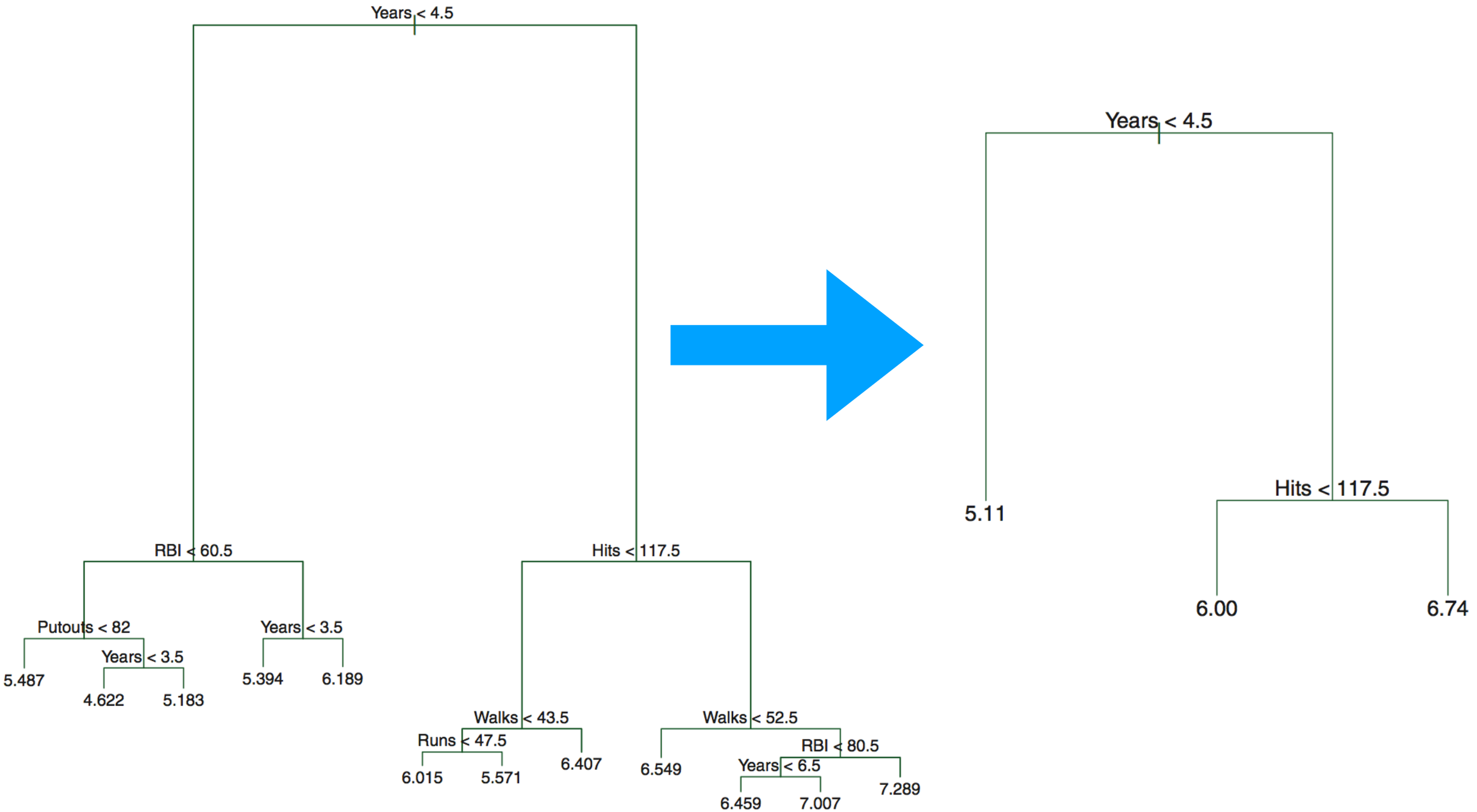
- Calculate the cross-validation error for the candidate tree sizes.

- Select the best tree size that minimizes the CV error (or use the 1-SE rule).



Note: The `rpart` package tunes a tree "complexity" parameter instead of tuning the tree size directly.

Pruned Tree for the baseball dataset

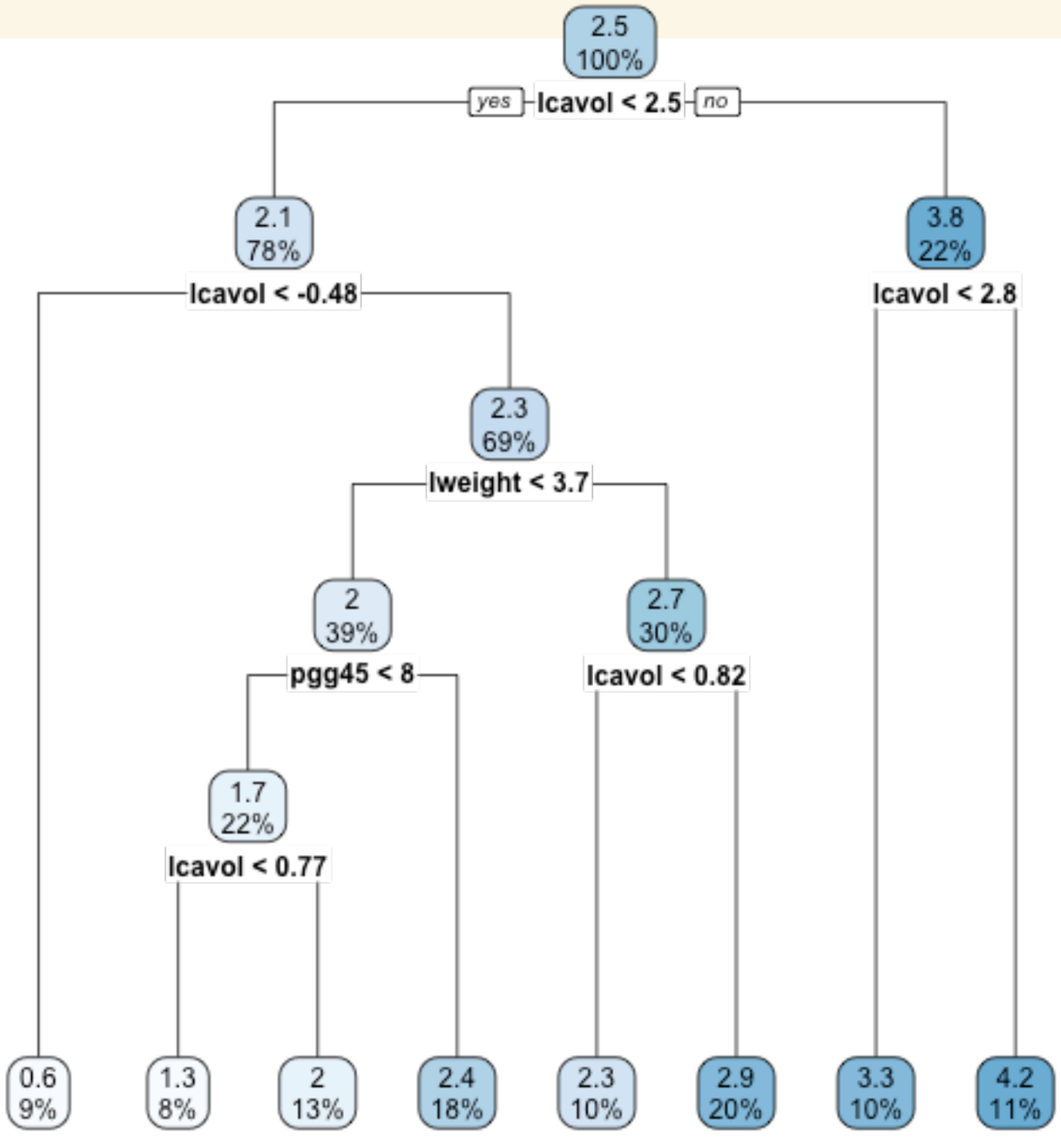


Example: Predicting log(PSA)

- Suppose we are studying the level of prostate-specific antigen (PSA), which is often elevated in men who have prostate cancer. We have $n = 97$ men with prostate cancer and $p = 8$ clinical measurements.

Example: Predicting log(PSA)

```
pros = read.table("http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.data")
psa_rpart <- rpart(lpsa ~ svi + lcavol + lweight + pgg45 + age + gleason, pros)
summary(psa_rpart)
rpart.plot(psa_rpart)
```



Missing Data

- Trees can easily handle data with missing predictor values. Two options are common:

    1. For categorical predictors, create a new category called “missing.”

    2. A more general approach is to create **surrogate splits**:

        - When considering a predictor for a split, use only observations for which that predictor is not missing.

        - After choosing the best **primary split**, then form a list of surrogate predictors and split points:

            - The first surrogate predictor is the best predictor and split point that best mimics the primary split.

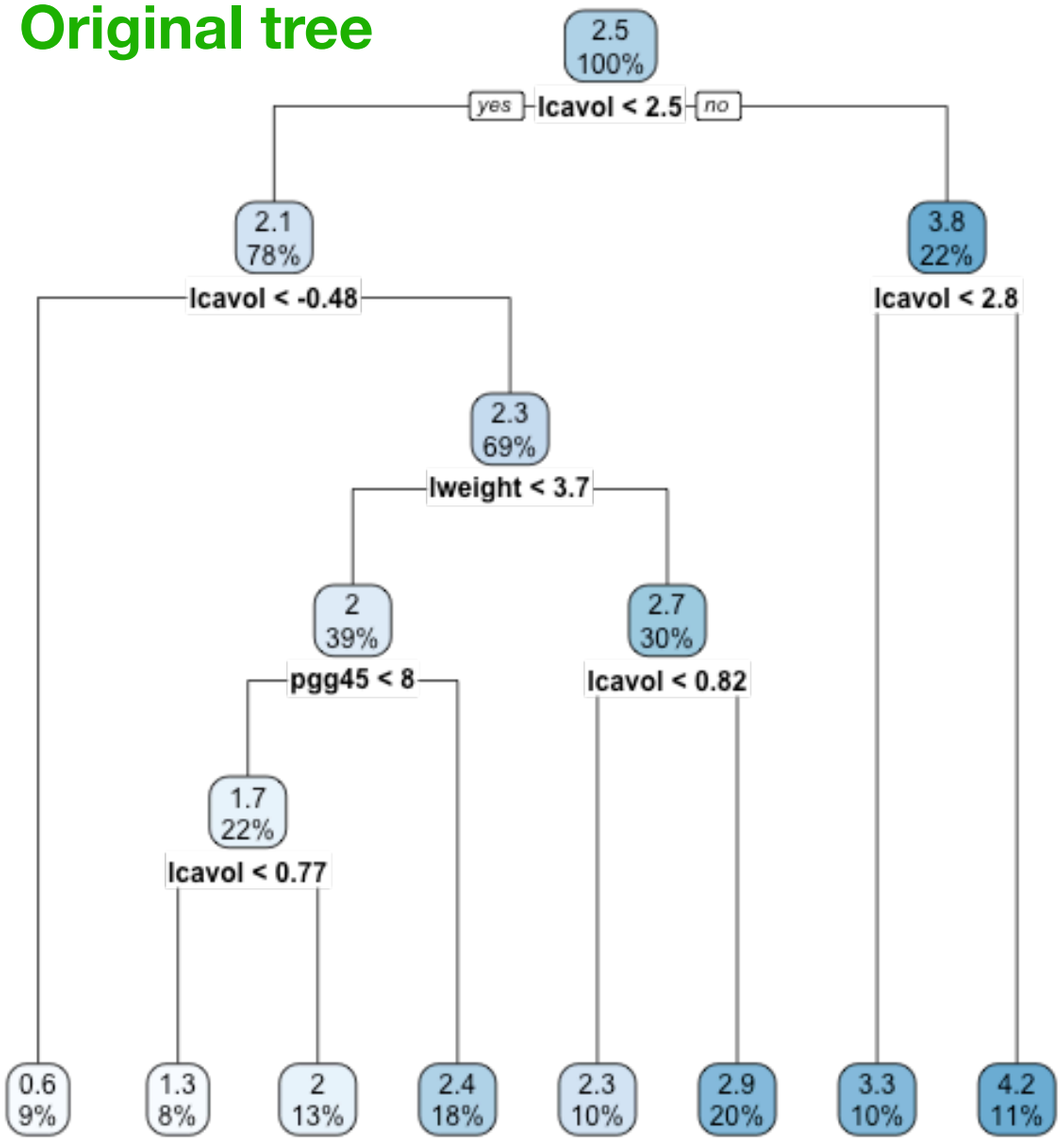
            - The second surrogate predictor is the predictor and split point that is second best at mimicking the primary split.

            - …

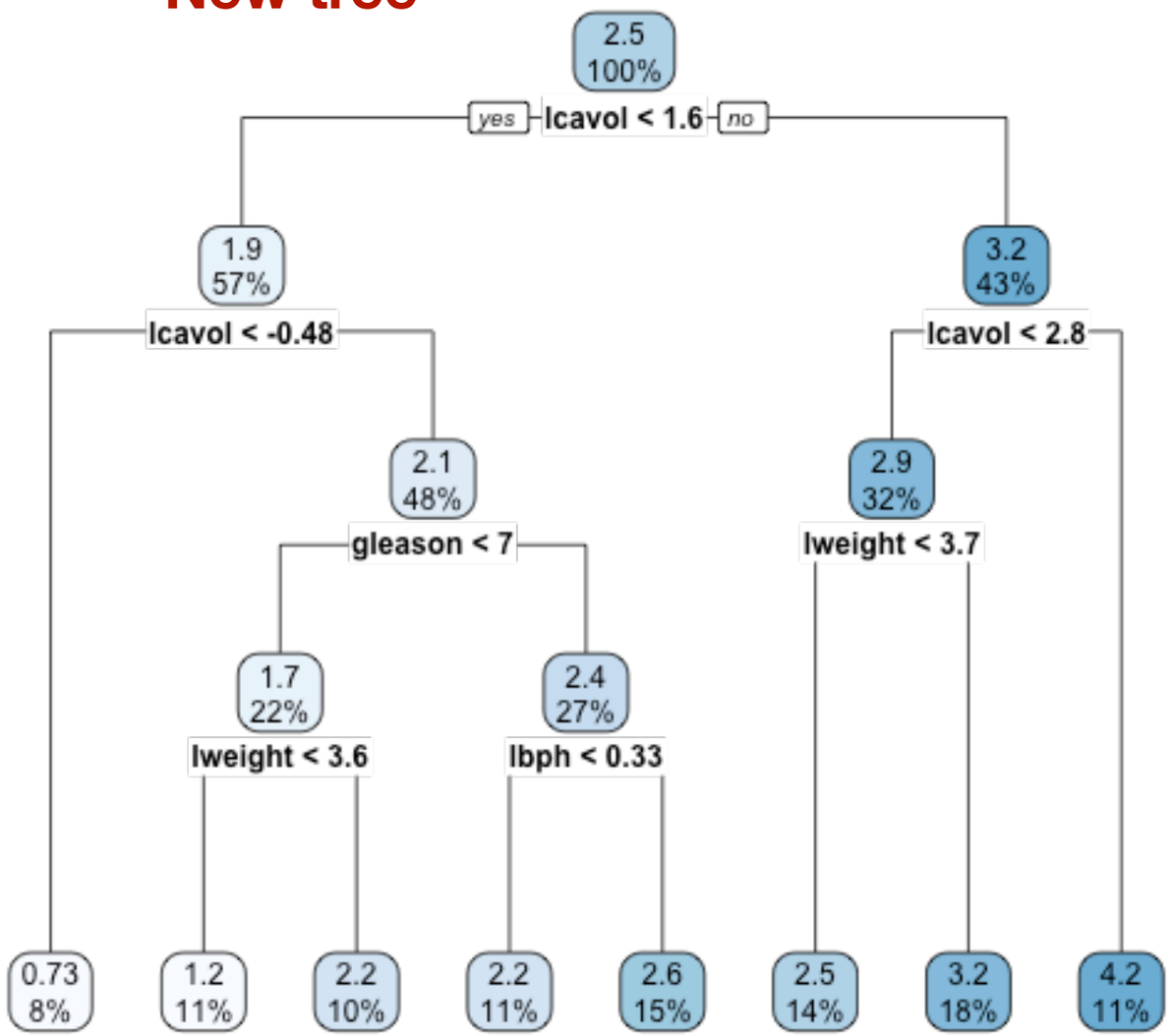
Example: Predicting log(PSA)

- Let's randomly remove some predictor values. For the variables age, gleason, and lcavol, I randomly selected 5 subjects and removed their values.

Original tree



New tree



Example: Predicting log(PSA)

- Let's look at the surrogate splits for the first node:

```
Node number 1: 97 observations,    complexity param=0.324024
  mean=2.478387, MSE=1.318739
  left son=2 (55 obs) right son=3 (42 obs)
  Primary splits:
      lcavol  < 1.551753    to the left,  improve=0.3232121, (5 missing)
      svi     < 0.5         to the left,  improve=0.3206031, (0 missing)
      lcp     < 0.1359669   to the left,  improve=0.2770979, (0 missing)
      pgg45   < 8           to the left,  improve=0.2519755, (0 missing)
      lweight < 3.672483    to the left,  improve=0.2254510, (0 missing)
  Surrogate splits:
      lcp     < 0.2616241   to the left,  agree=0.870, adj=0.707, (5 split)
      svi     < 0.5         to the left,  agree=0.761, adj=0.463, (0 split)
      pgg45   < 27.5        to the left,  agree=0.750, adj=0.439, (0 split)
      lweight < 3.446796    to the left,  agree=0.620, adj=0.146, (0 split)
      gleason < 6.5         to the left,  agree=0.609, adj=0.122, (0 split)
```

This is the primary split we chose for the first node.

This is the best split for mimicking the selected primary split.

Summary: Trees

- Pros:
    - Very easy to interpret and explain to others
    - Can easily handle both categorical and continuous predictors, as well as missing data
    - Invariant to monotonic transformations of the predictors
    - Fast to fit
- Cons:
    - Doesn’t model linear decision boundaries very well.
    - Decision trees have **high variance**. If we resample the training data, we may get a very different tree.

Today's Lecture

- Decision Trees
- Bagging
- Random Forests
- Variable Importance

Bagging

- Bootstrap aggregation, or **bagging**, is a general-purpose procedure for reducing the variance of a statistical learning method by averaging.

- We know that **averaging reduces the variance**:

  - Specifically, if we take the average of *n* independent observations $Z_1, \cdots, Z_n$, each with variance σ^2, then

$$Var\left(\frac{1}{n}\sum_{i=1}^{n} Z_i\right) = \sigma^2/n.$$

- Idea: What if we can average models estimated on *B* datasets?

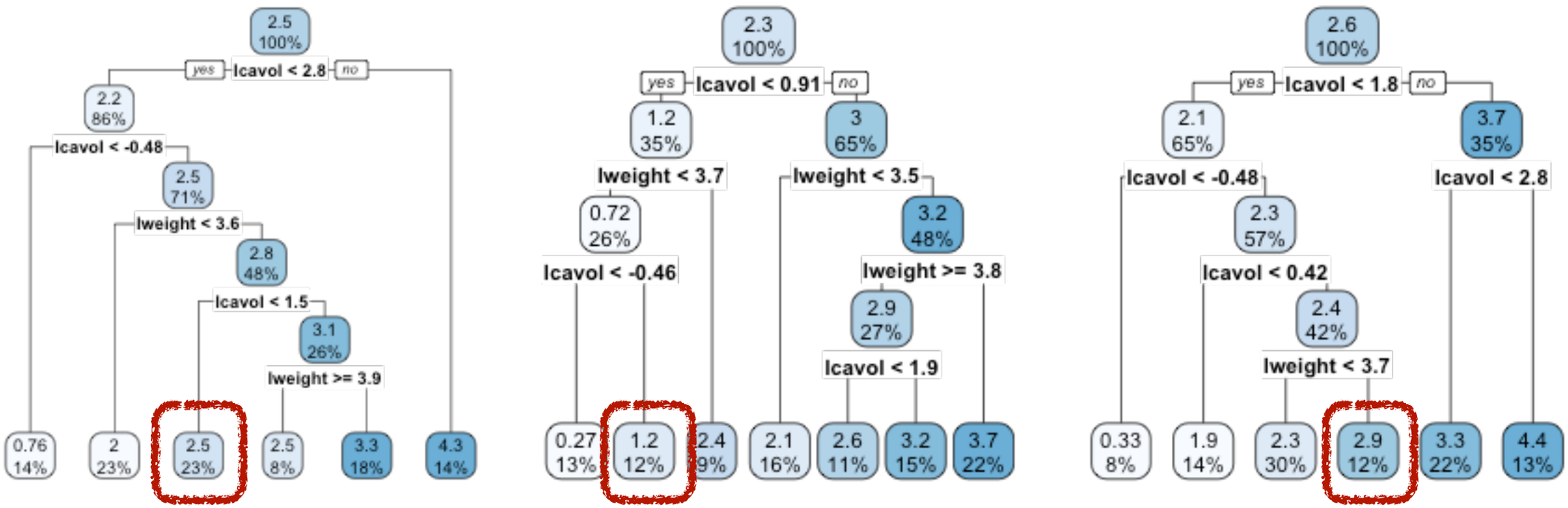
$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$$

Bagging

- How do we make *B* datasets when we only have one dataset? We create bootstrap samples:
  - For $b = 1, \cdots, B$: Randomly draw *n* observations with replacement.
- Bagging averages the models trained on *B* bootstrap samples:

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

Averaging the output from multiple trees



For a new observation with lcavol = 0.5, lweight = 3.8:

Tree 1 predicts 2.5
Tree 2 predicts 1.2 → We average the predictions and output 2.2
Tree 3 predicts 2.9

- For classification trees, output the average as a probability or do majority voting.

Bagging

- The number of bootstrap samples *B* is a hyperparameter…

- But the procedure is not that sensitive to the value of *B:*

  - As *B* increases, the bagged model will converge.

  - In practice, *B* = 100 to *B* = 1000 work pretty well

An issue with Bagging

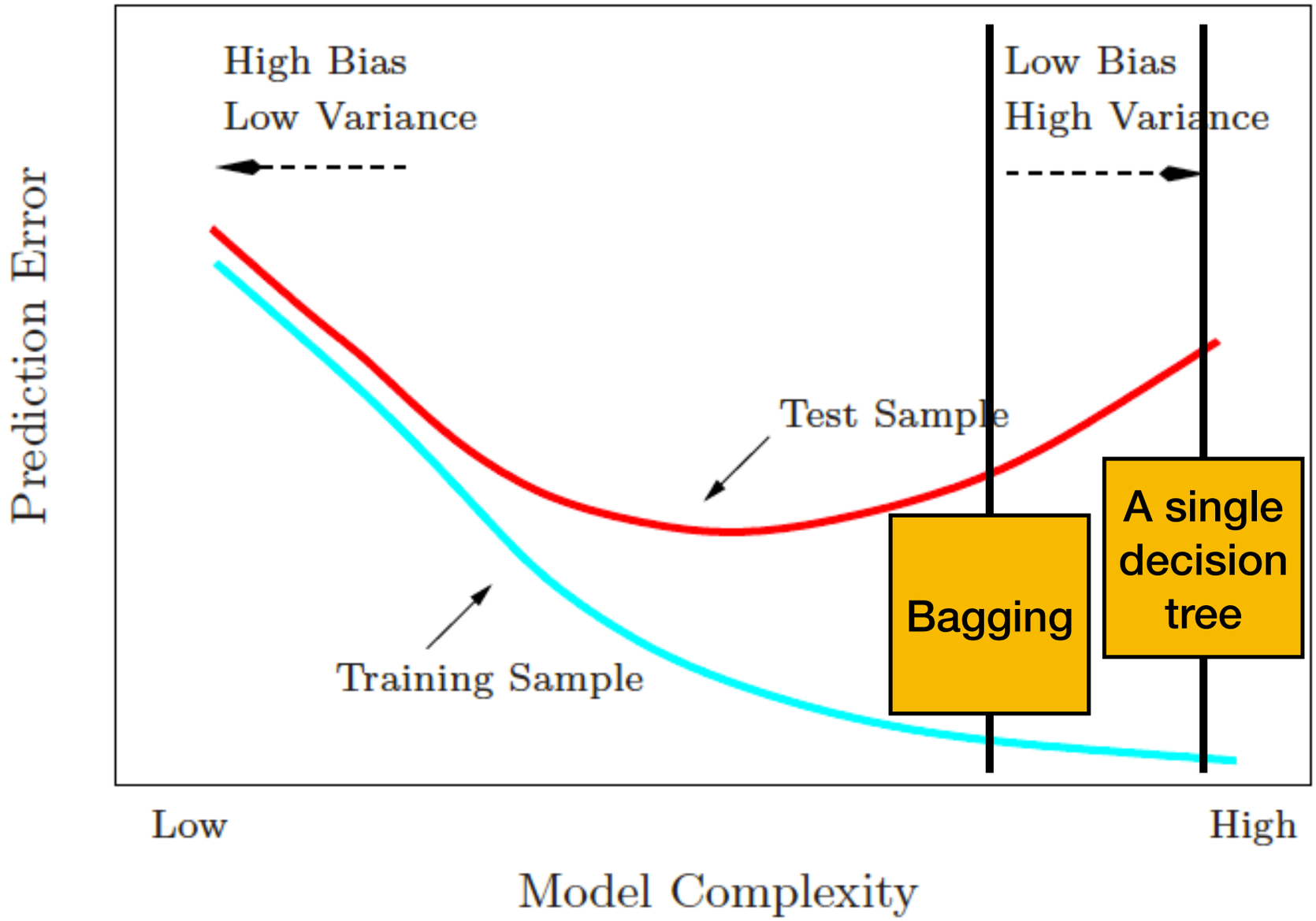
- Suppose that there is one very strong predictor in the data set, along with a number of moderately strong ones

- Then in the collection of bagged trees, most or all of the trees will **use this strong predictor in the top split**, and they all look somewhat similar

- This means that **predictions from the bagged trees can be highly correlated**.

- In this setting, bagging will only reduce the model variance slightly

Bias-variance trade-off



Today's Lecture

- Decision Trees
- Bagging
- Random Forests
- Variable Importance

Random Forests

- Goal: Make the trees less similar.

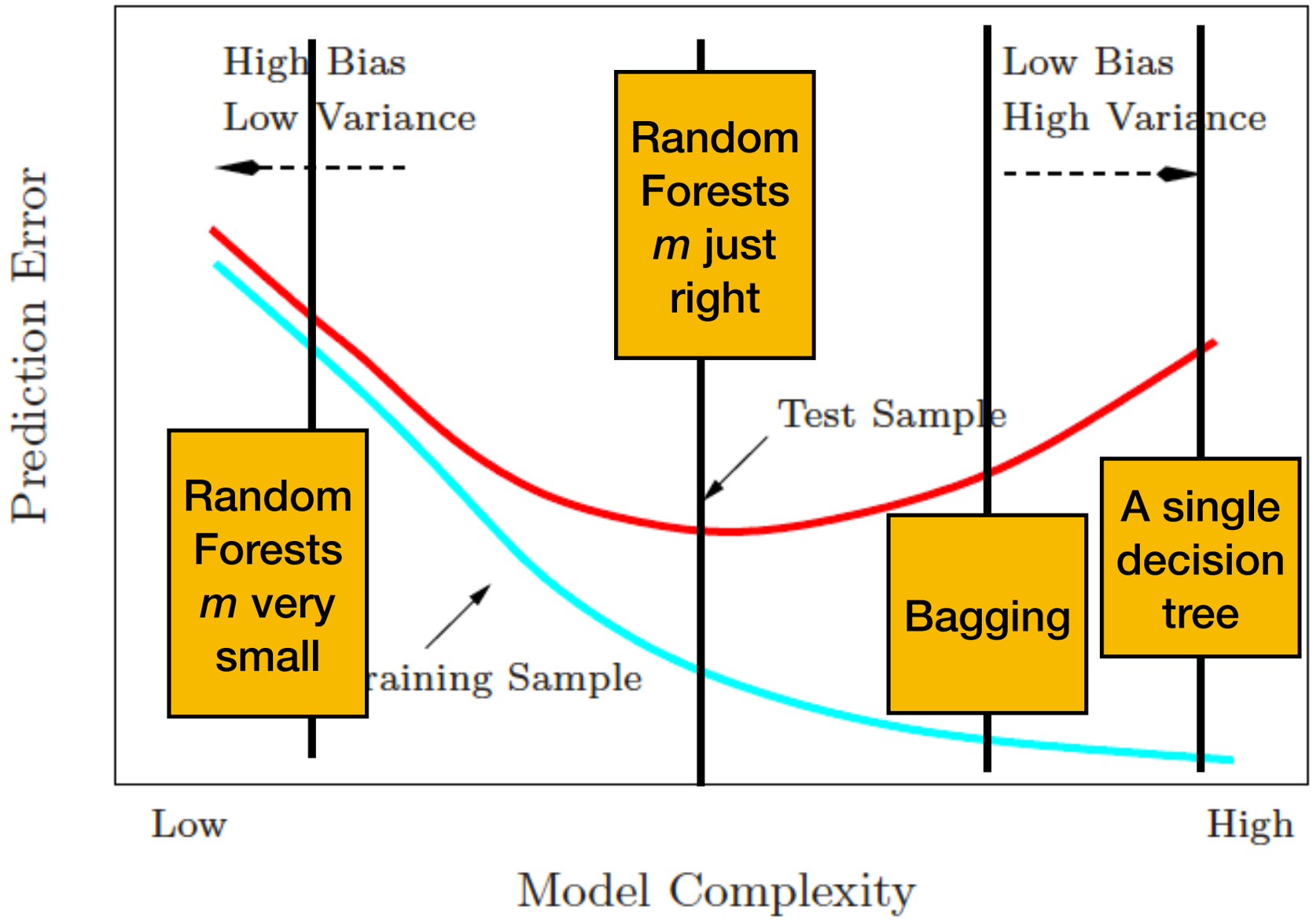
- Idea: When fitting a tree for each bootstrap sample, randomly pick m variables to consider at each split.

- By randomly selecting variables to use in each tree, we "**decorrelate**" the trees and achieve a larger reduction in variance.

Random Forests

- How many predictors should I pick at random?
  - As m increases, the trees become more similar.
  - If m is very small, the randomly selected variables will have very little predictive power and the trees will have higher bias.
- Typical choice is $m = \sqrt{p}$.

Bias-variance trade-off



Random Forests

Journal of Machine Learning Research 15 (2014)

Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Abstract: We evaluate **179 classifiers** arising from **17 families**… We use 121 data sets, which represent the whole UCI data base and other own real problems… **The classifiers most likely to be the bests are the random forest (RF) versions**, the best of which… achieves 94.1% of the maximum accuracy over all the datasets…

Evaluating the test error of a random forest

- Every time we create a bootstrap sample to fit a tree, there will be leftover observations. These are referred to as **out-of-bag (OOB) observations**.

- We can use these to estimate the generalization error of the random forest:

  - For the *i*-th observation, predict its response using all the trees for which that observation was OOB.

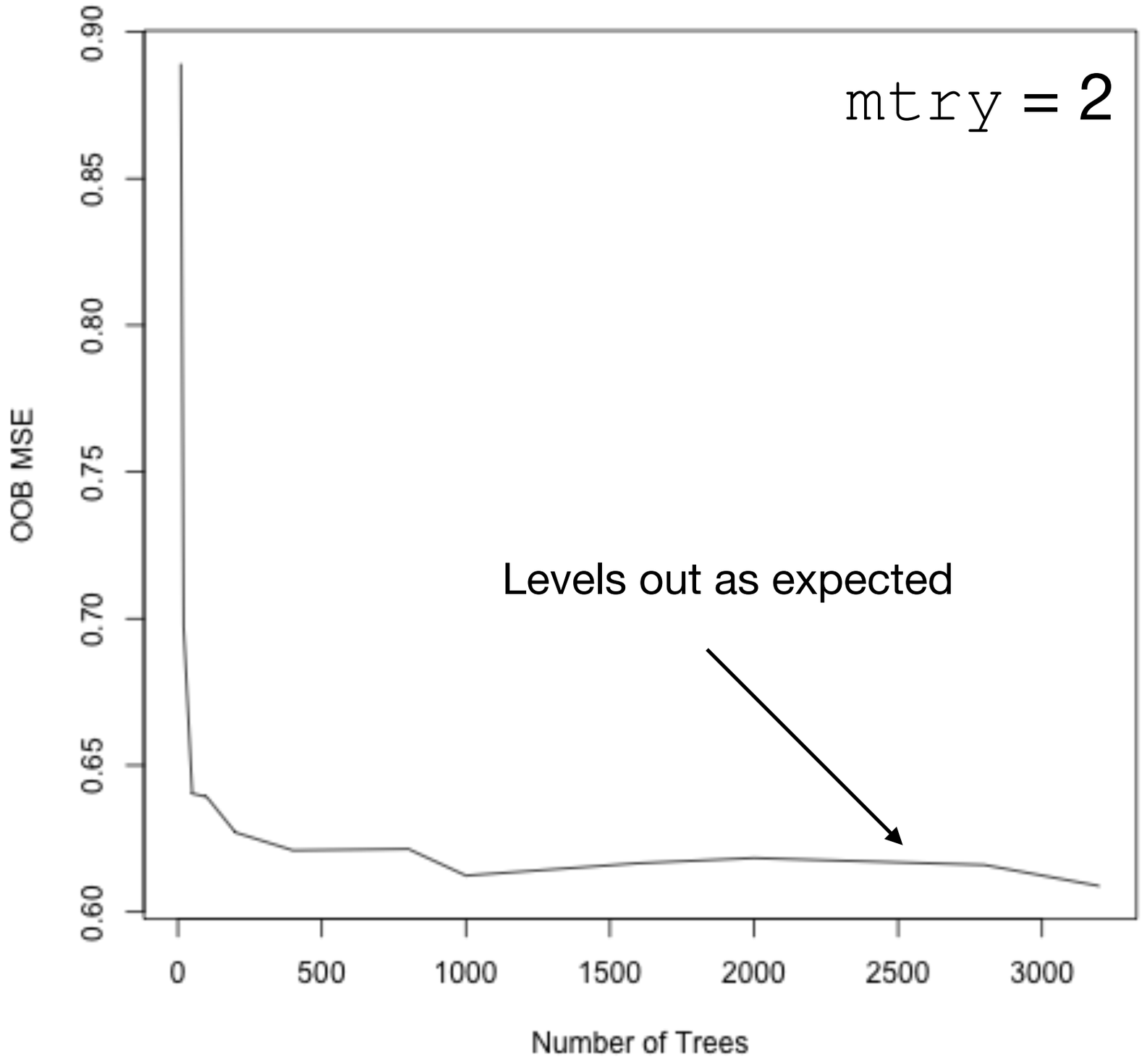
  - We estimate the generalization error of the RF using the average error of the OOB predictions.

Example: Random Forest for Prostate data

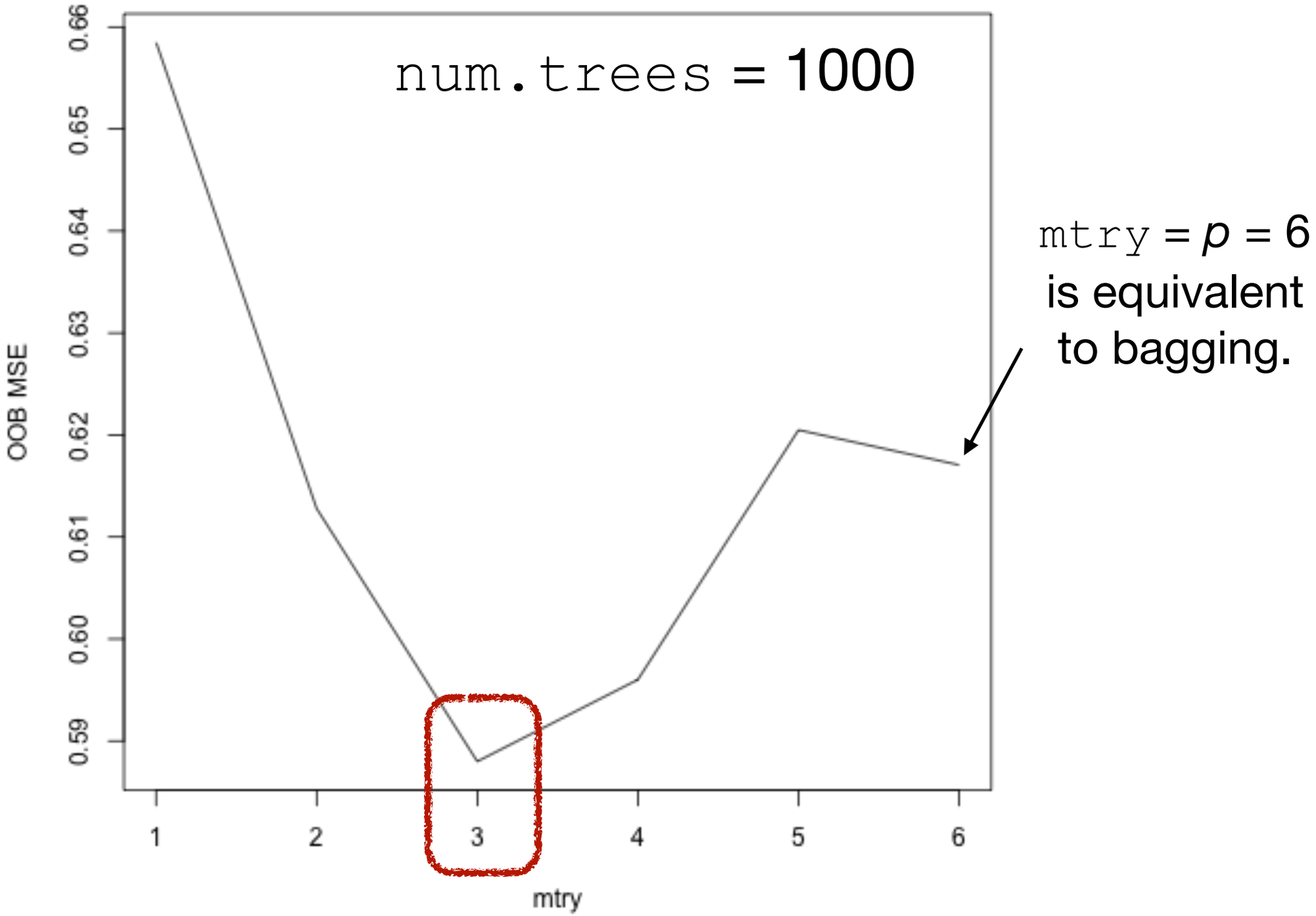
```
> psa_rf <- ranger(lpsa ~ svi + lcavol + lweight + pgg45 + age + gleason, pros
> psa_rf
Ranger result

Call:
 ranger(lpsa ~ svi + lcavol + lweight + pgg45 + age + gleason,

Type:                             Regression
Number of trees:                  500
Sample size:                      97
Number of independent variables:  6
Mtry:                             2
Target node size:                 5
Variable importance mode:         impurity
Splitrule:                        variance
OOB prediction error (MSE):       0.6197062
R squared (OOB):                  0.5349212
```

`mtry` is the number of covariates randomly sampled at each node

Example: Hyperparameters in the Random Forest



Example: Hyperparameters in the Random Forest



Random forests for high-dimensional data

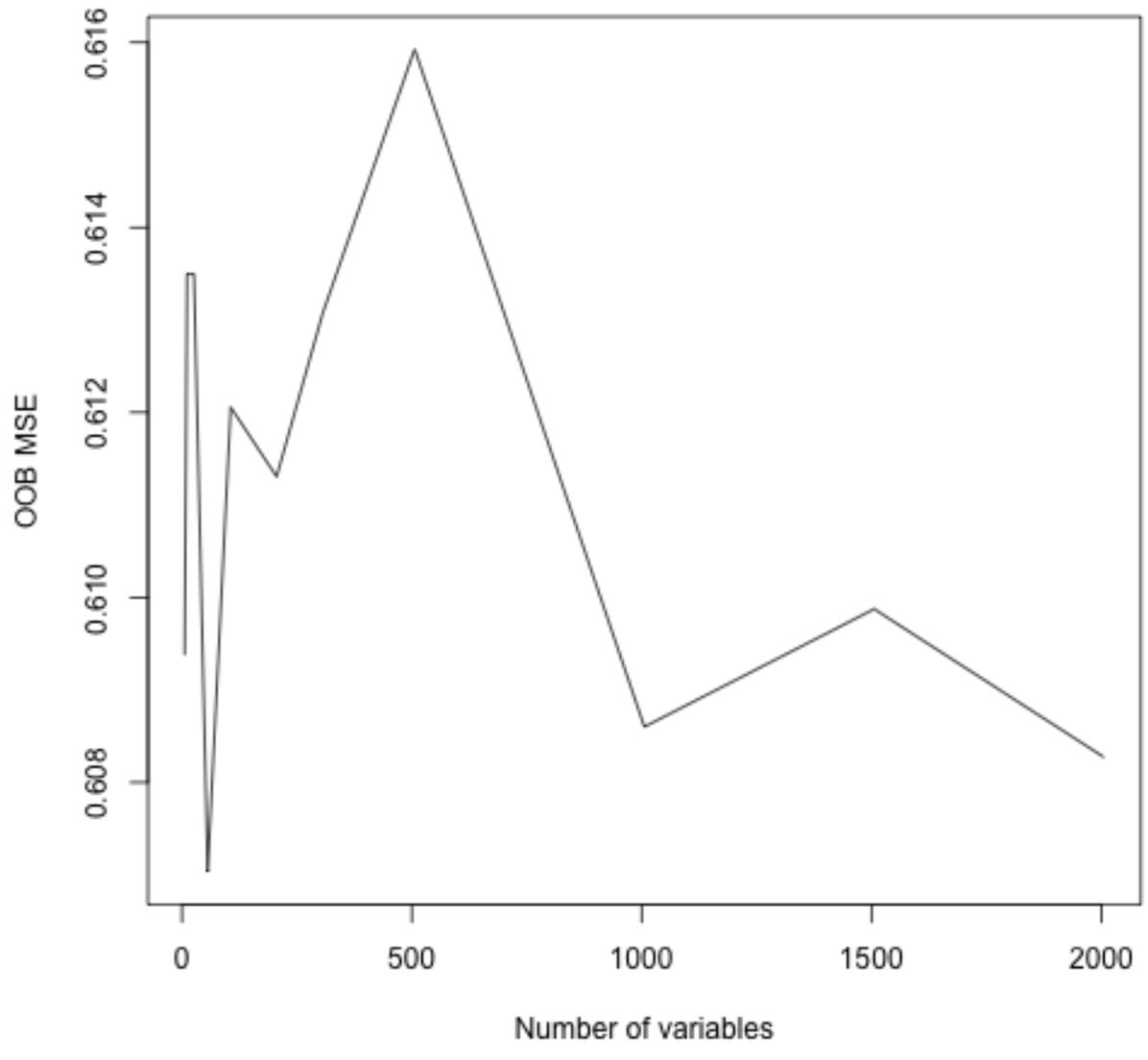
- **Q**: How do you think random forests will perform for high-dimensional data?

- **A**: Terribly. A random forest model will get confused by spurious correlations between the variables and the outcome.

- **B**: Random forests are naturally doing some sort of variable selection and this will help regularize the model. They'll probably do fine.

Example: RF for Prostate data with more variables

- Let's add many additional variables to the prostate data and see how this affects the OOB error.



Looks like RF is actually doing quite well in this high-dimensional example.

Today's Lecture

- Decision Trees
- Bagging
- Random Forests
- Variable Importance

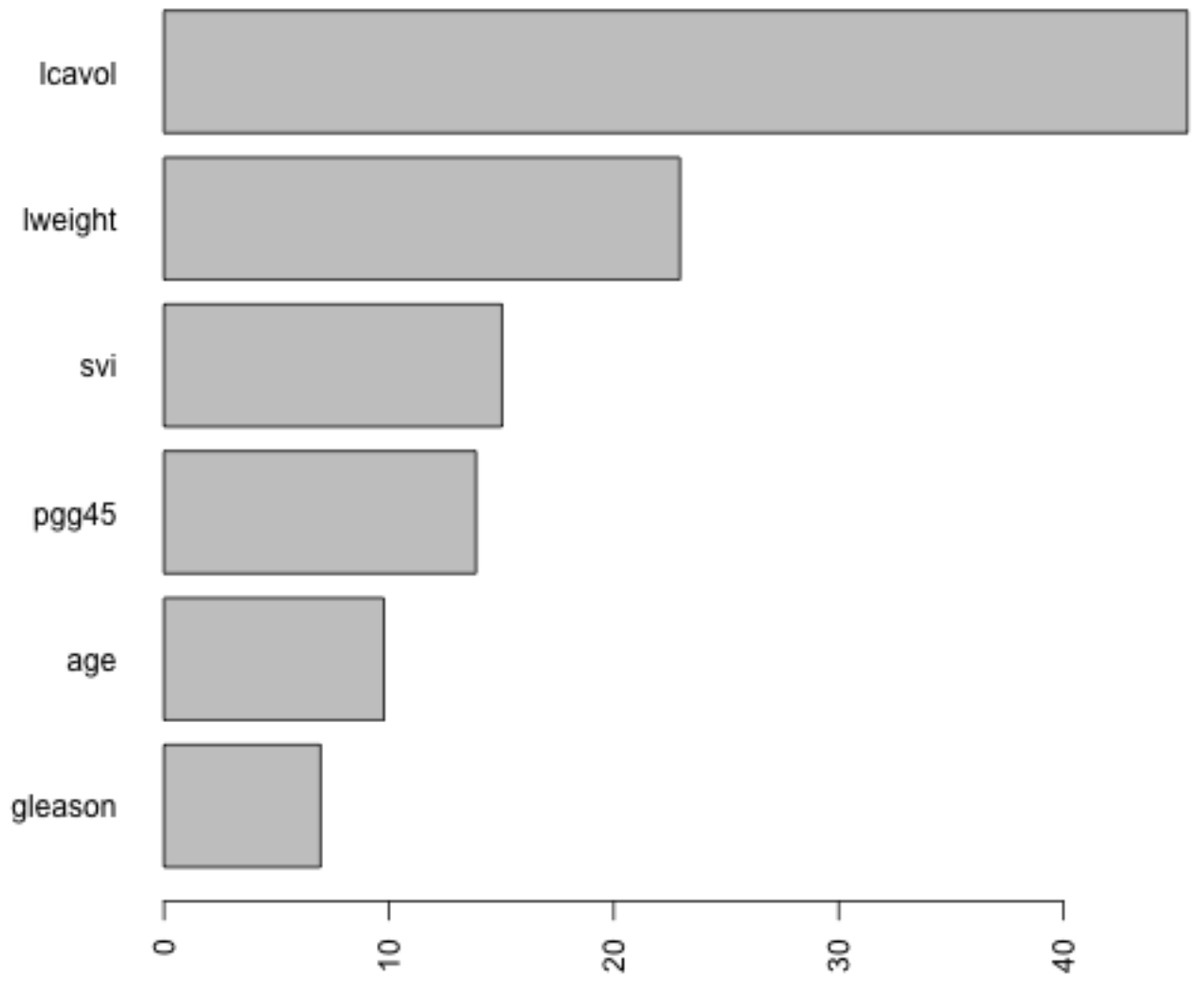
Interpreting random forests

- Interpreting random forests is tricky because we are averaging across trees with different structures.

- A very popular approach is to **estimate the importance of each variable**.

Variable importance

- A typical variable importance plot:



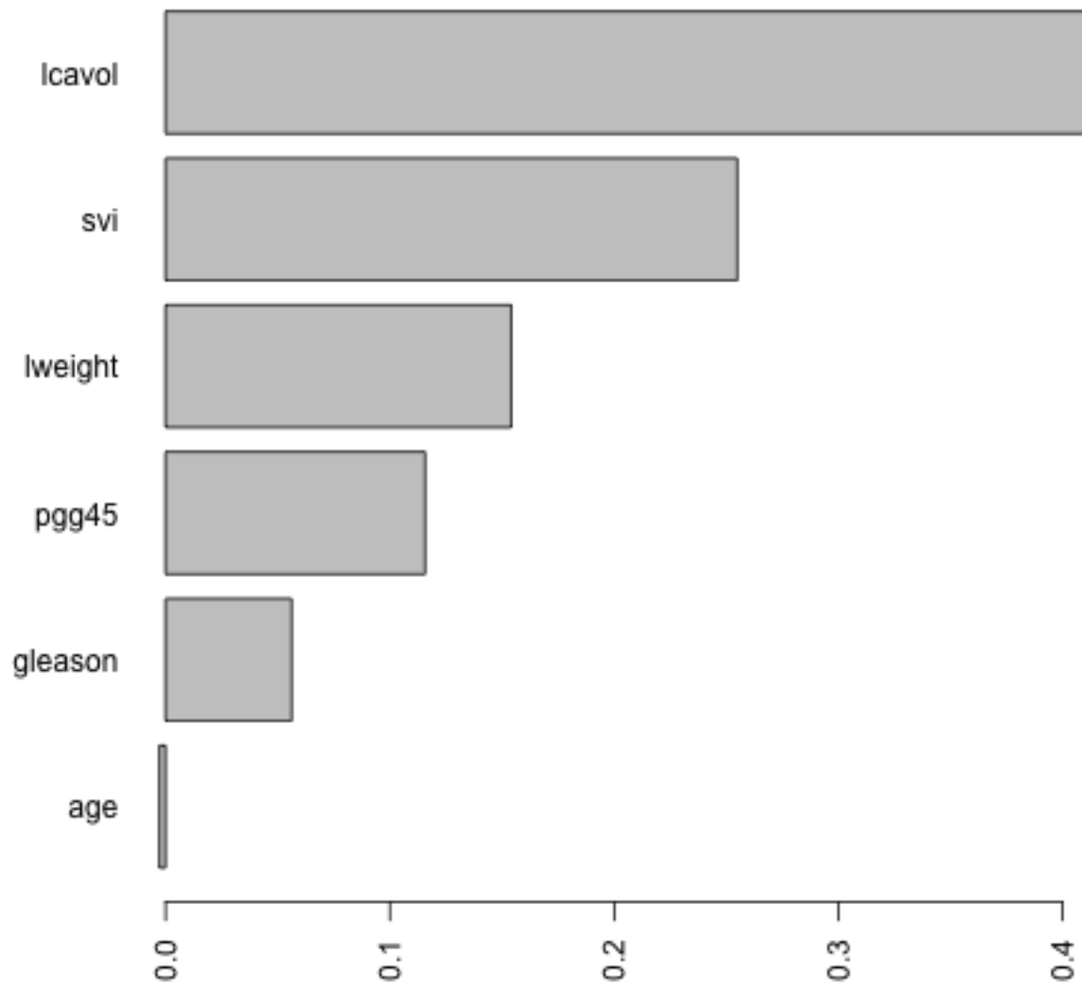
Permutation-based variable importance

- Fit the random forest.
- Compute the OOB error.
- For j = 1,…,p:
    - Permute the values of the j-th variable. We compute the OOB error for the permuted dataset.
- Compare the OOB error for the original dataset versus the permuted dataset.

Example in R

```
> psa_rf <- ranger(lpsa ~ svi + lcavol + lweight + pgg45 + age + gleason, pros, importance = "permutation")
> importance(psa_rf)
         svi       lcavol      lweight        pgg45          age      gleason
 0.254875333  0.456267123  0.154121162  0.115708769 -0.002733986  0.056206880
```

- The importance of the *j*-th variable is the increase in OOB mean squared error when you permute the *j*-th variable.



A zoo of variable importance measures

- There are many variable importance measures specific to random forests.

- However, many variable importance measures have been developed since then, some of which are not model-specific.

- Generally speaking, there are two categories:

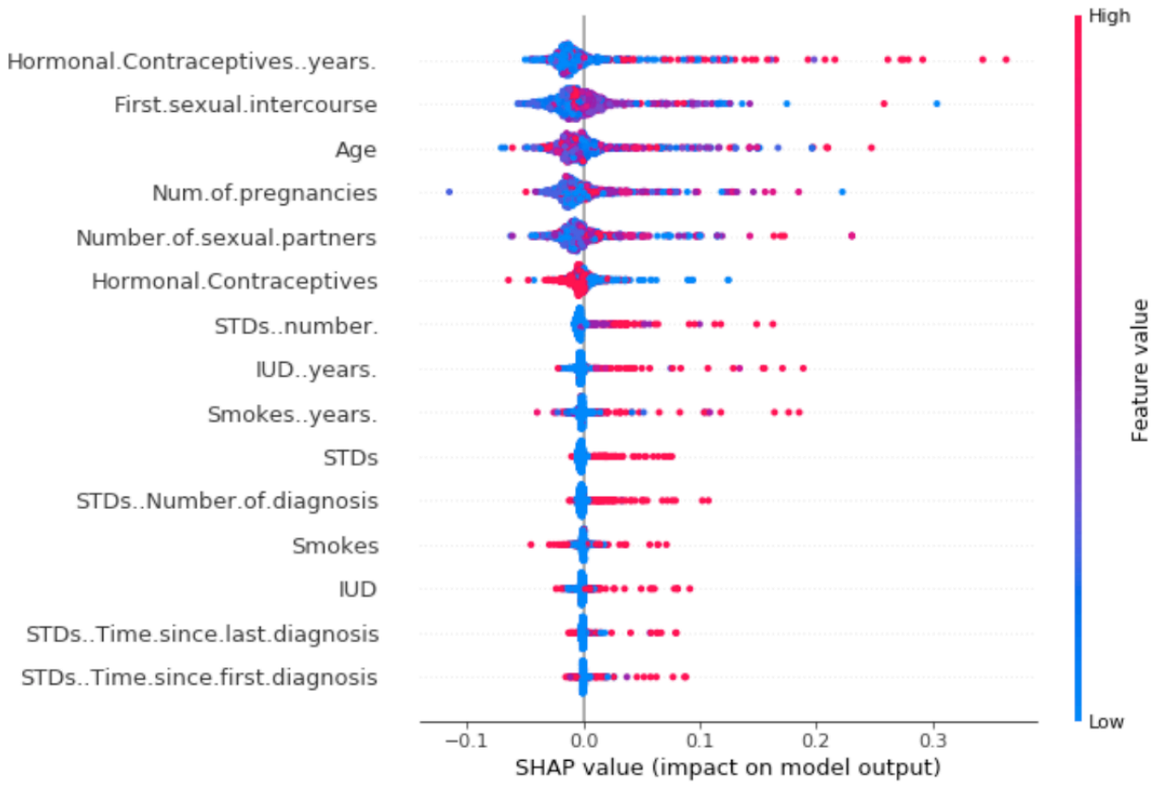
  - Explaining the importance of a variable for a fitted model

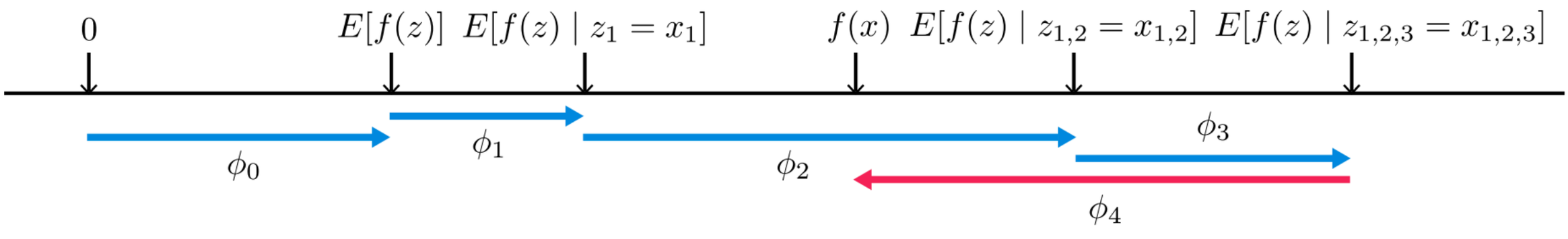
  - Explaining the importance of a variable in the true model

SHAP values

- Shapley Additive exPlanation (SHAP) values summarize how a given prediction from a given model change when it gets to observe a feature versus when it doesn't.

- It averages over all possible feature subsets to determine how much the model prediction changes on average.





Today's Lecture

- Decision Trees
- Bagging
- Random Forests
- Variable Importance