Supervised Learning: Trees, Bagging, Random Forests, Boosting

Jean Feng & Ali Shojaie

Aug 19-21, 2024
Summer Institute in Statistics for Big Data
University of Washington

Outline

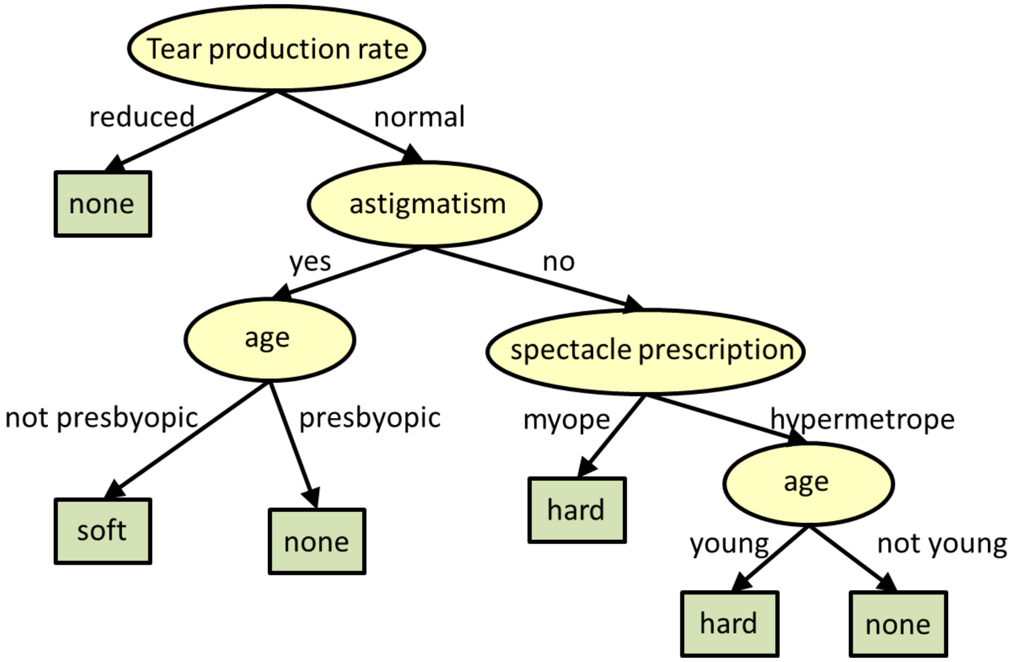
- Decision Trees
- Ensembling
    - Bagging
    - Random Forests
    - Gradient boosted trees
- Variable Importance

Outline

- **Decision Trees**
- Ensembling
    - Bagging
    - Random Forests
    - Gradient boosted trees
- Variable Importance

Decision Trees

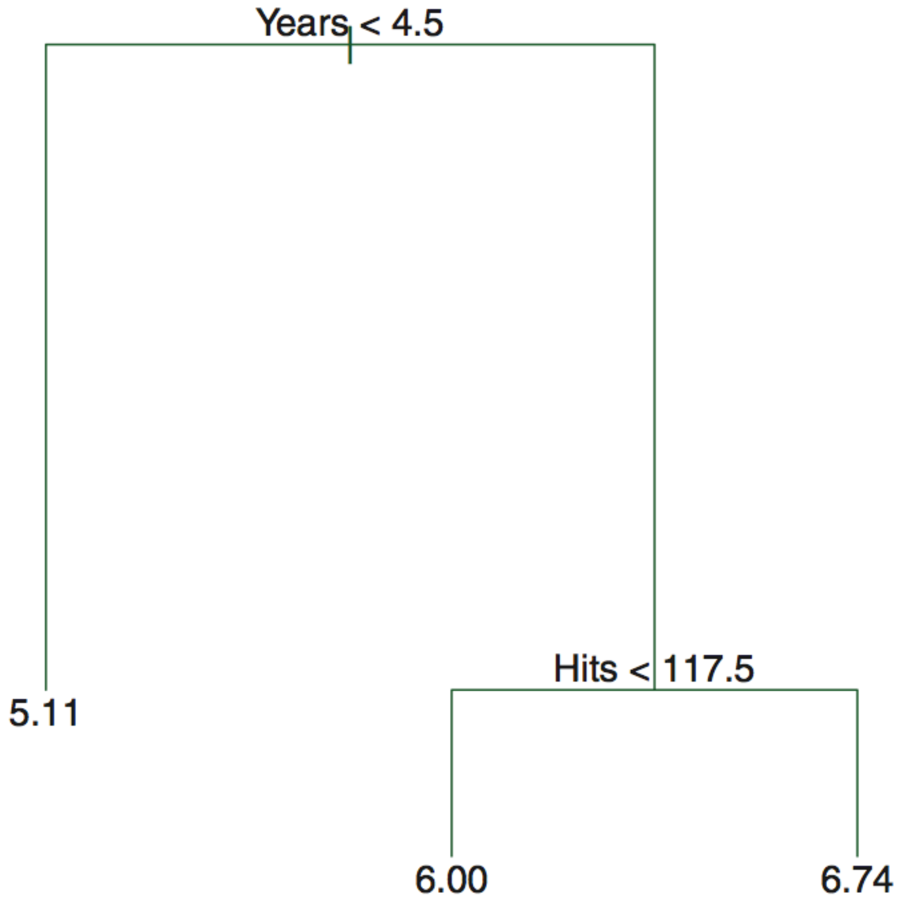
- Decision trees — also known as **Classification and Regression Trees (CART)** — can be used in both regression and classification tasks.
    - They have also been extended for survival outcomes.
- Trees are able to model **non-linearities** and **interactions** between variables very naturally.
- Trees are probably an oversimplification of the true data, but they are very intuitive and interpretable.



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)
    - number of hits made in the previous year (Hits)



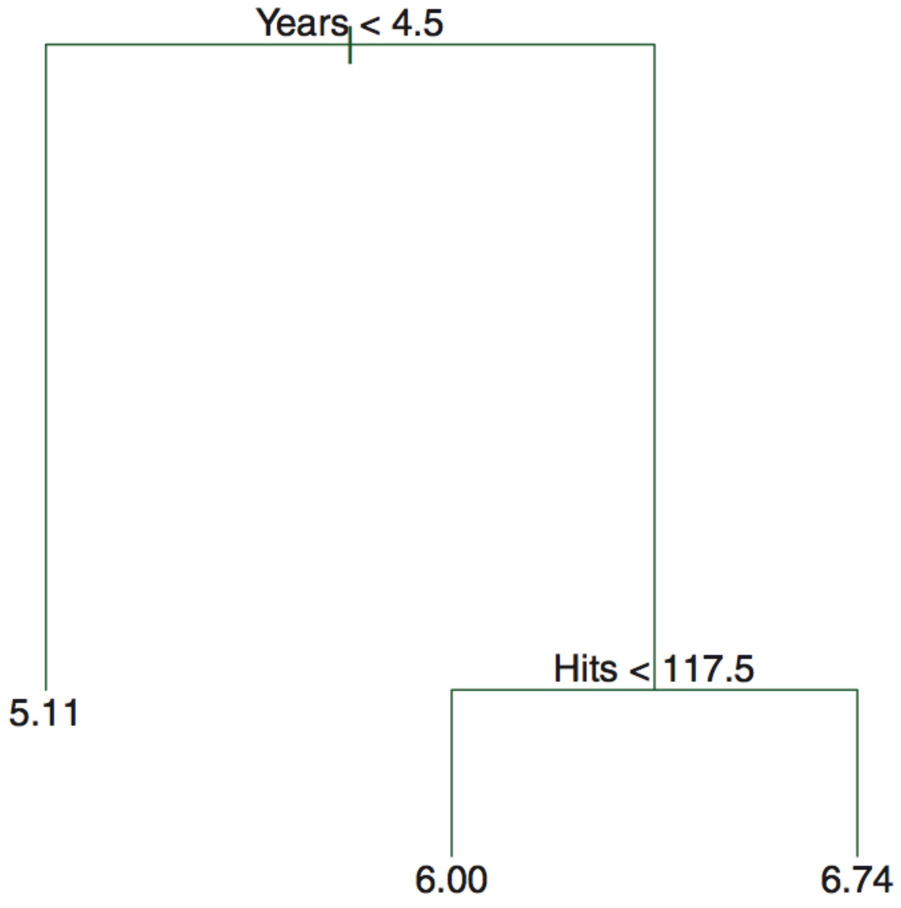
Toy Example: Predicting salaries of baseball players

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

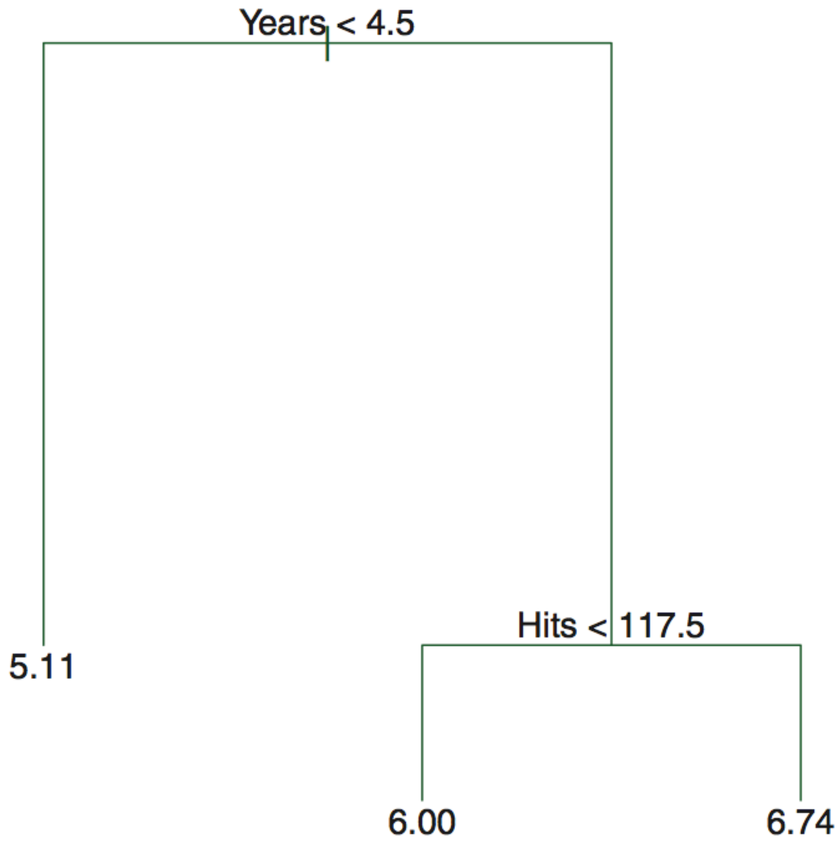
- Among players with > 4.5 Years of experience...

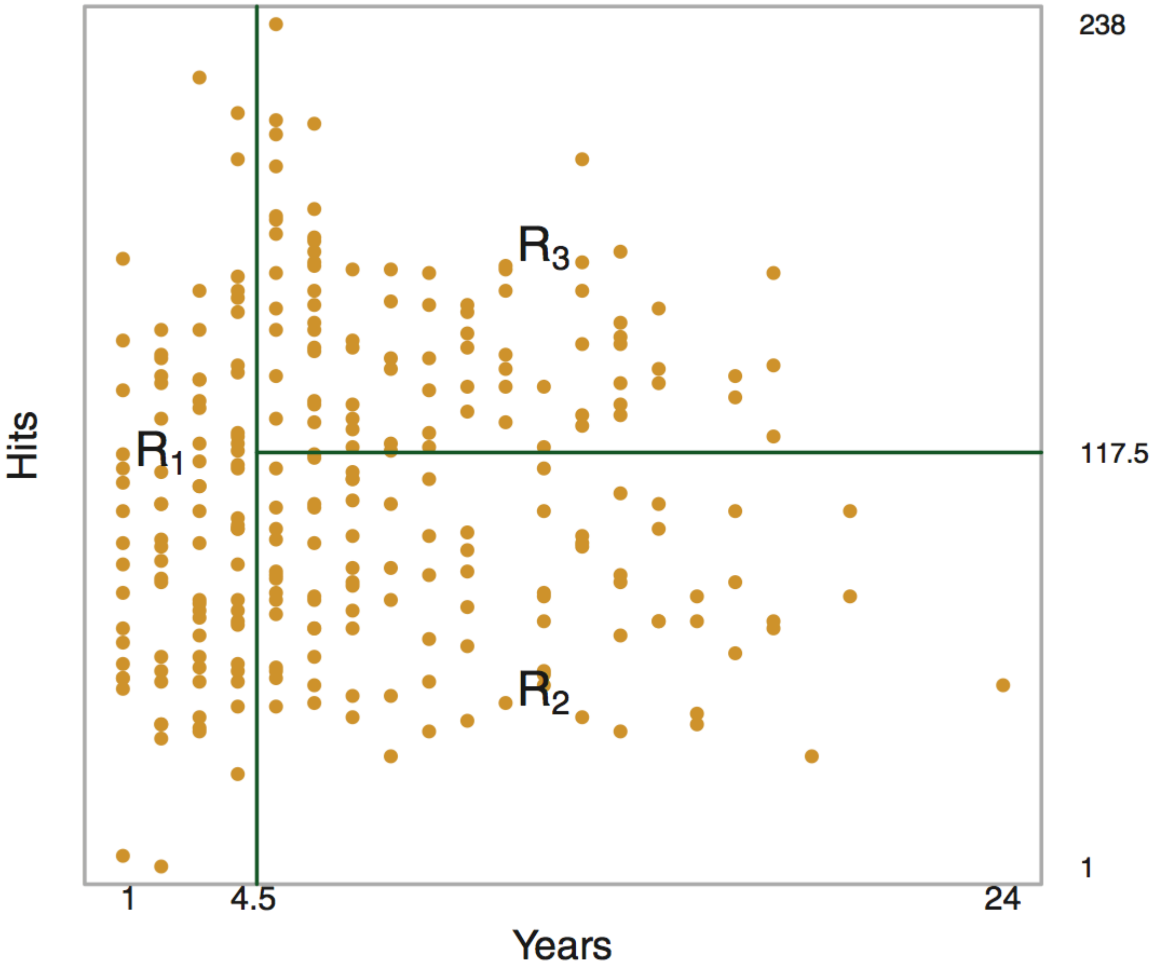
  - For those with < 117.5 Hits, the predicted salary is \$6.00 million.

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



A region-based view





$$R_1 = \{X \mid Years < 4.5\}$$

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

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

Fitting a decision tree

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

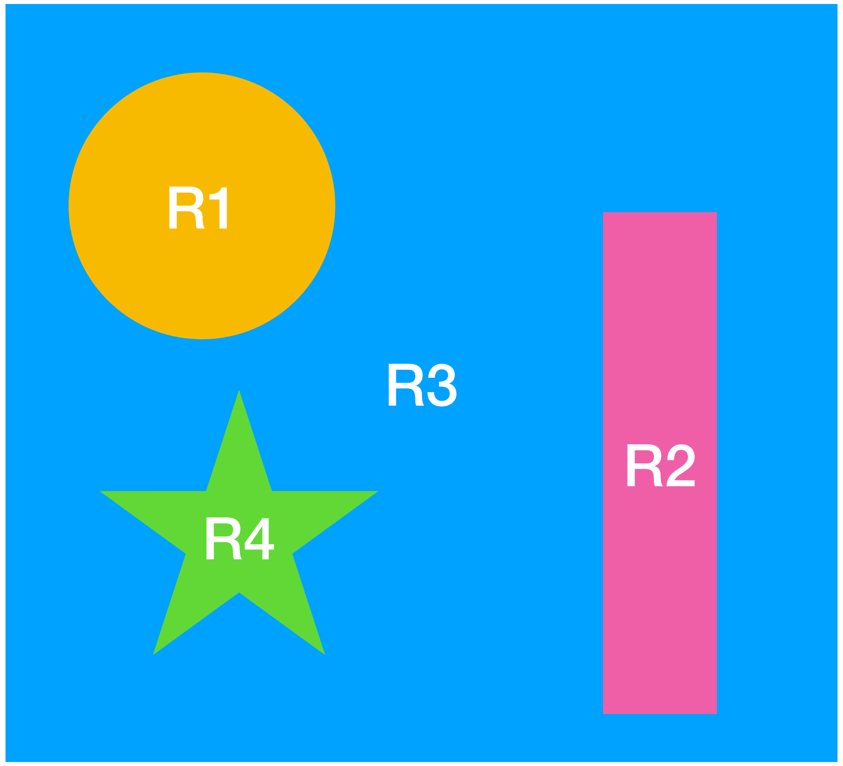
- For now assume *J* is known.

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

- We introduce two simplifications:

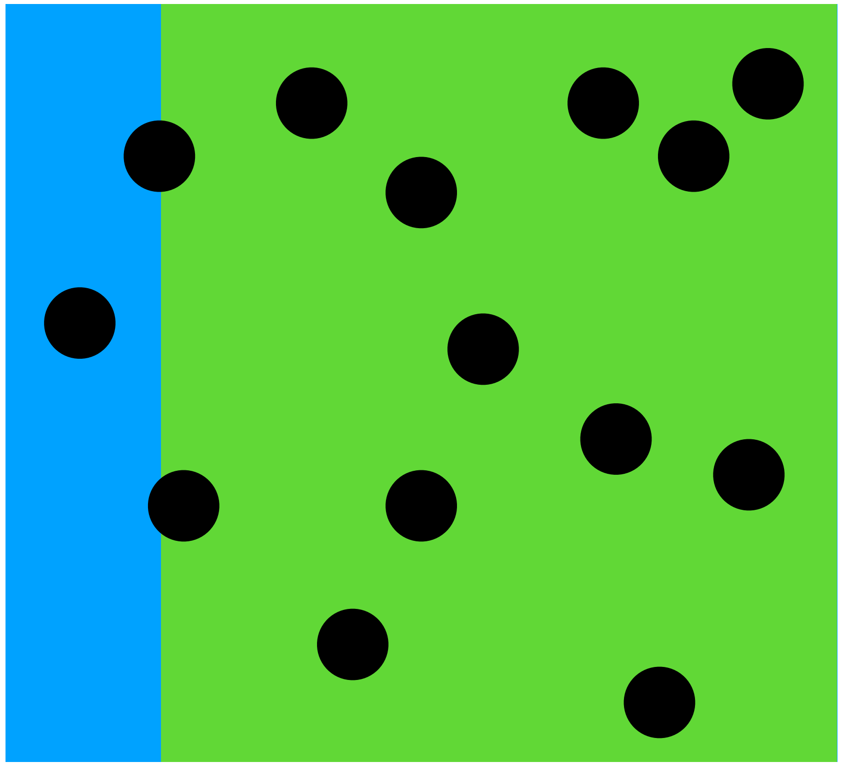
  1. We will only split the region into rectangles/boxes in a hierarchical manner.

  2. We'll create splits in a greedy fashion known as **recursive binary splitting**.



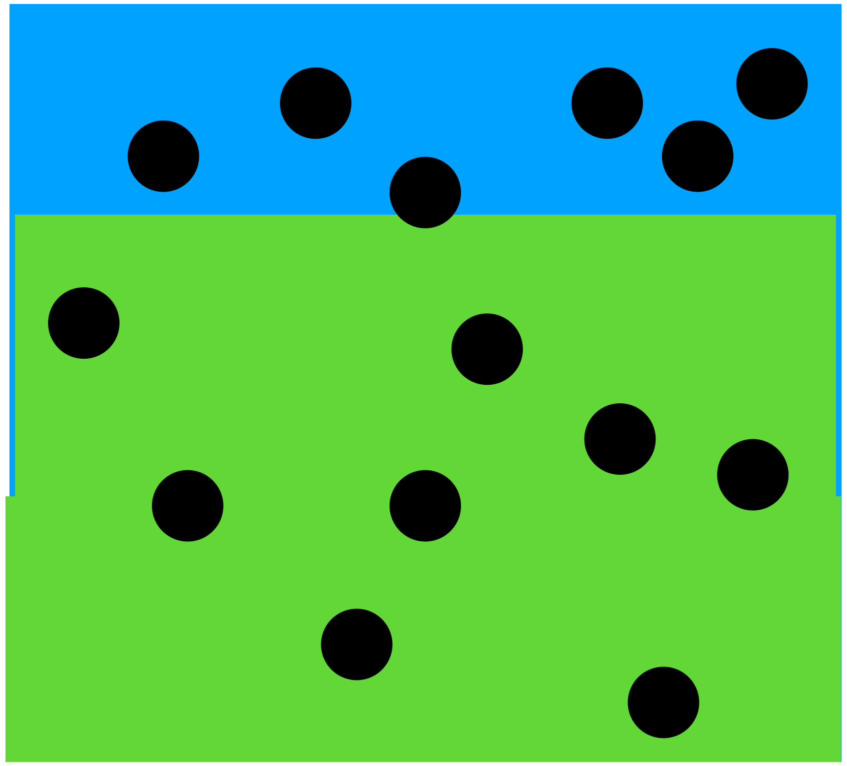
Recursive Binary Splitting

- Start with all the data.
- For all predictors X_j and cut points *t*:
  - Consider splitting the space into: $R_1(j, t) = \{X \mid X_j < t\}, R_2(j, t) = \{X \mid X_j \geq t\}$
  - 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



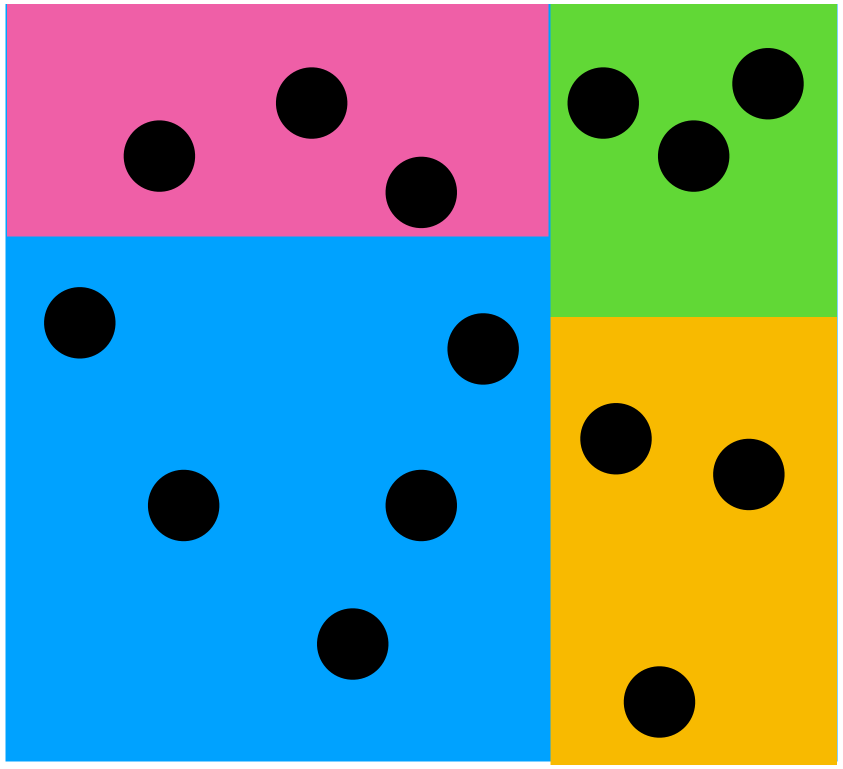
Recursive Binary Splitting

- 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: $R_1(j,t) = \{X \mid X_j < t\}, R_2(j,t) = \{X \mid X_j \geq t\}$
  - 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



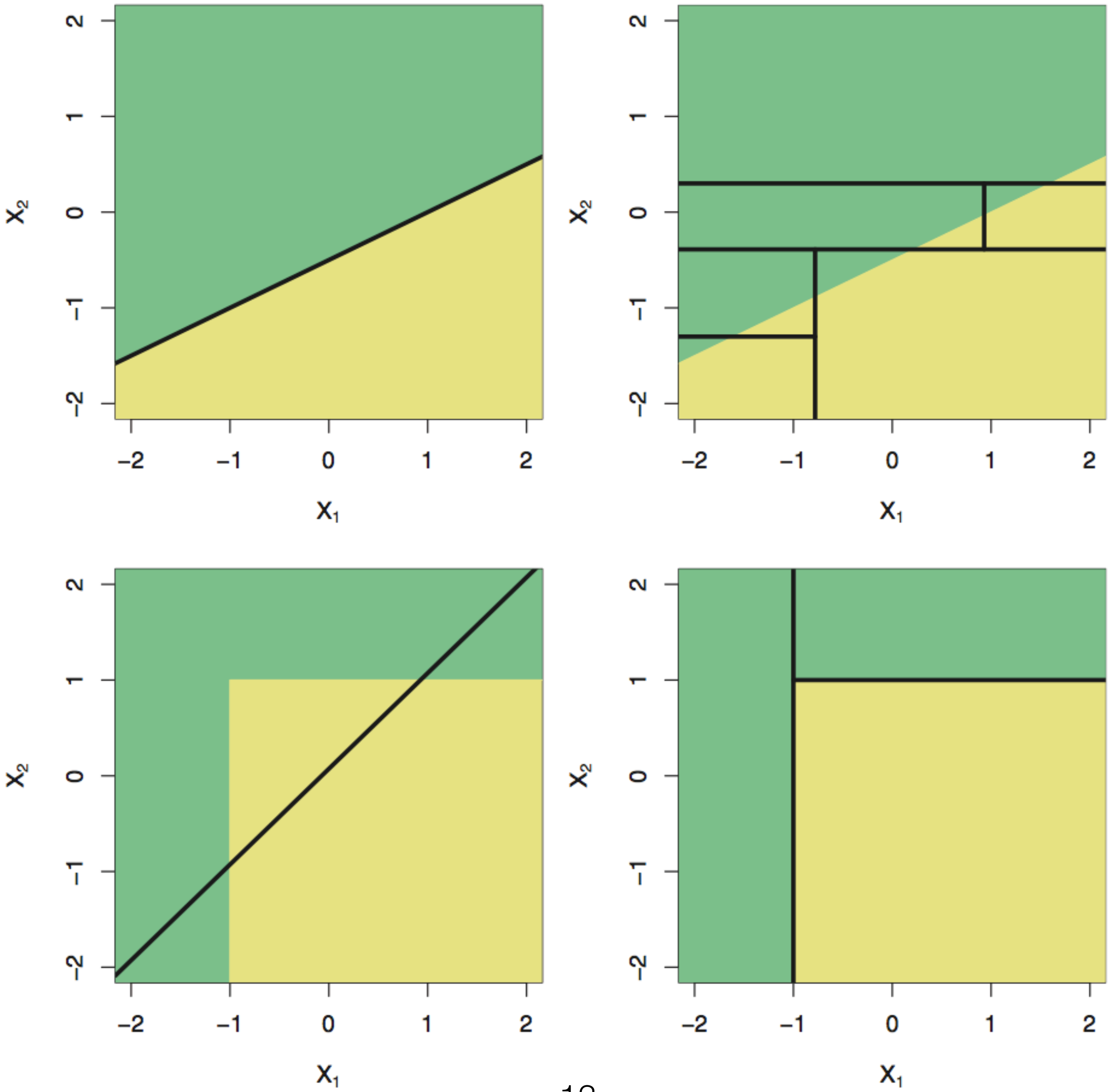
Tree versus Linear Models

Linear model

Tree-based model

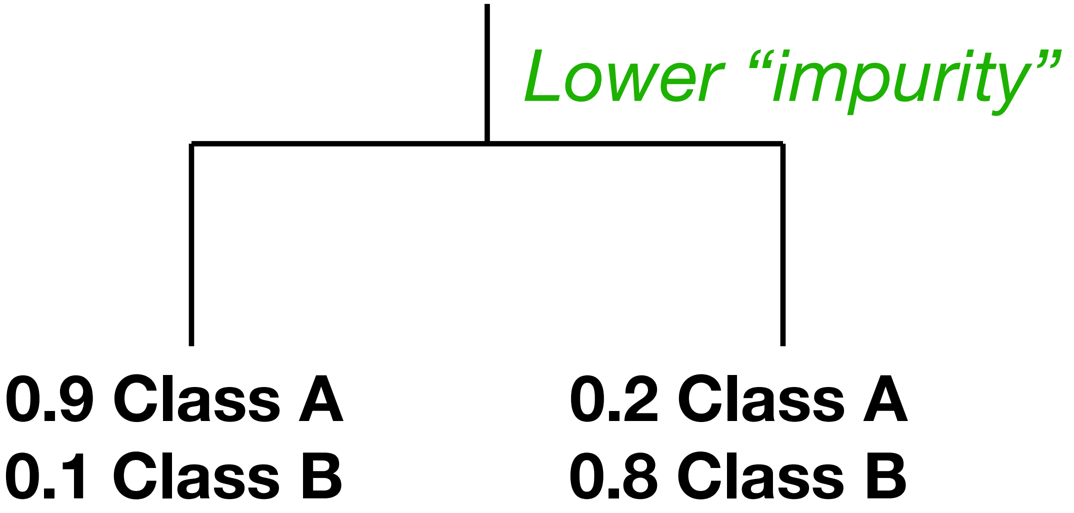
True linear boundary

True non-linear boundary



Split Criterion Functions

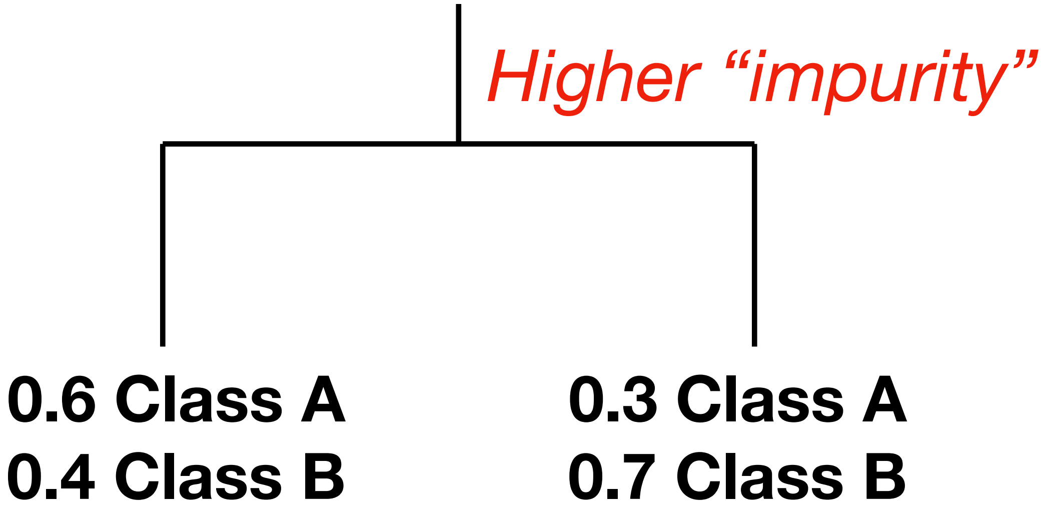
- **Classification**: Evaluate split by decrease in impurity. How often do outcomes differ from the mode for its split.
    - Gini index
    - Cross-entropy



Lower "impurity"

0.9 Class A
0.1 Class B

0.2 Class A
0.8 Class B



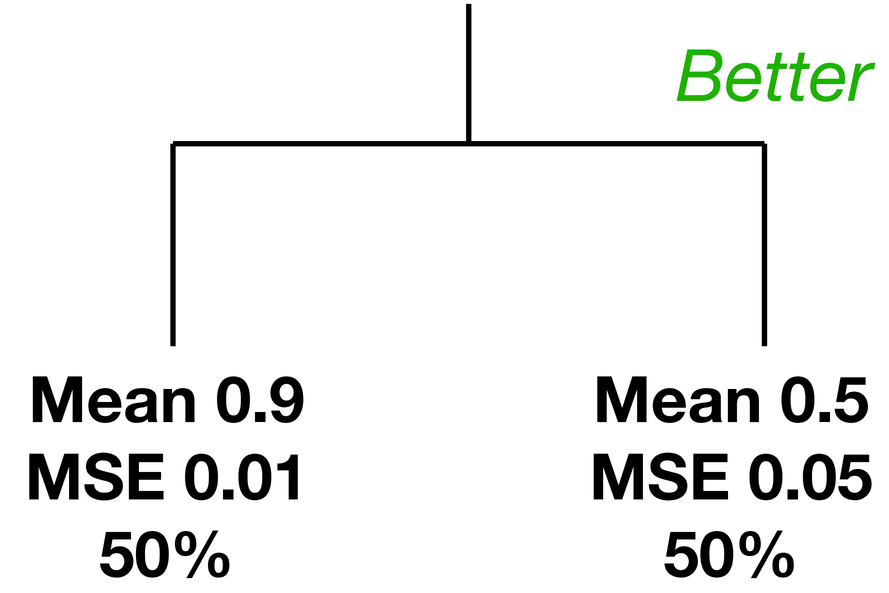
Higher "impurity"

0.6 Class A
0.4 Class B

0.3 Class A
0.7 Class B

Split Criterion Functions

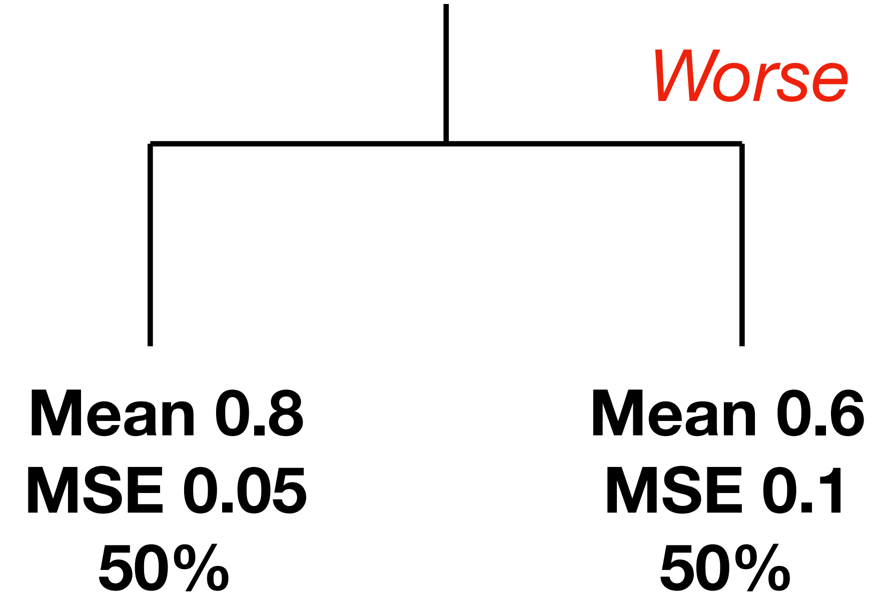
- **Regression:** Evaluate split by decrease in average deviation from the mean. How different are the outcomes from the mean for that split.
  - Squared-error deviations
  - Absolute deviations



Better

Mean 0.9
MSE 0.01
50%

Mean 0.5
MSE 0.05
50%



Worse

Mean 0.8
MSE 0.05
50%

Mean 0.6
MSE 0.1
50%

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:

$\{X_1 < 0.5\}, \{X_1 \geq 0.5\}$

$\{X_2 = A\}, \{X_2 \neq A\}$

$\{X_2 = B\}, \{X_2 \neq B\}$

$\{X_2 = C\}, \{X_2 \neq C\}$

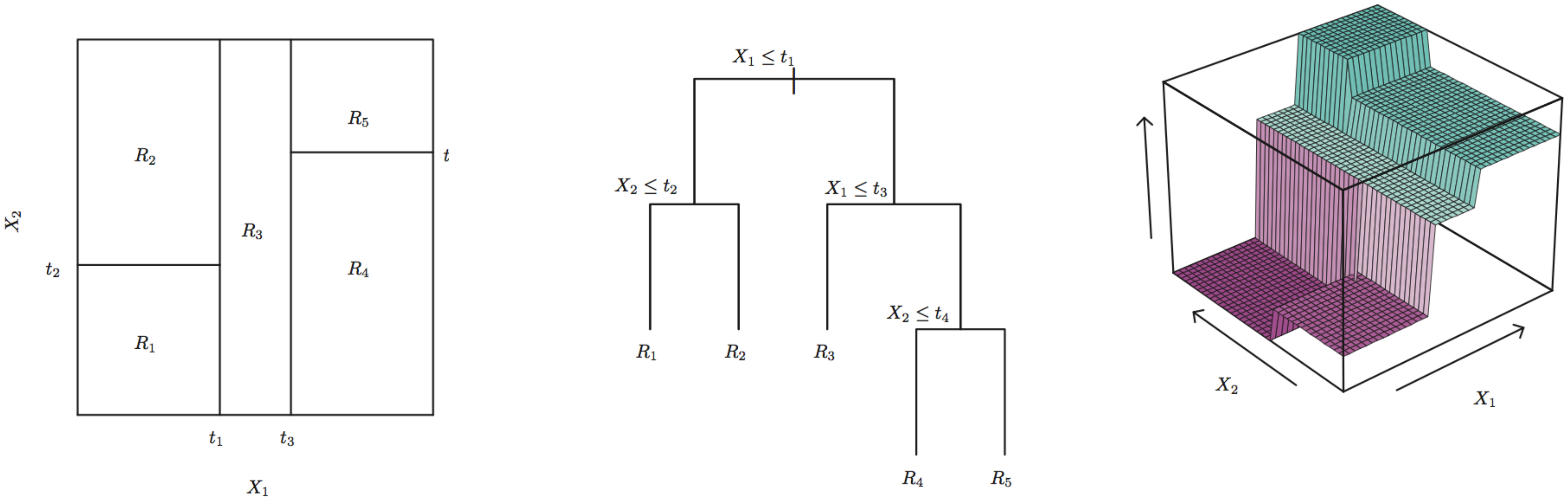
Fitting a decision tree

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), but increased **bias**.

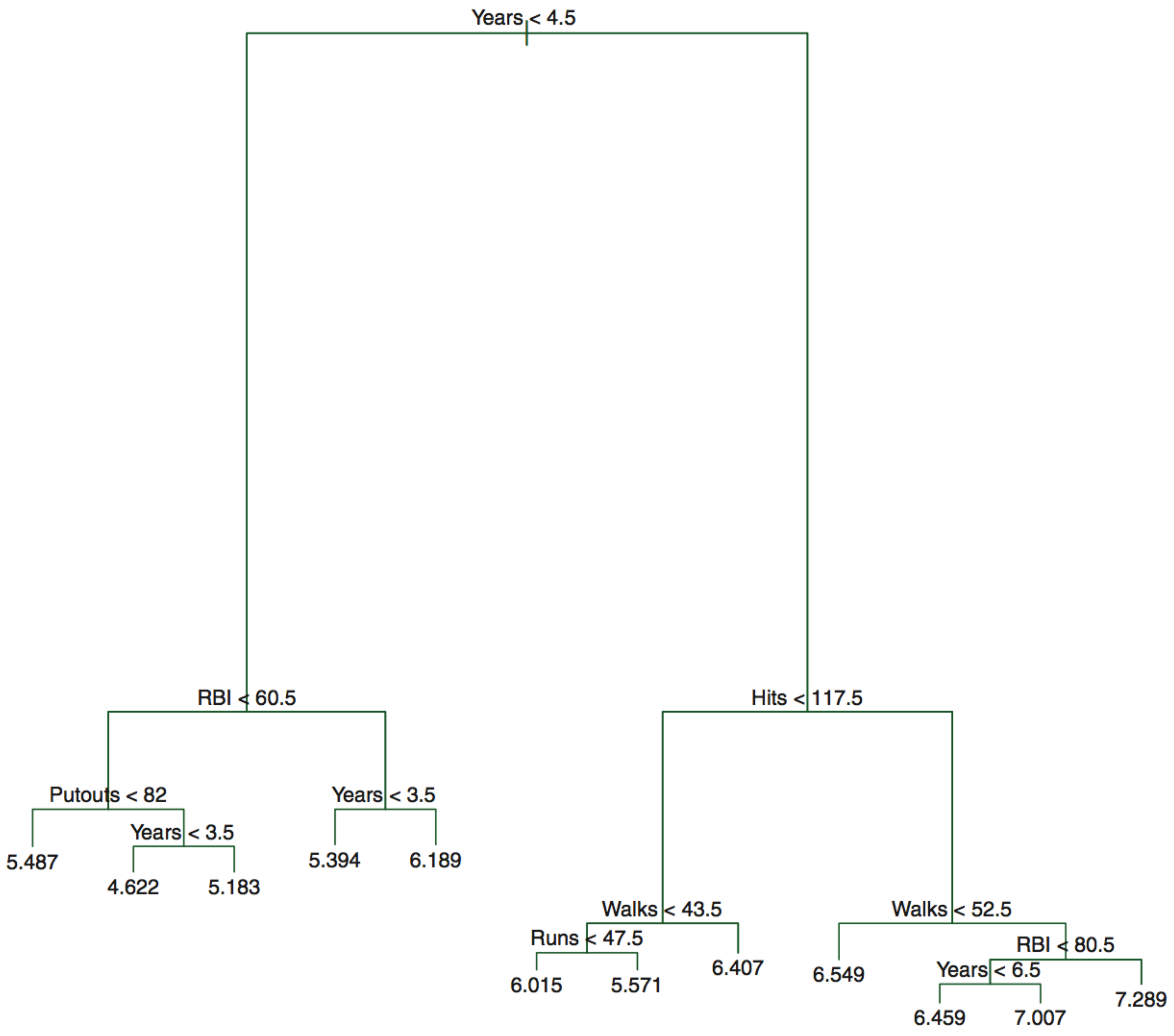
- To find a good tree, a common approach is to:

  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

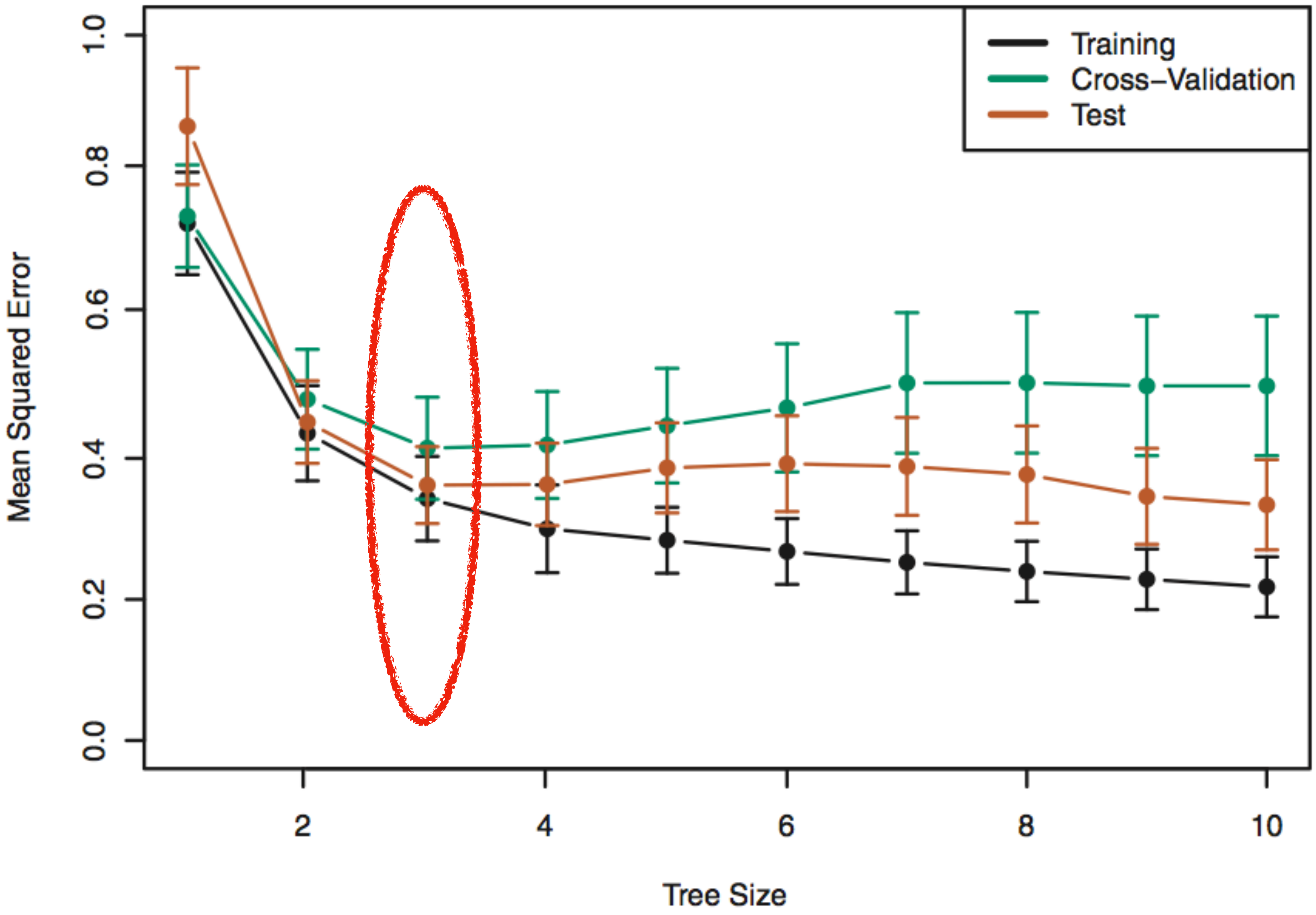


Cross-validation to determine tree size

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

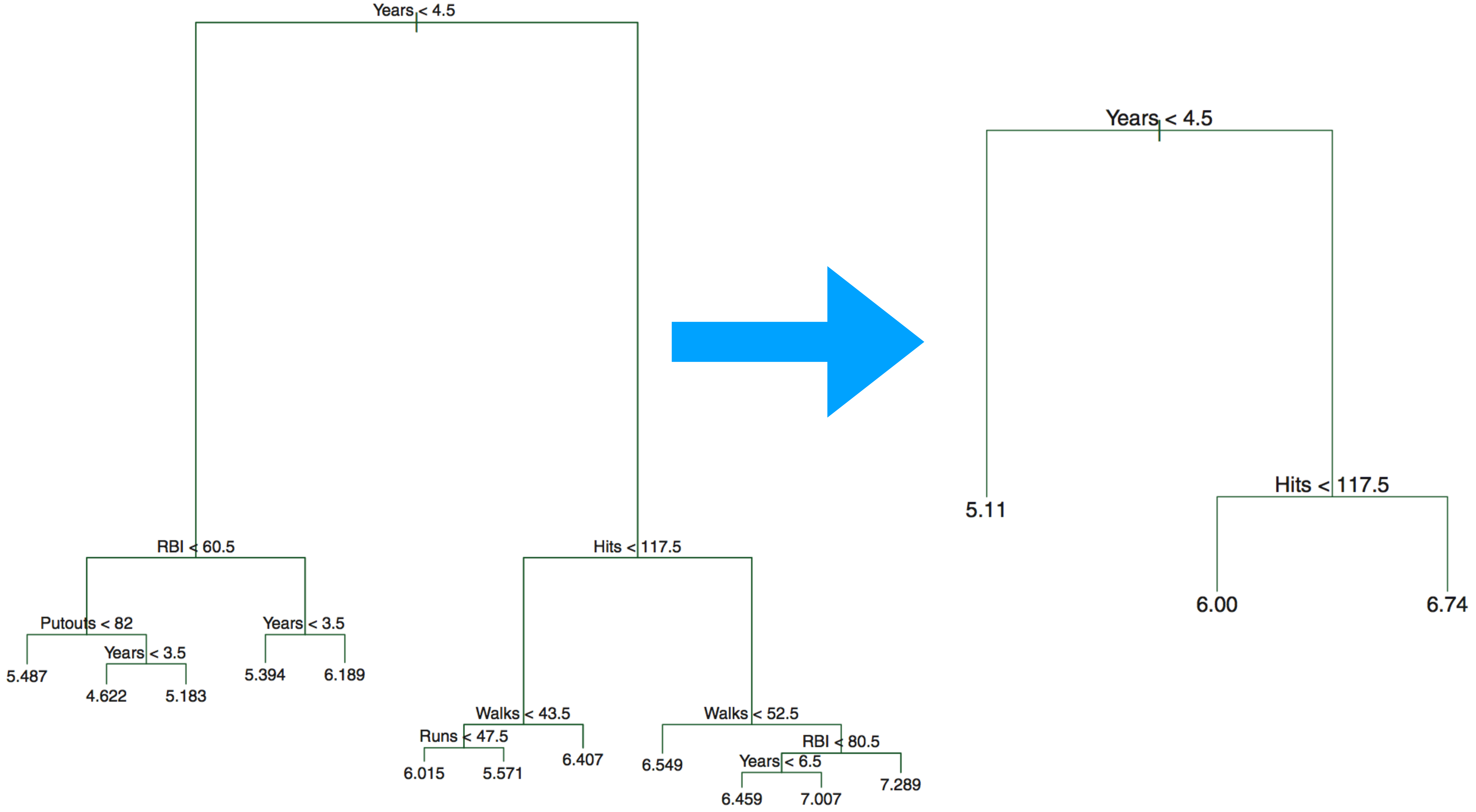
- Calculate the cross-validation error for the candidate tree sizes.

- 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



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.

Outline

- Decision Trees
- Ensembling
  - **Bagging**
  - Random Forests
  - Gradient boosted trees
- 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?

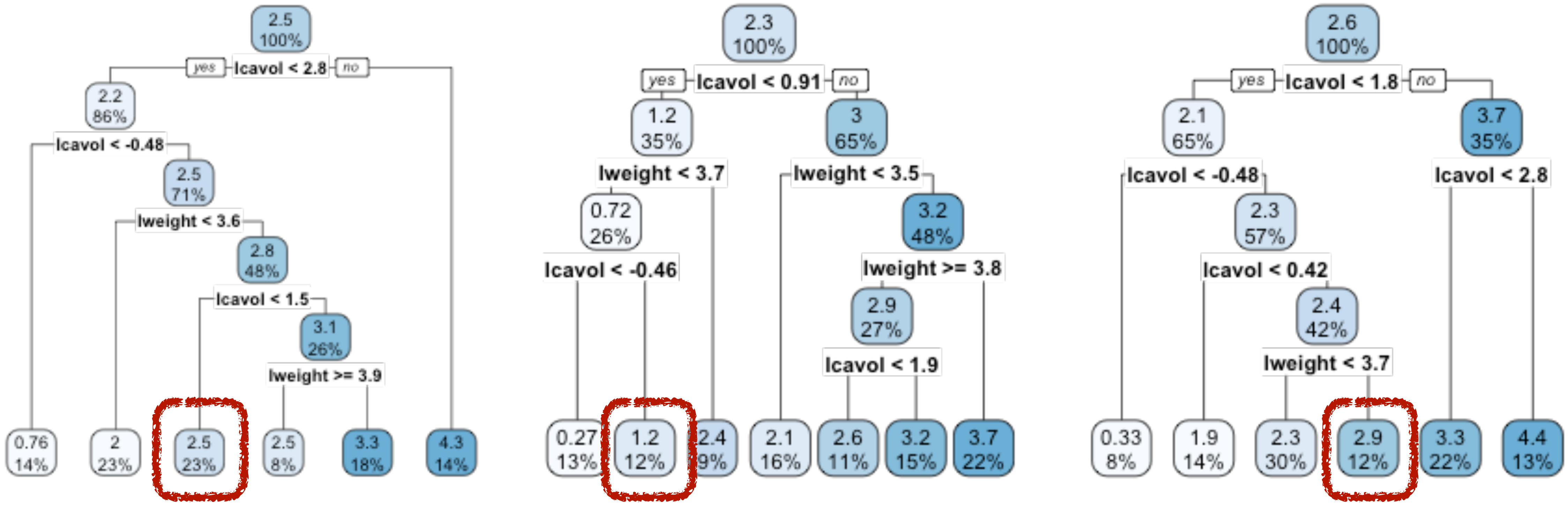
$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$$

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.
    - Train model $\hat{f}^{*b}$ on the bootstrap sample.
- 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

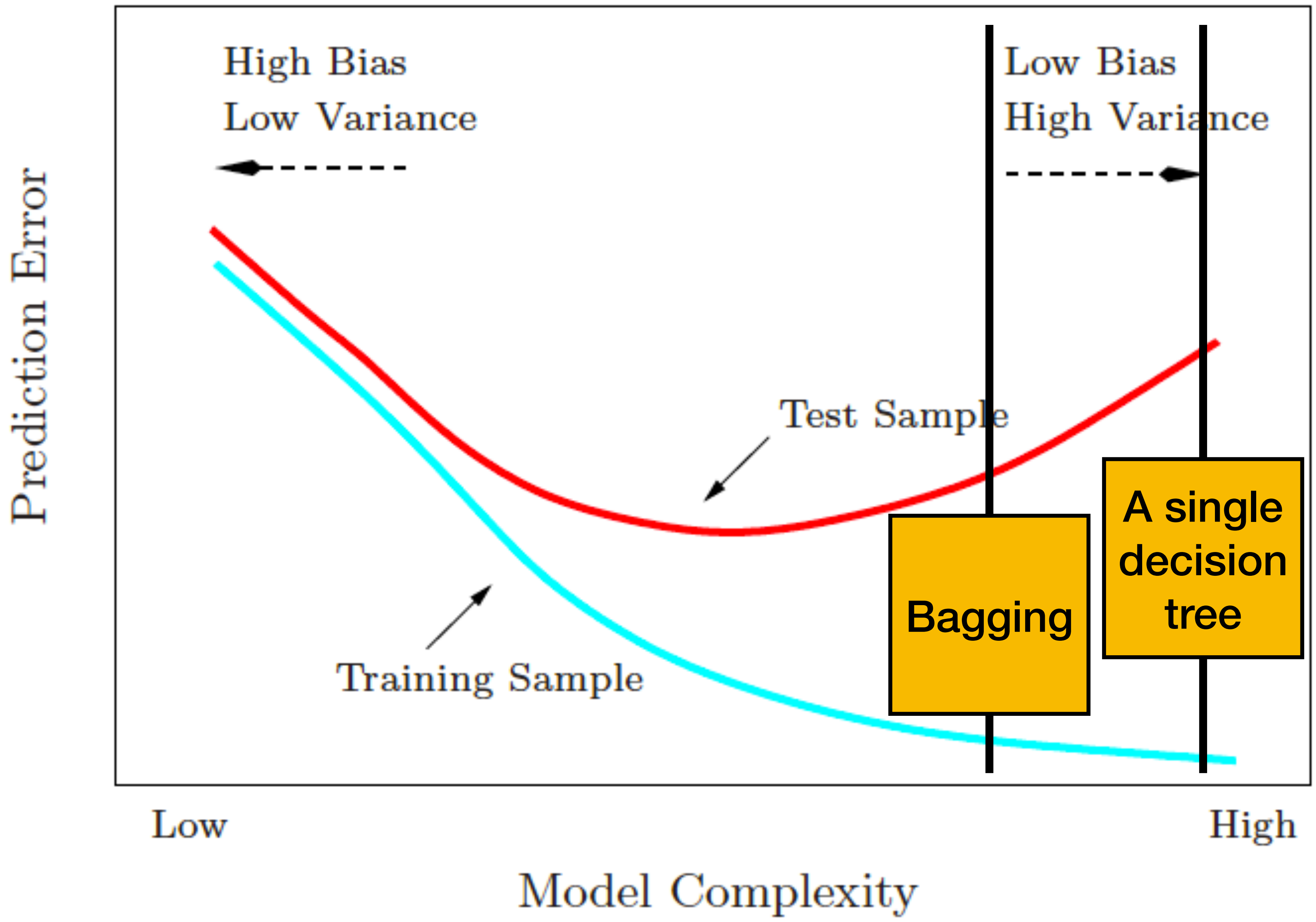
- Suppose that there is one very strong predictor in the data set, along with a number of moderately strong ones

- 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



Outline

- Decision Trees
- Ensembling
    - Bagging
    - **Random Forests**
- Variable Importance

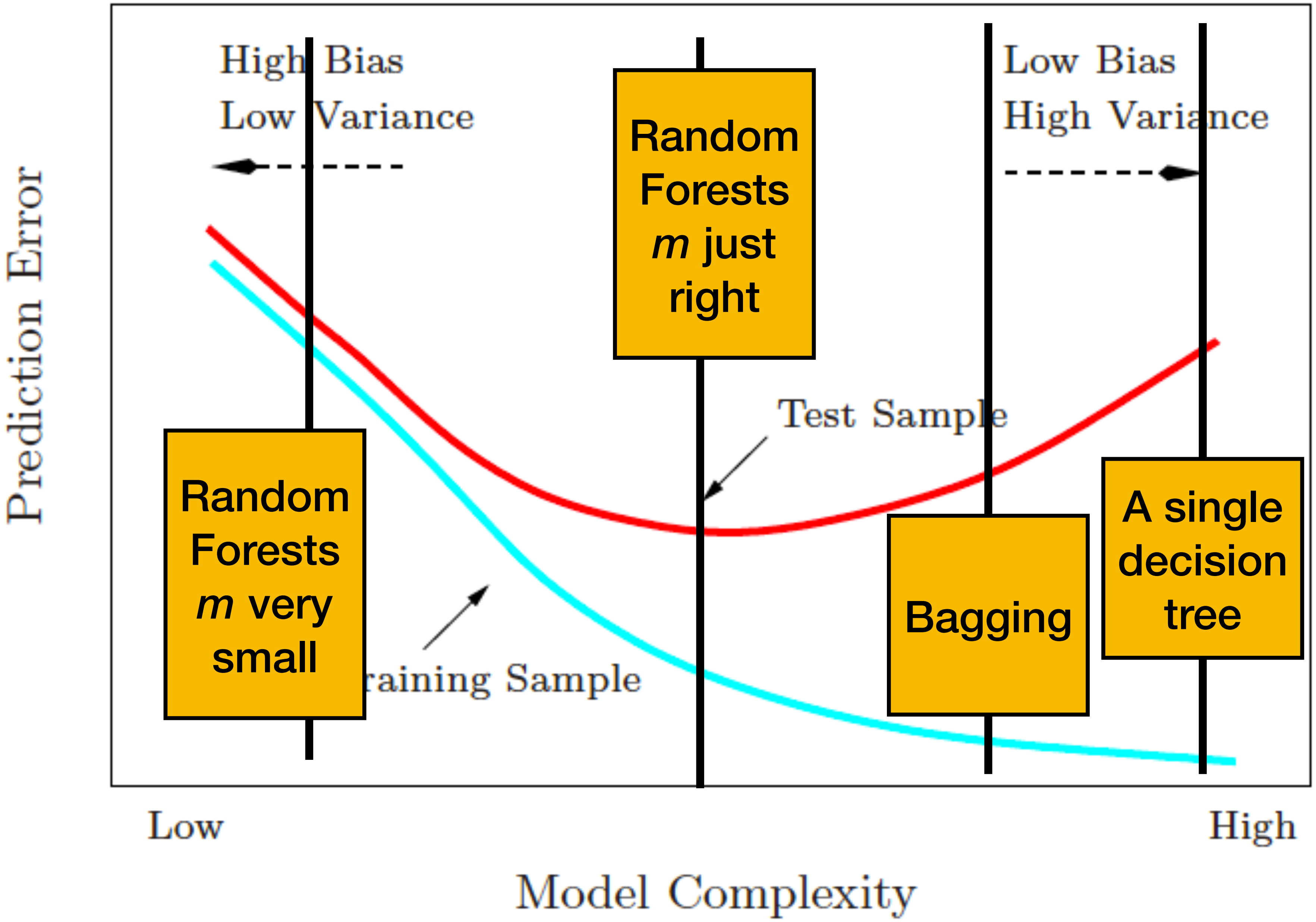
Random Forests

- 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.

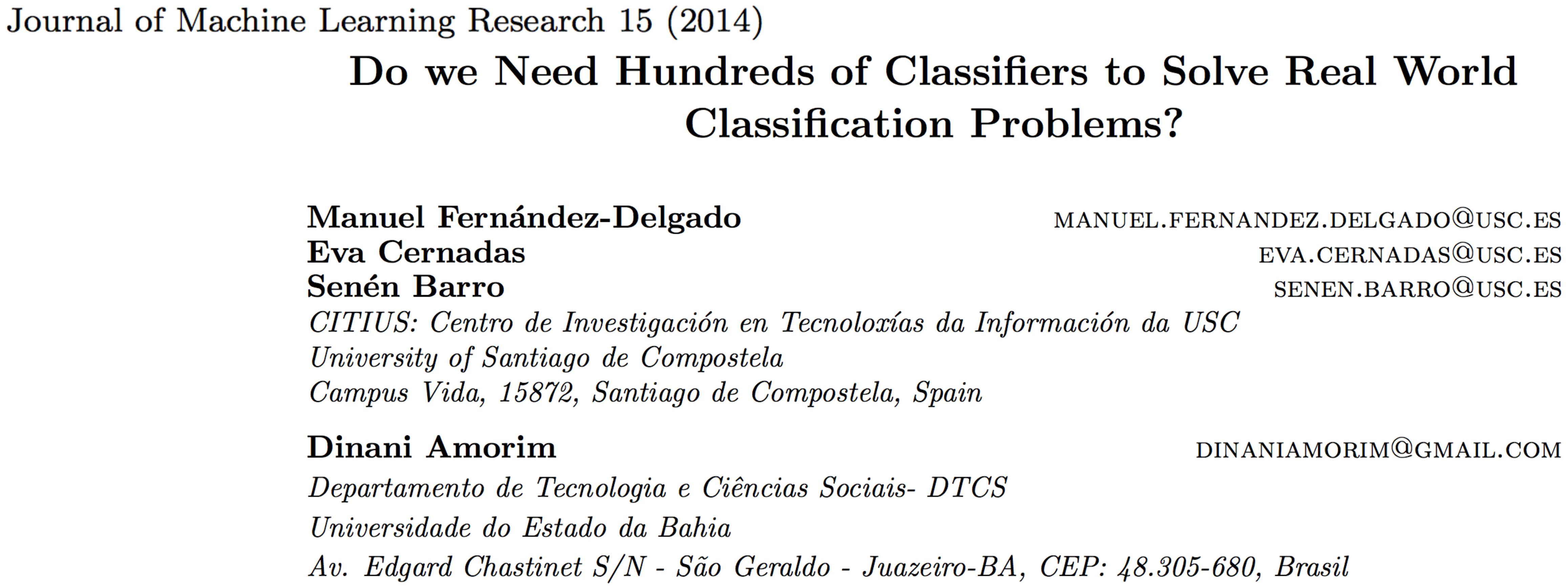
Random Forests

- 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.

Outline

- Decision Trees
- Ensembling
    - Bagging
    - Random Forests
    - **Gradient boosted trees**
- Variable Importance

The hypothesis boosting problem

- Suppose there is an efficient learning algorithm whose performance, on any dataset, is slightly better than random guessing.

- Can these weak learners be combined into an efficient algorithm to achieve very good prediction accuracy?

- Can we "boost" a set of weak learners to create a single strong learner?

"Weak learner":
an algorithm that does slightly better than random

"Strong learner":
an algorithm that achieves arbitrarily good prediction accuracy as the sample size grows

The answer is yes!

Boosting algorithms

- Boosting is a general-purpose method that builds an ensemble of “weak learners”.

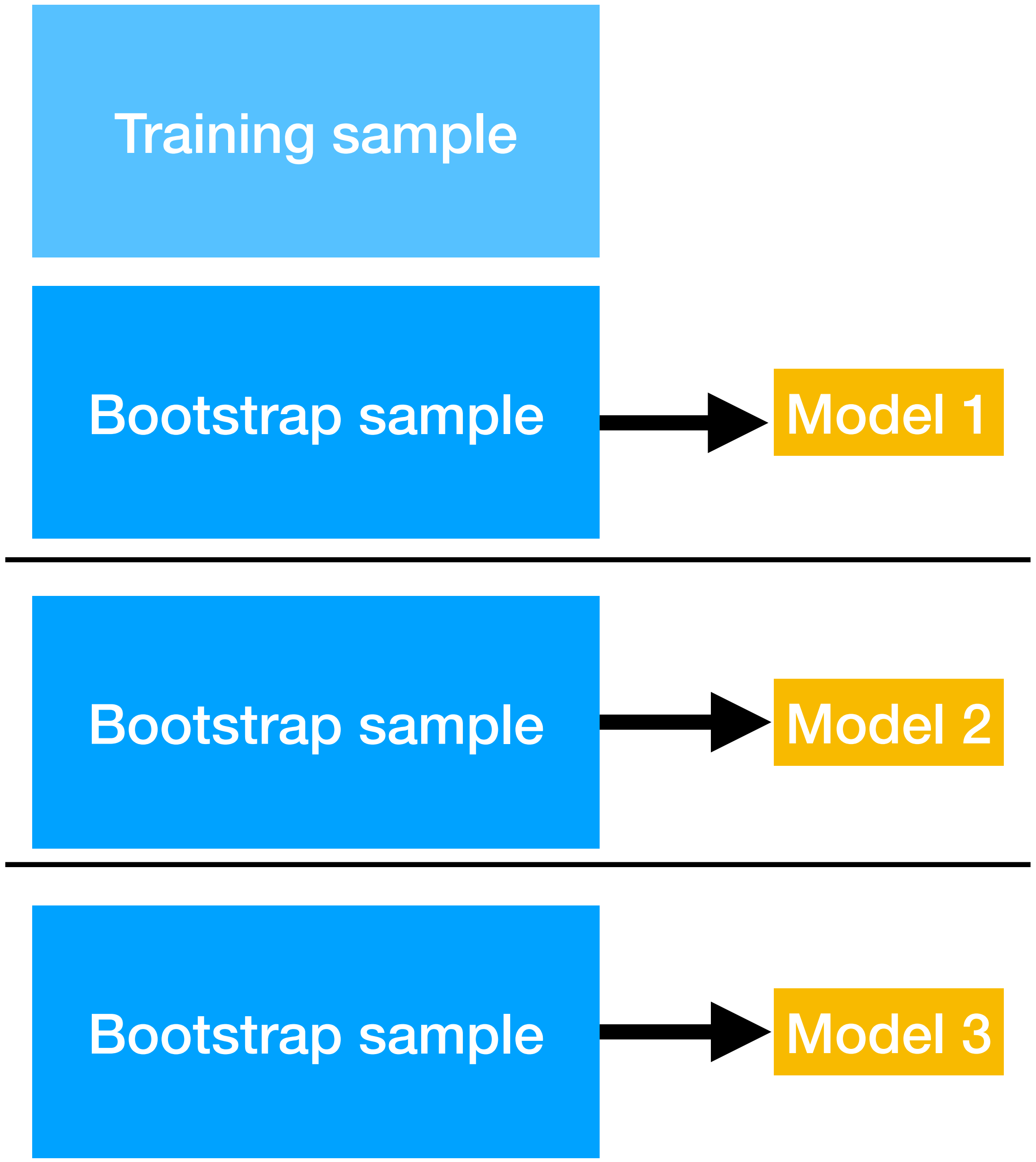
- Like bagging and random forests, boosting averages the predictions from weak learners to make a prediction.

- Unlike bagging and random forests, boosting trains each weak learner ***sequentially***. Boosting optimizes how the weak learners are combined to maximize prediction accuracy.

- Can be used for both regression and classification.

Ensemble methods

Bagging-type methods:
Models are built independently



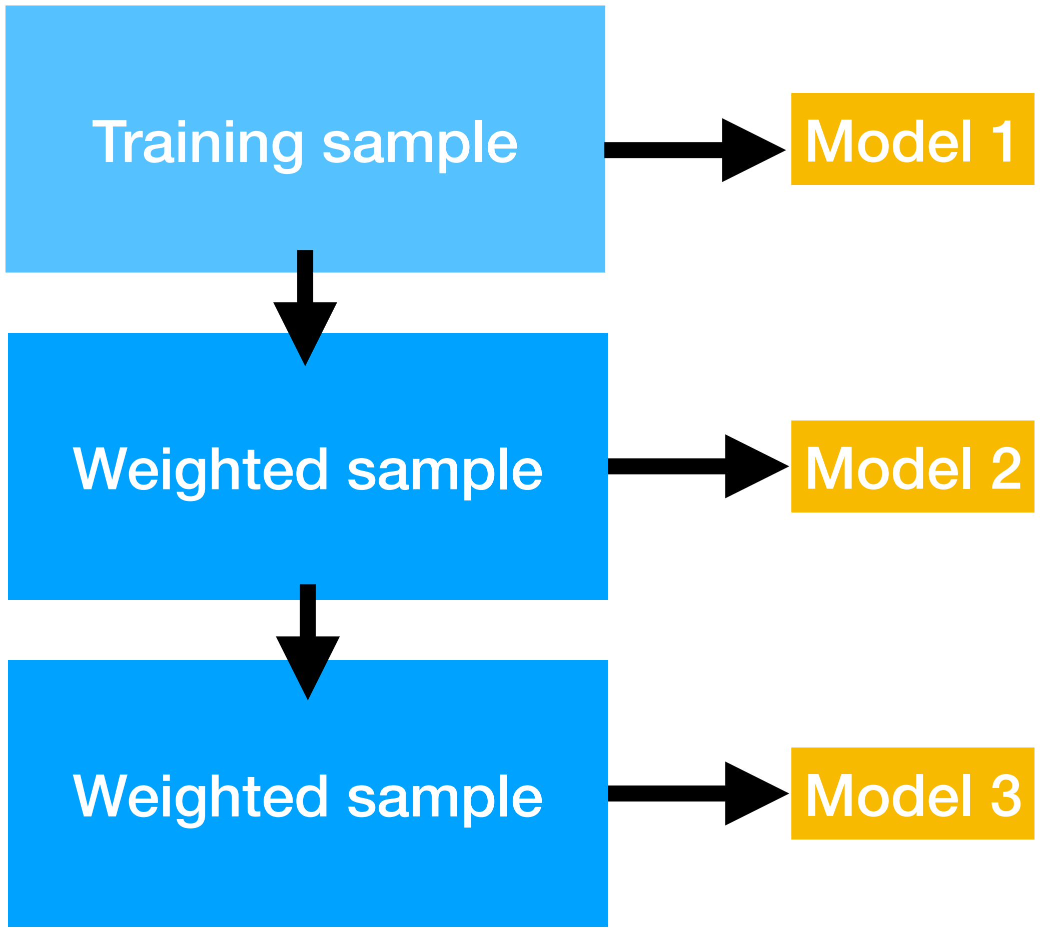
Training sample

Bootstrap sample → Model 1

Bootstrap sample → Model 2

Bootstrap sample → Model 3

Boosting:
Models build off one another



Training sample → Model 1

Weighted sample → Model 2

Weighted sample → Model 3

The many variants of Boosting

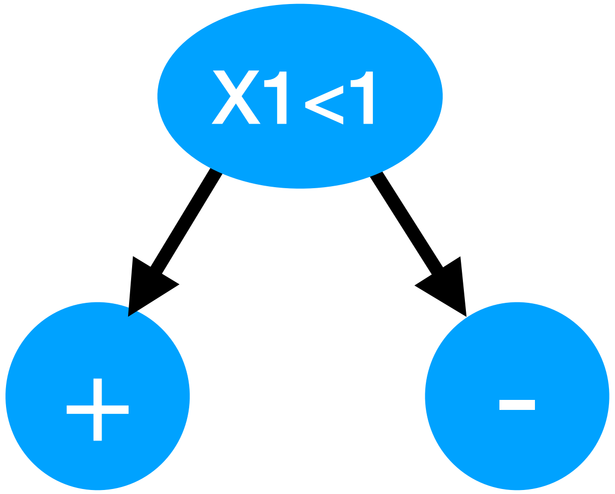
- **Adaptive Boosting (AdaBoost):** The first major boosting algorithm. Designed for classification.

- **Forward Stagewise Additive Modeling**: Generalization of AdaBoost

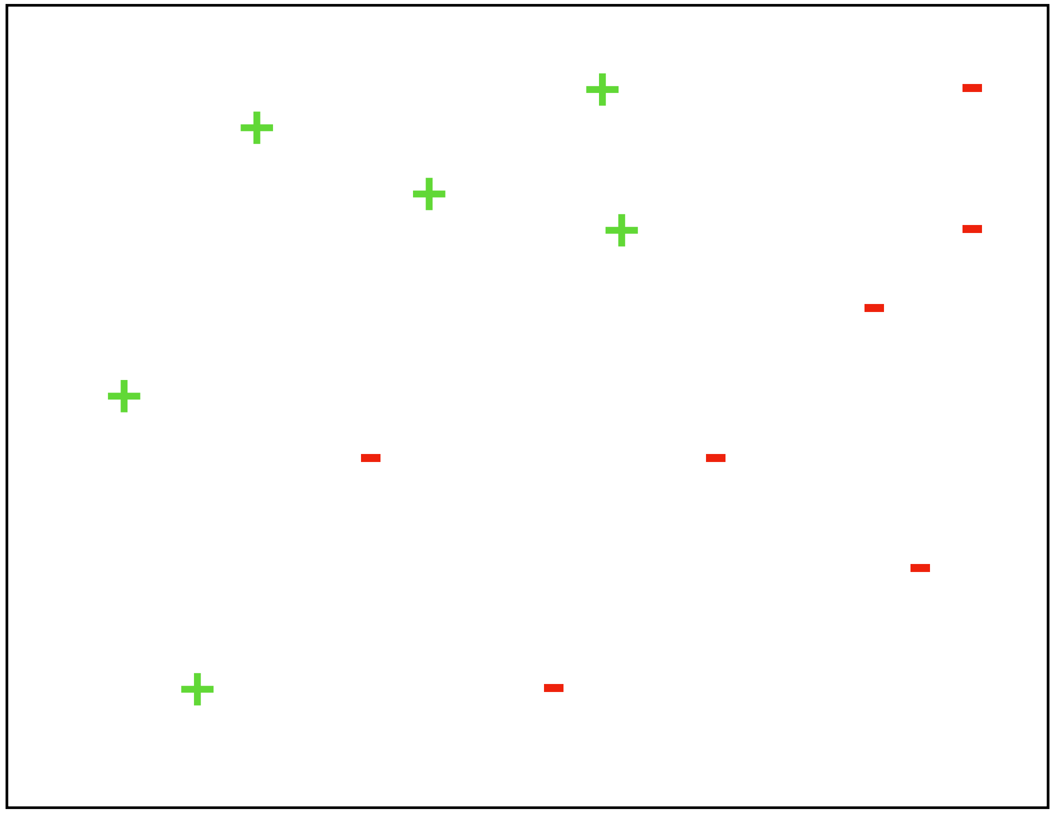
- **Gradient Boosting**: A gradient-based perspective of Boosting

AdaBoost: the basic idea

- Let’s suppose our “weak learners” are decision stumps:



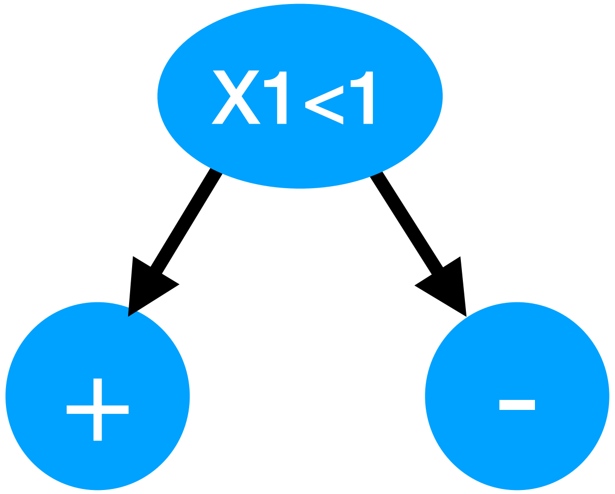
Ex: Decision stump



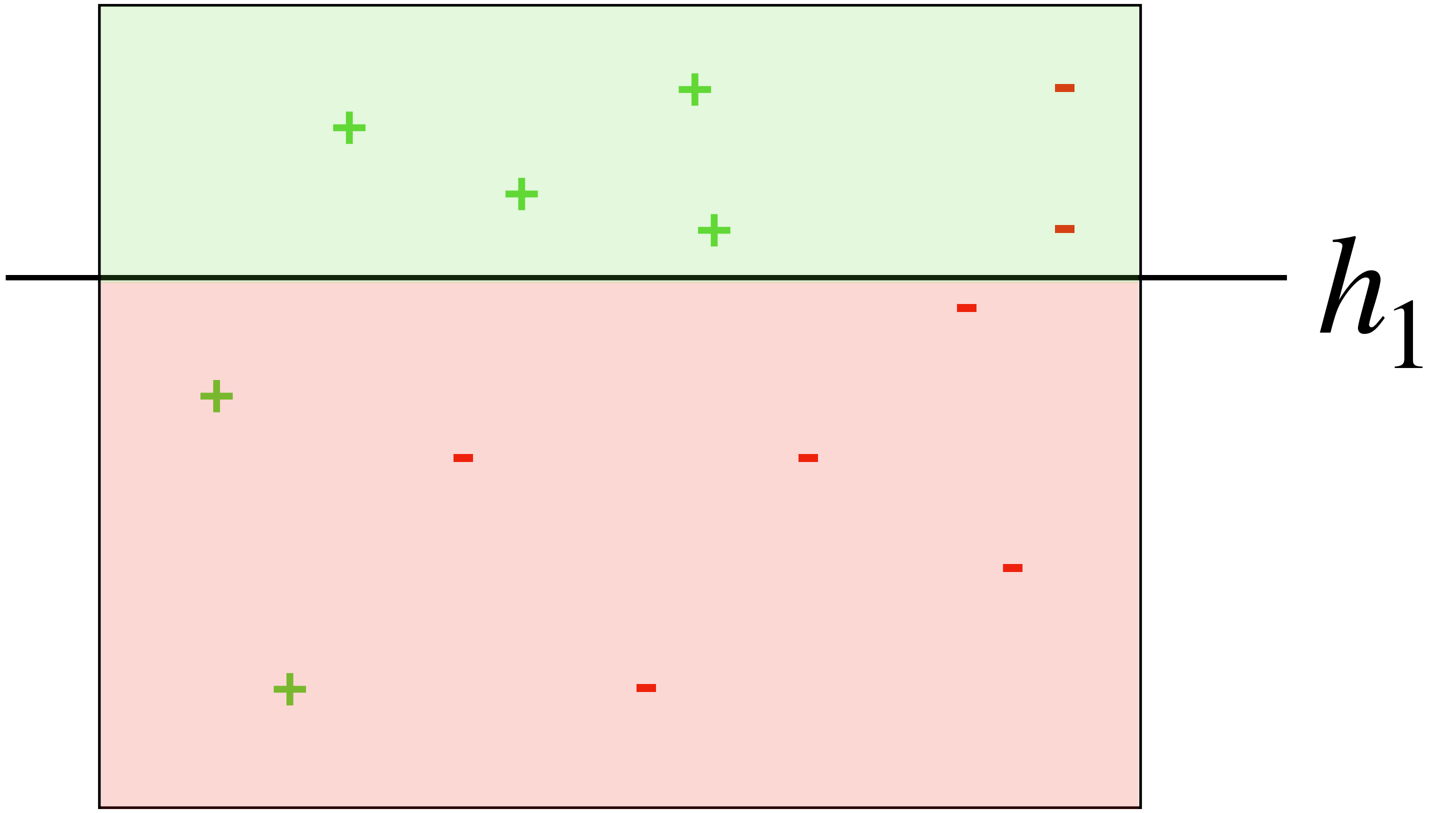
AdaBoost: the basic idea

- Let’s suppose our “weak learners” are decision stumps:

Step 1



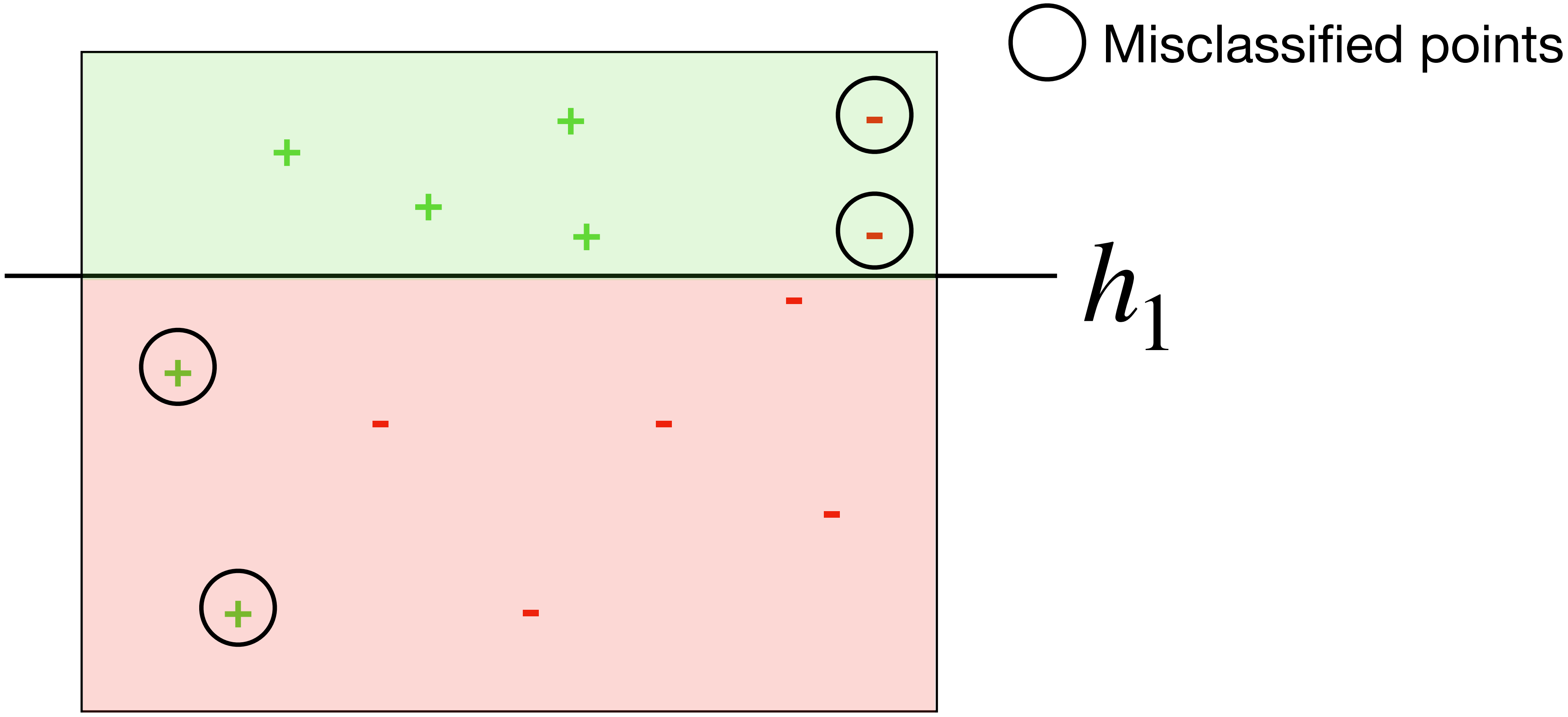
Ex: Decision stump



AdaBoost: the basic idea

- There are a lot of points that are misclassified. Let's try to fix this.

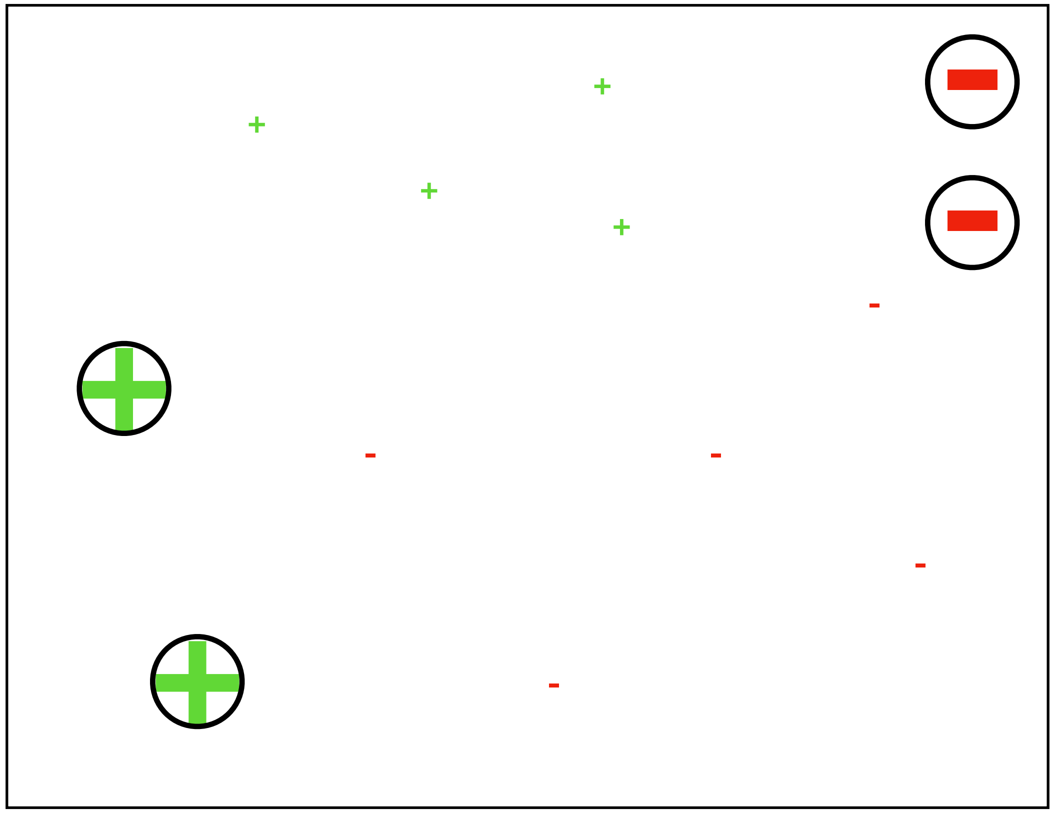
Step 1



AdaBoost: the basic idea

- Let’s upweight misclassified points and downweight correctly classified points. Fit a new decision stump on this reweighed dataset.

Step 2

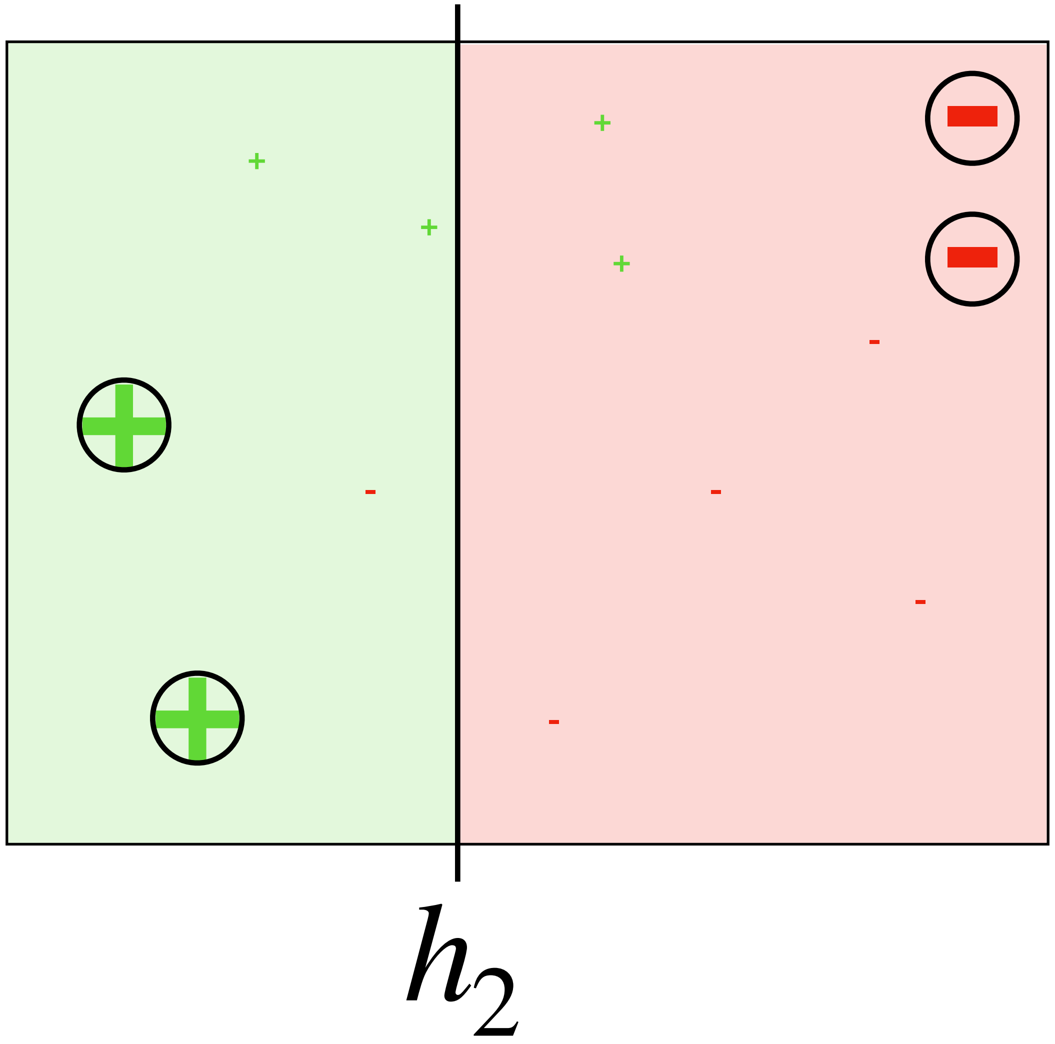


○ Misclassified points

AdaBoost: the basic idea

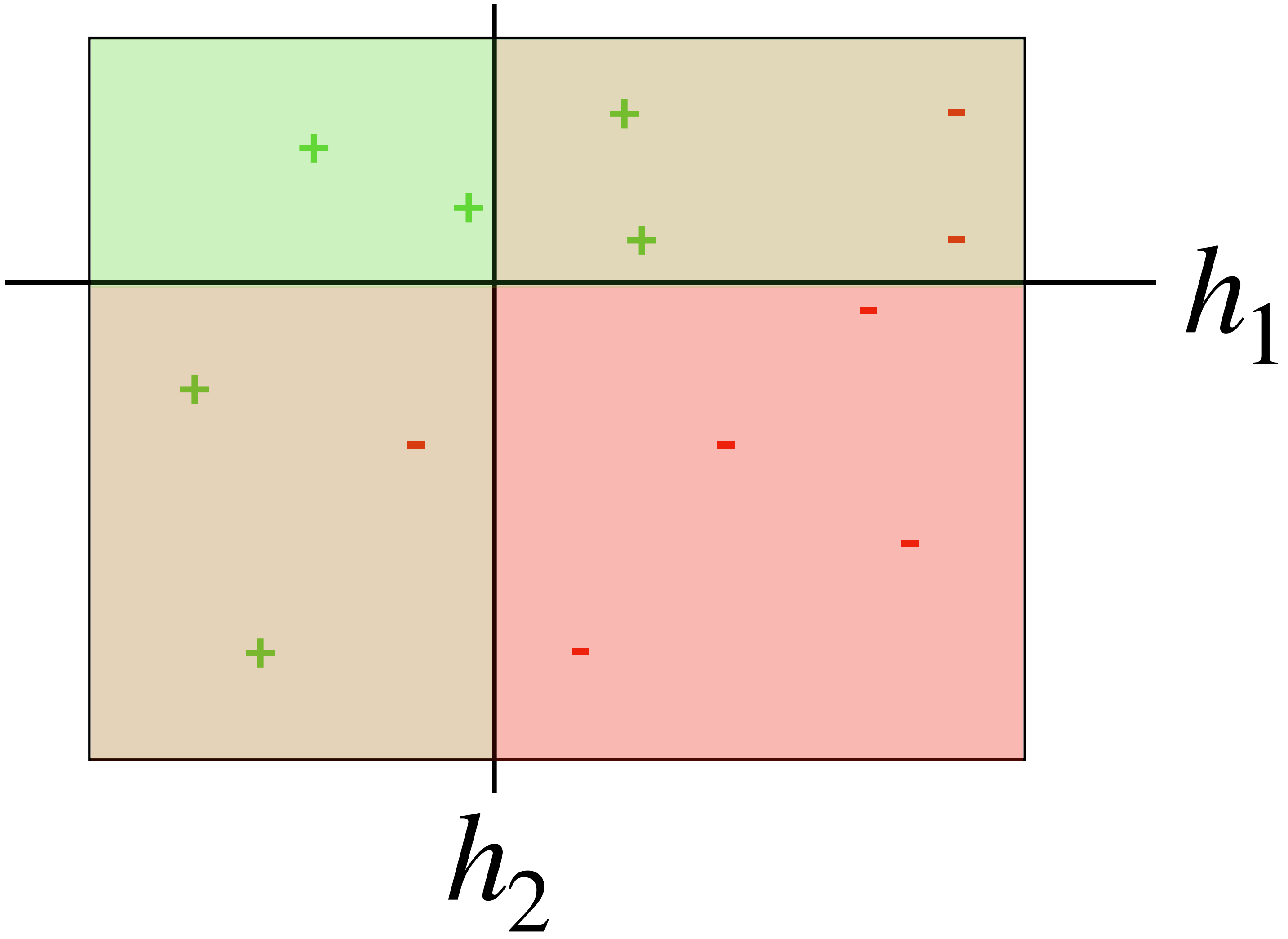
- Based on this reweighed dataset, we fit the following decision stump:

Step 2



AdaBoost: the basic idea

- The combined model looks like:



Moreover, we can take a weighted average of these two models:

$$f(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x)$$

Weights

AdaBoost vs Gradient Boosting

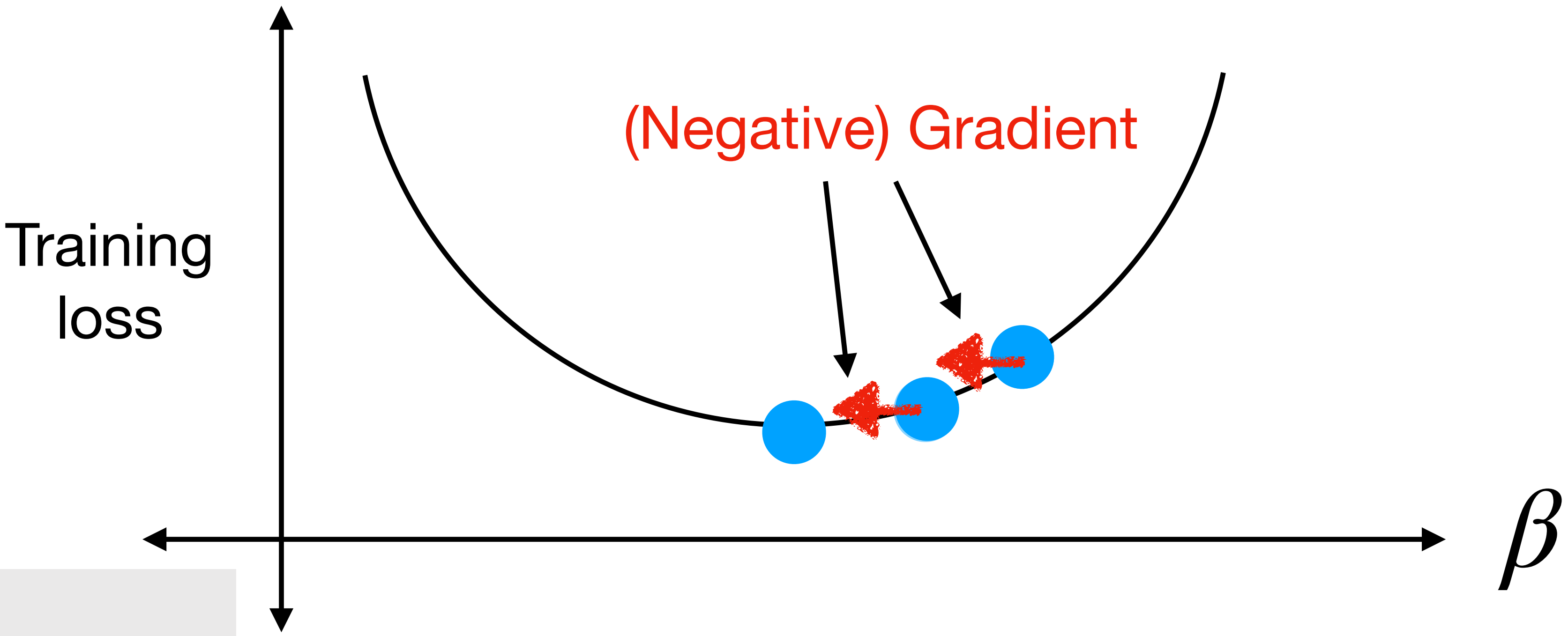
- Each successive tree is fit on a **reweighed version of the original dataset**, where the weighting scheme emphasizes the misclassified points.

- Each successive tree is fit to predict the **gradient/ residuals**

Recall: A gradient-based approach

- The **gradient** of a function is the direction that increases the function's value the most. The **negative gradient** is the direction of "**steepest descent**" of the function's value.

- To minimize the training error, many machine learning algorithms compute the gradient and follow the direction of steepest descent.



Gradient update:

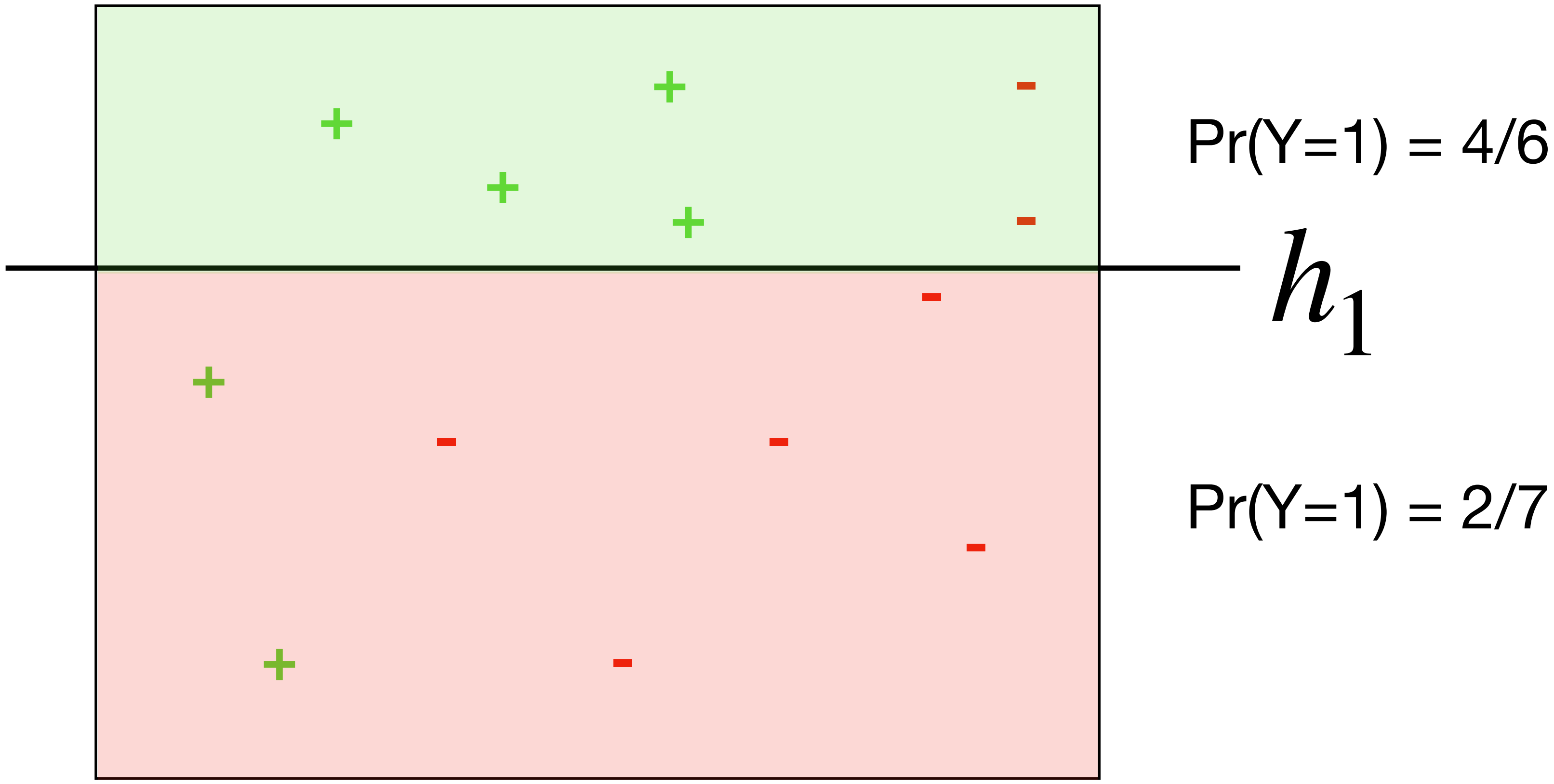
$$\beta \leftarrow \beta - \lambda \underbrace{\nabla_\beta \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_\beta(x_i))}_{\text{Gradient}}$$

Step size

The step size λ determines how fast we update the model based on the gradient.

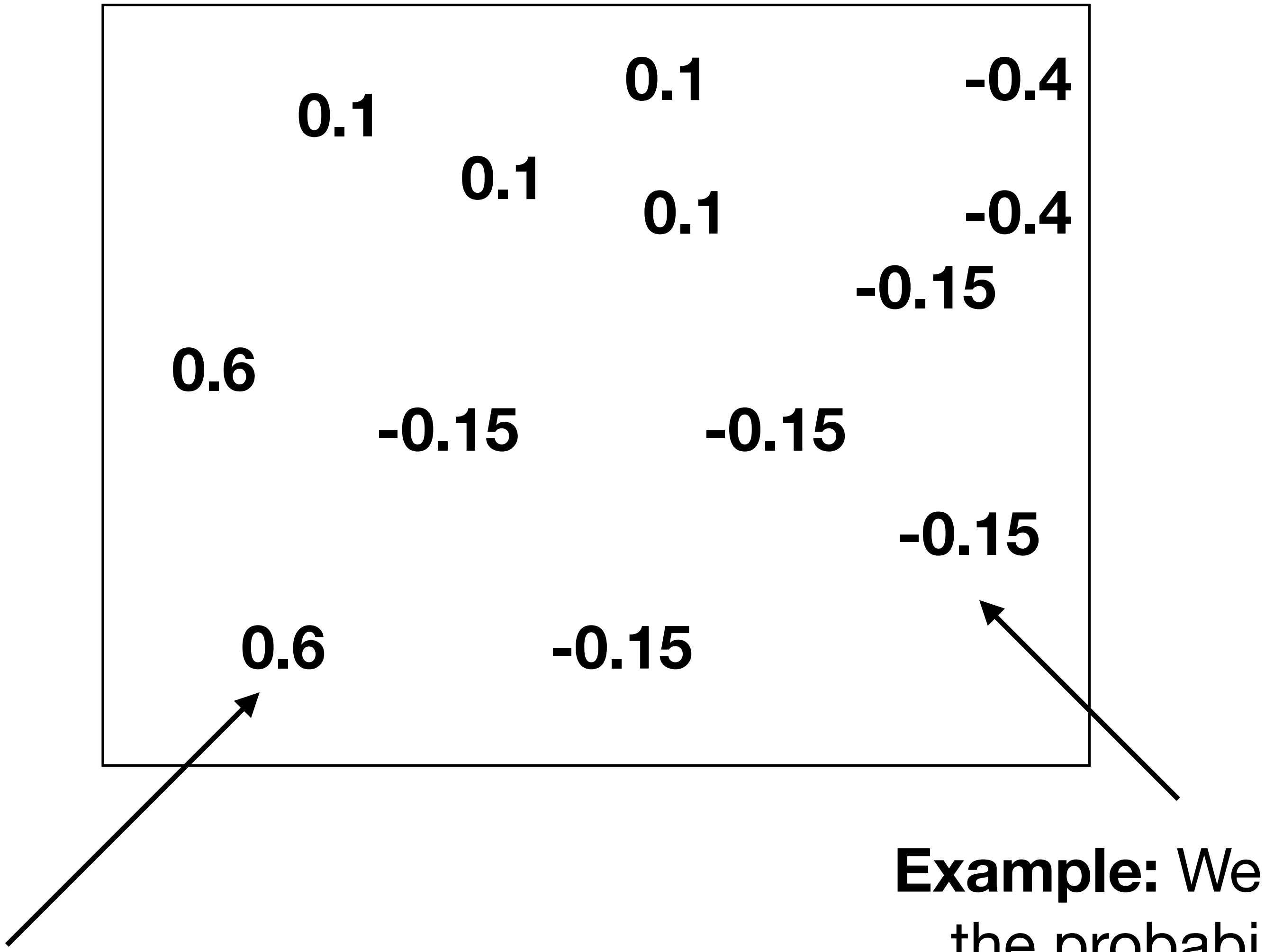
Gradient-boosting: the basic idea

- **Step 1**: Fit a “weak learner” (Here we use probability decision stumps)



Gradient-boosting: the basic idea

- **Step 2:** Compute the negative gradient $-\frac{\partial \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i))}{\partial f(x_i)}$ at observations $i = 1,2,...,n$ to determine how we can decrease the training error.

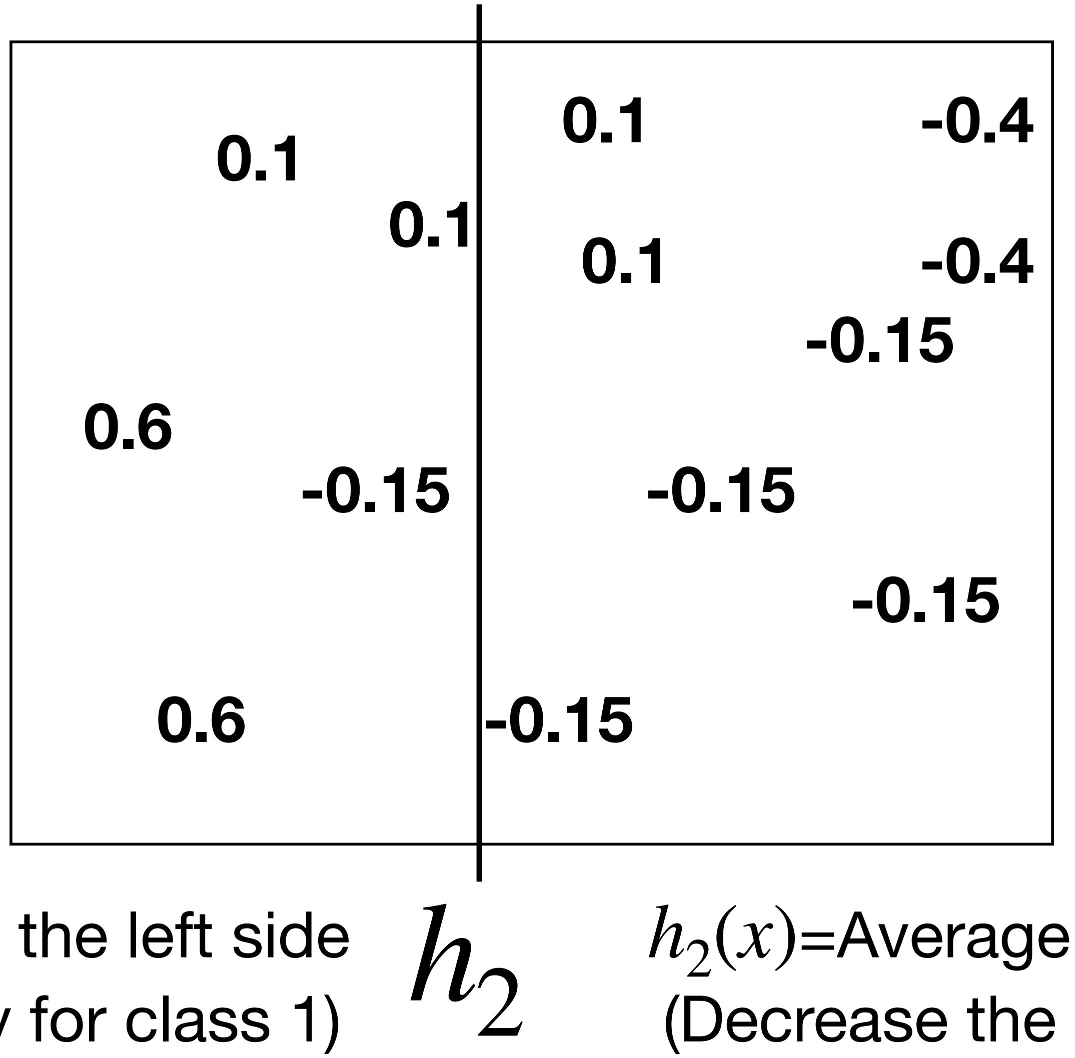


Example: We should decrease the probability assigned to class 1 for this label by a little.

Example: We should increase the probability assigned to class 1 for this label by a lot.

Gradient-boosting: the basic idea

- **Step 2B:** Fit a decision stump that best mimics the values of the gradient.

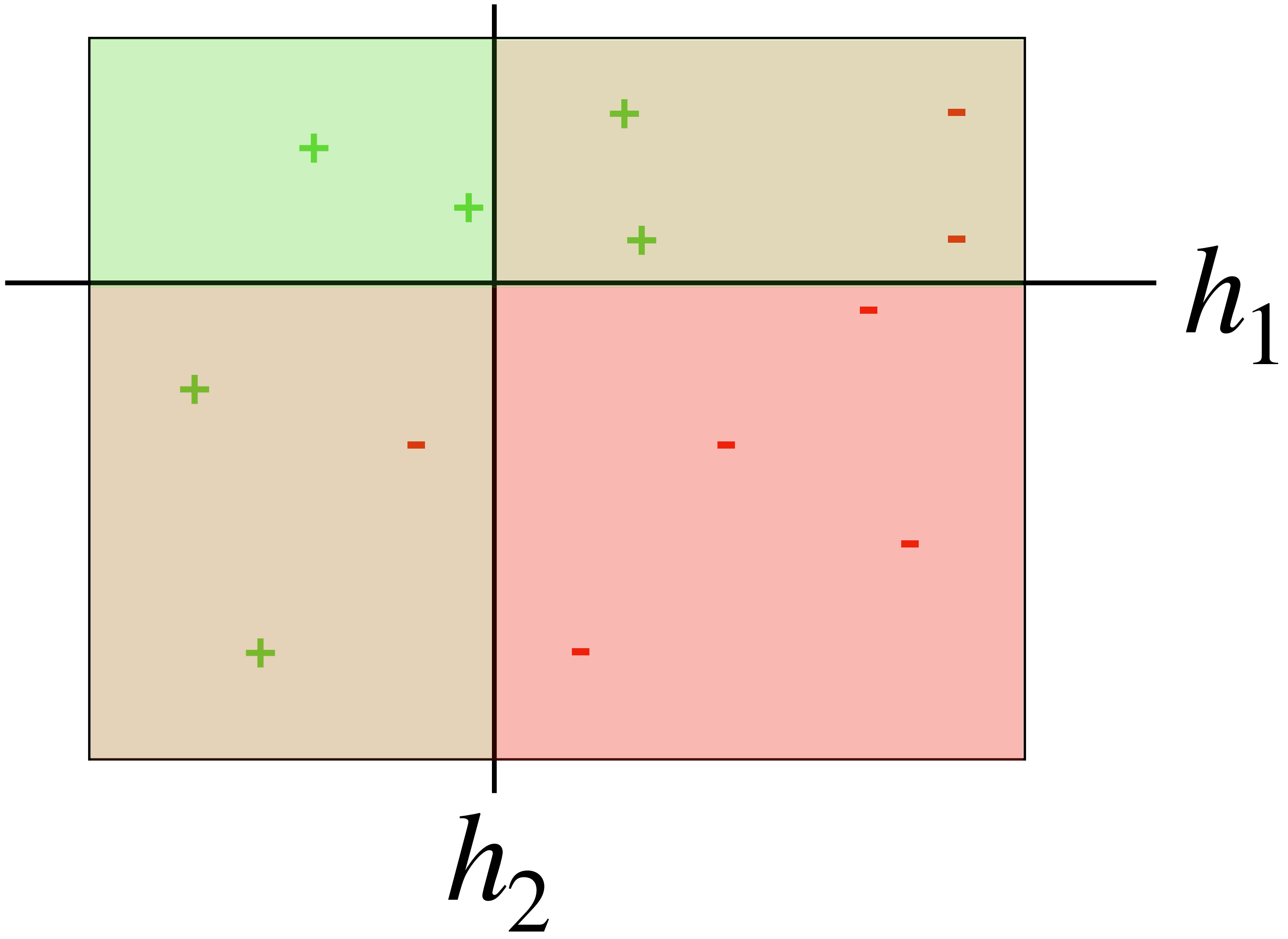


$h_2(x)$=Average value on the left side (Increase the probability for class 1)

$h_2(x)$=Average value on the right side (Decrease the probability for class 1)

Gradient-boosting: the basic idea

- **Step 3**: Combine the two models



$$f(x) = \underset{\nwarrow}{\alpha_1} h_1(x) + \underset{\nearrow}{\alpha_2} h_2(x)$$

Weights

Gradient boosting

- At each step, fit a model that best mimics the direction of the gradient.

- For b = 1,2,…B:
  - Calculate the negative gradient of the training loss with respect to the predicted outcomes.
  - Find a model h_b and weight α_b that best mimics the negative gradient.
  - Add this new model to the ensemble for step size λ:

$$f(x) = h_1(x) + \lambda \sum_{j=2}^{b} \alpha_j h_j(x)$$

Gradient boosted + Trees

- **Gradient boosted trees** are very popular off-the-shelf algorithms because of their high predictive accuracy.

- The tuning parameters you need to think about are:

  - **The number of splits *d* in each tree**: Controls the complexity of each tree. In practice, $d = 1$ often works well.

  - **The number of trees *B***: As B increases, the complexity of our boosted model increases.

  - **The shrinkage parameter λ**: λ controls the rate of learning (i.e. the step size), where a small value means that the model update at each iteration is small.

Outline

- Decision Trees
- Ensembling
  - Bagging
  - Random Forests
  - Gradient boosted trees
- **Variable Importance**

Why do we care about variable importance?

What is the importance of different **variables** X *in our prediction model* for predicting **outcome** Y? $\Bigg\}$ *Algorithmic variable importance*

What is the importance of different **variables** X *in the oracle model* for predicting **outcome** Y? $\Bigg\}$ *Population variable importance*

The zoo of variable importance methods

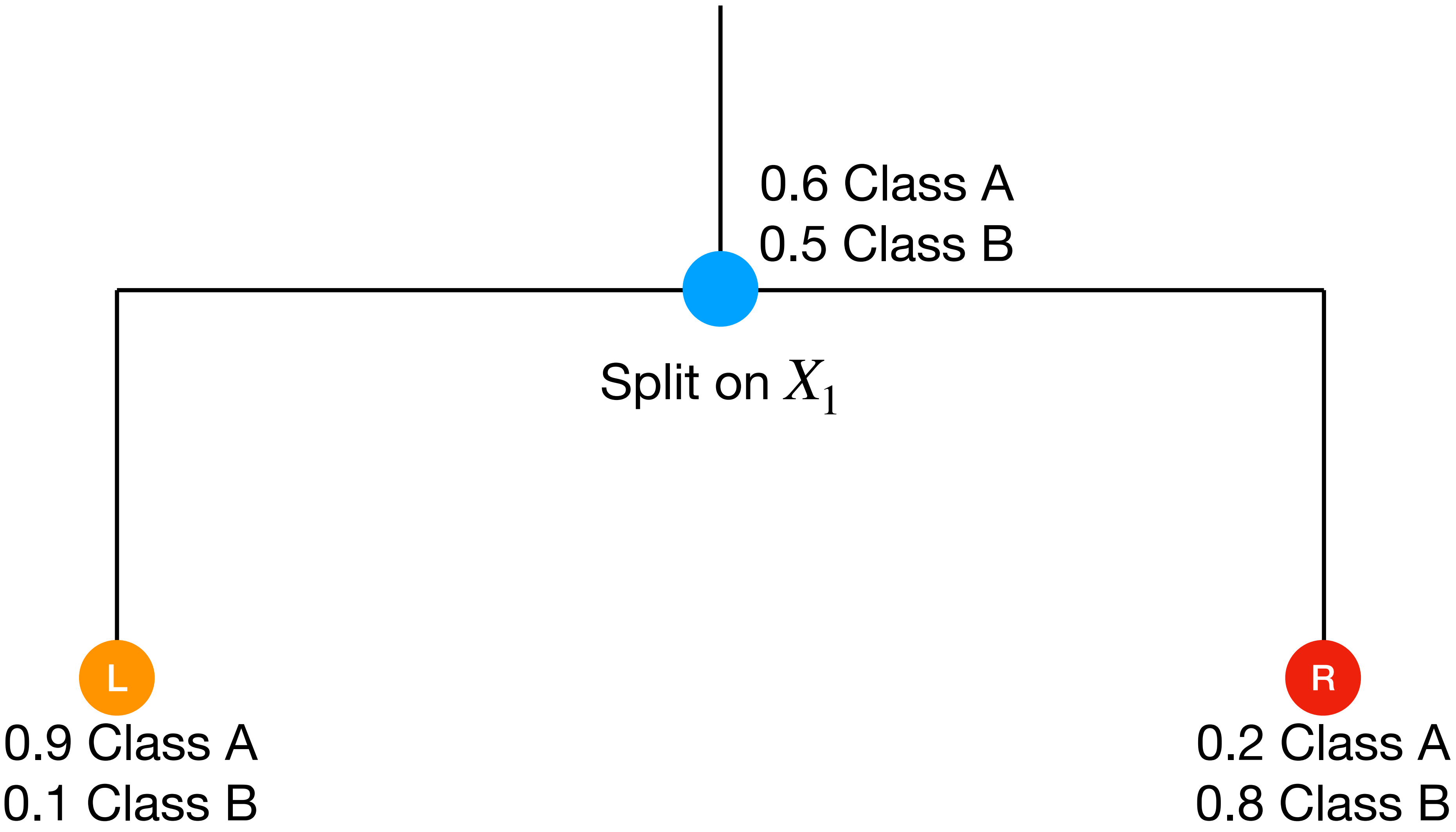
Algorithmic variable importance

- Tree-based methods: Gini/impurity-based variable importance measures
- Neural networks: Integrated Gradients, Saliency Maps, Grad-CAM, DeepLIFT
- SHAP values
- LIME variable importance

Population variable importance

- ANOVA decomposition in a low-dimensional parametric model
- Nonparametric extensions of ANOVA/R^2
- Causal variable importance

Impurity-based variable importance measures



0.6 Class A
0.5 Class B

Split on X_1

L

0.9 Class A
0.1 Class B

R

0.2 Class A
0.8 Class B

Importance of X_1 in this tree = Loss at ● - Loss at L - Loss at R

Importance of X_j in this tree = 0 for $j \neq 1$

Permutation-based variable importance

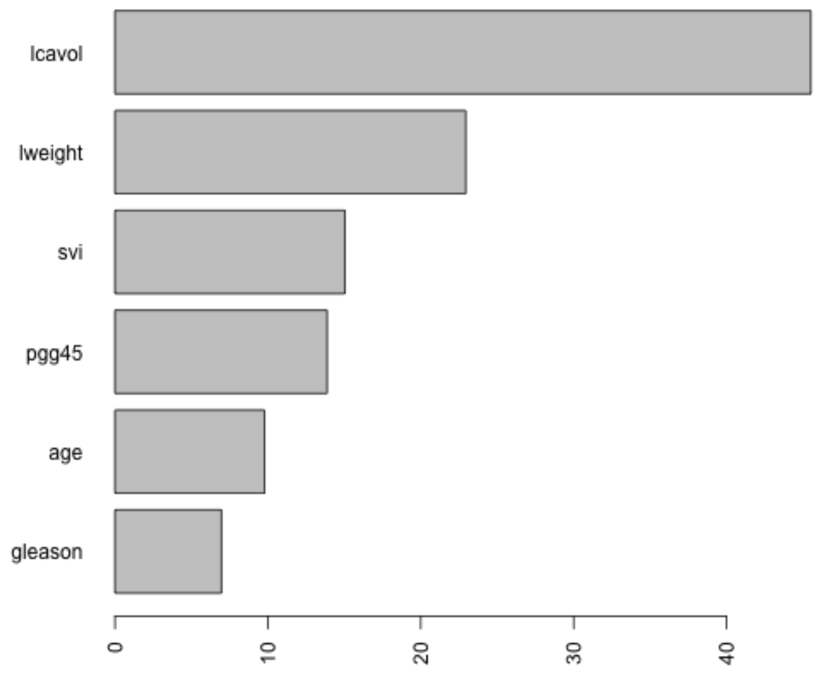
- Fit the random forest.
- Compute the OOB error.
- For j = 1,…,p:
    - Permute the values of the j-th variable. Compute the OOB error of the fitted model for the permuted dataset.
- Compare the OOB error for the original dataset versus the permuted dataset.

Is this variable important?

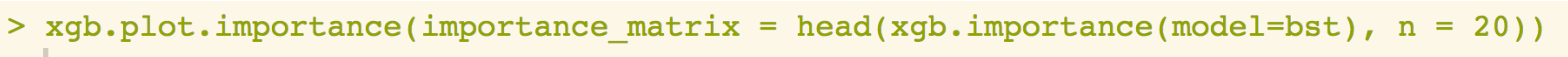
Q: Suppose I fit a random forest and it gets an AUC of 0.6. The plot of permutation variable importance is given below. Which of the following are true:

- **A**: The trained model assigned `lcavol` high importance.
- **B**: The variable `lcavol` is the most important variable for making accurate predictions.
- **C**: The trained model assigned `gleason` low importance.
- **D**: Suppose I was allowed to collect a larger training dataset for retraining the model, but I'm only allowed to collect data on 5 of 6 predictors. In this case, I should drop `gleason` from data collection because it has the lowest importance.

Permutation variable importance describes the importance of a variable to the ***fitted model****, not the importance of a variable in the* ***best possible model*** *for the target population.*



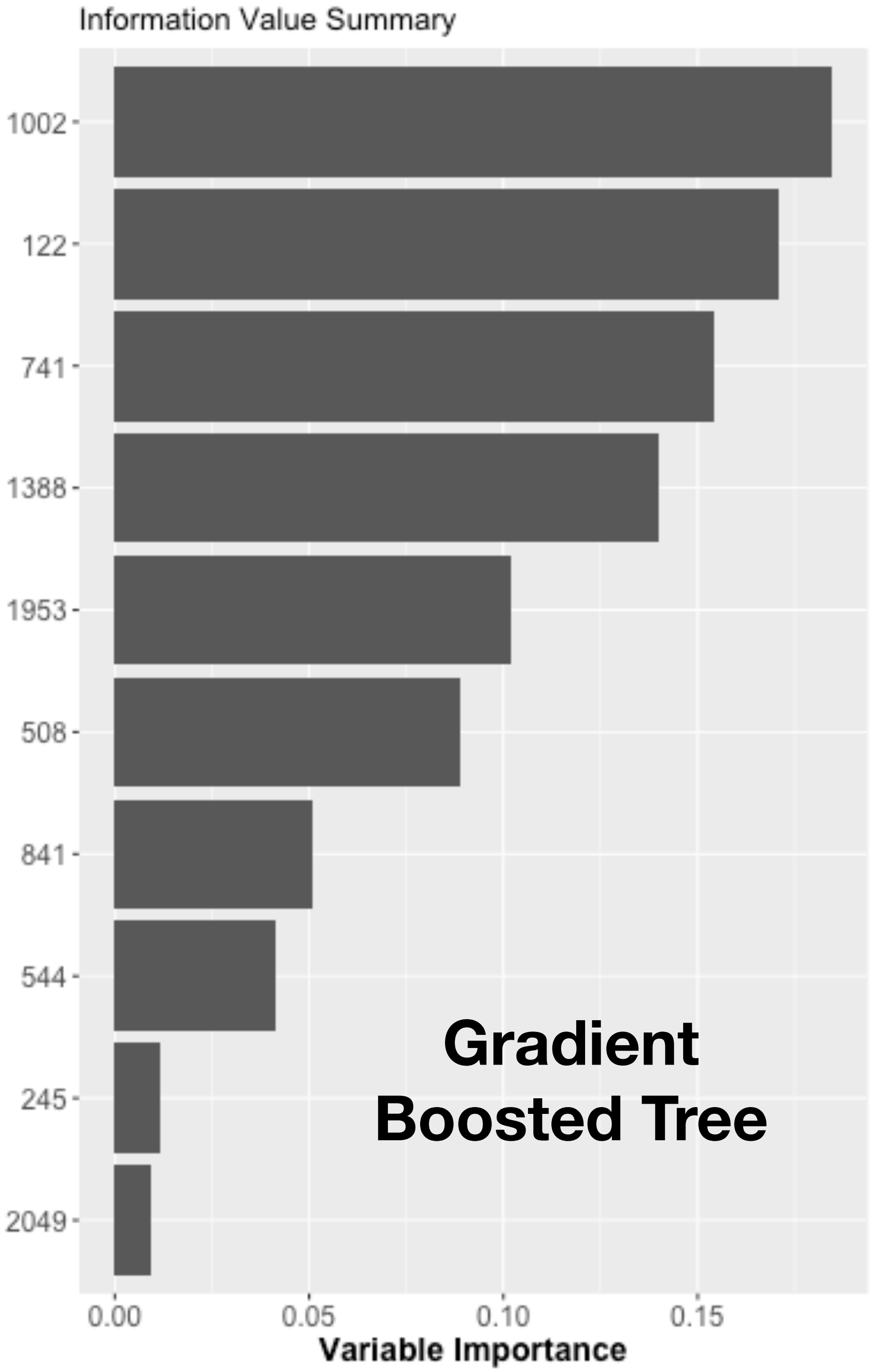
Example: Variable importance



```
> xgb.plot.importance(importance_matrix = head(xgb.importance(model=bst), n = 20))
```

- Let’s now try to interpret the gradient boosted tree we fit to the Khan gene expression dataset.

- Here we show the top 10 most important genes according to the variable importance function from xgboost.

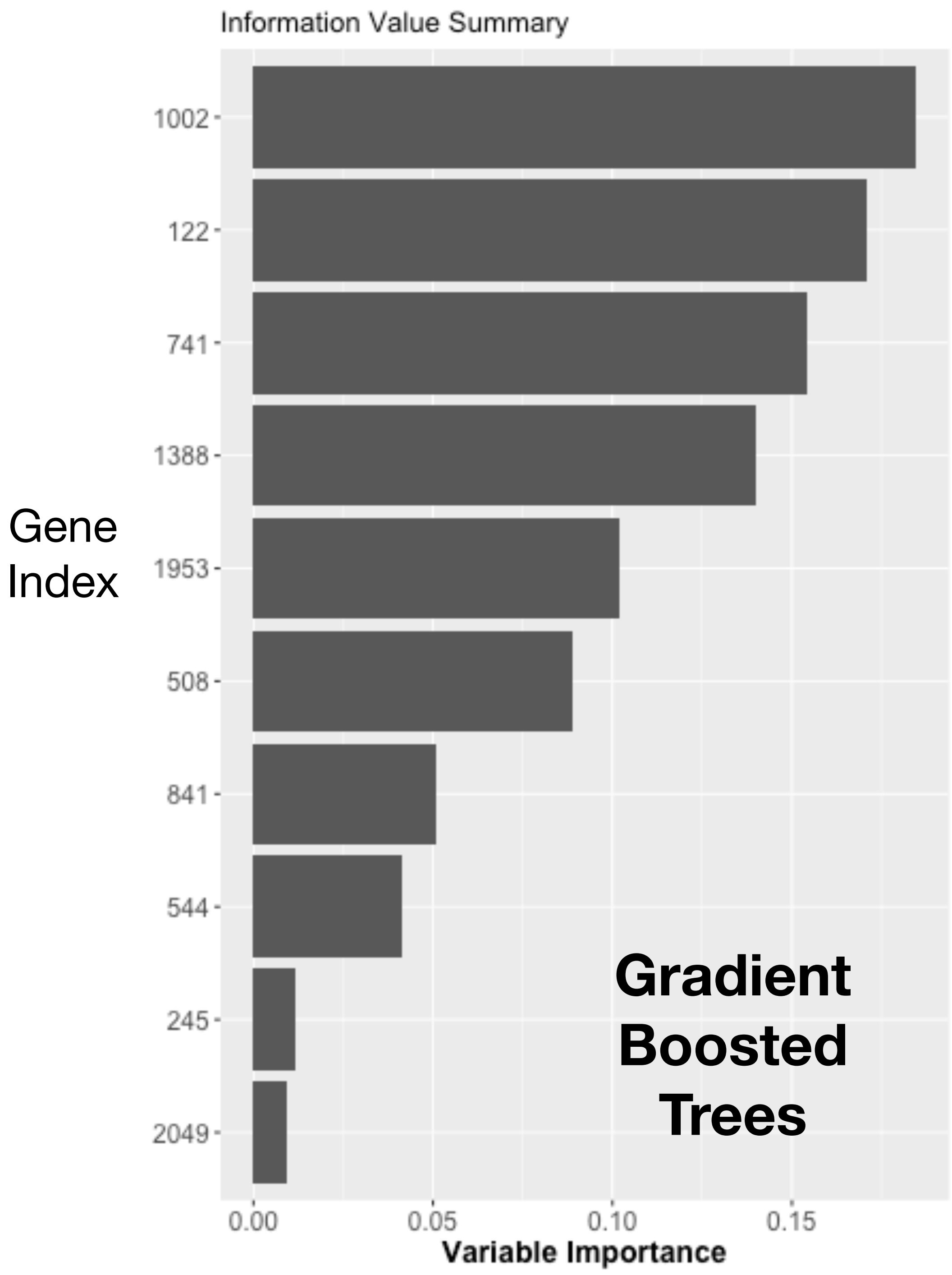


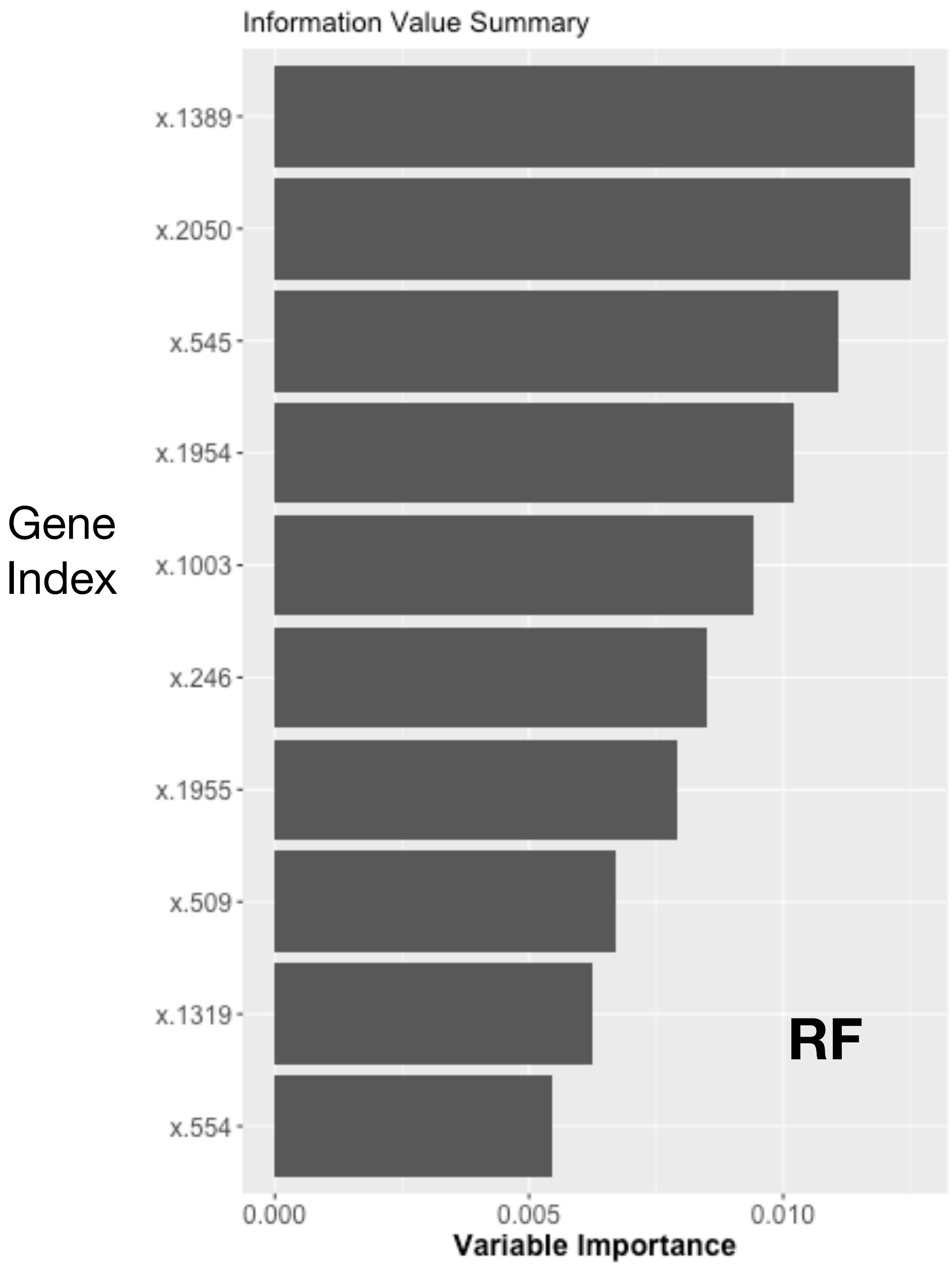
Gene Index

Gradient Boosted Tree

Example: Variable importance

- These variable importance estimates are model specific, so be careful when you interpret them!





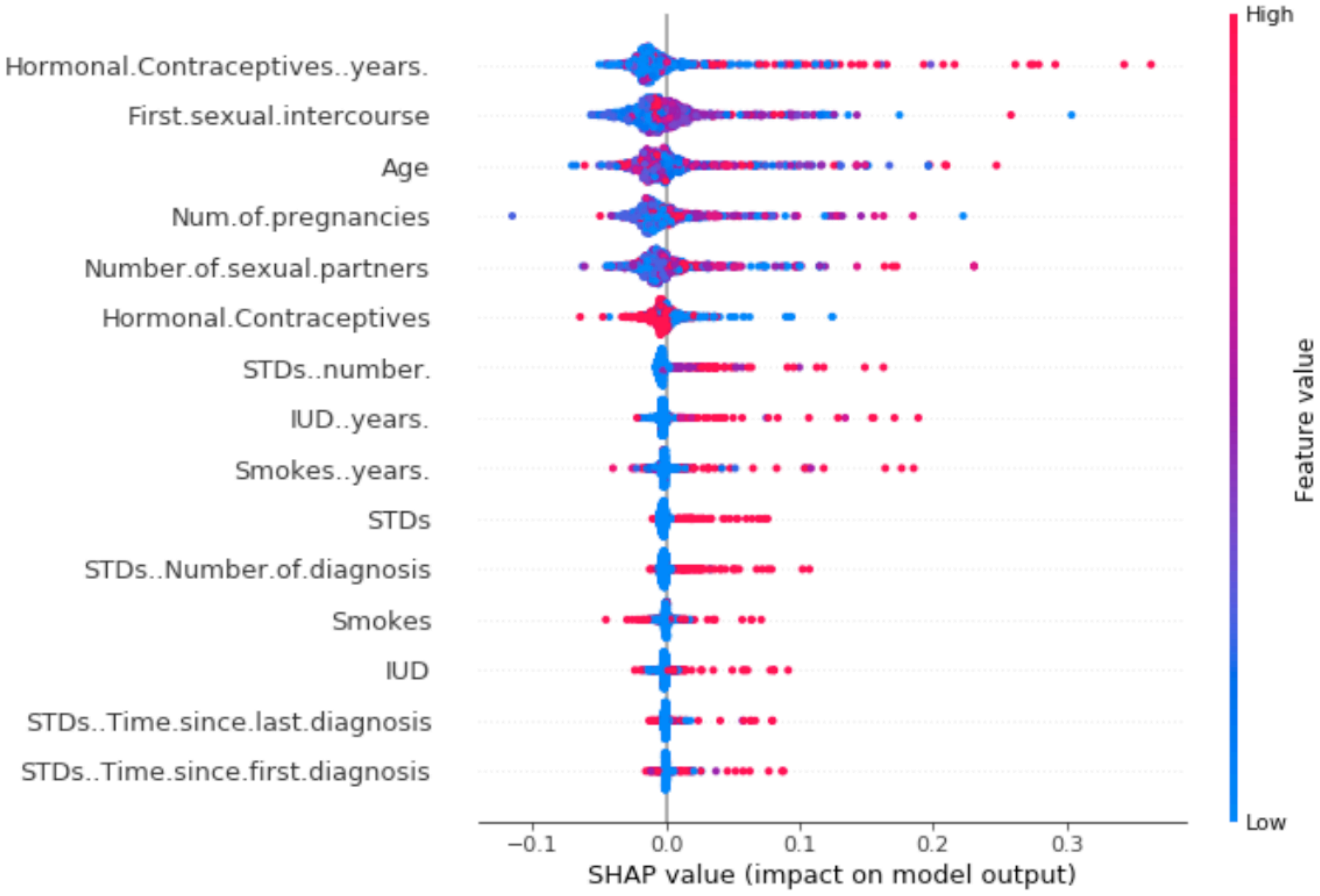
Why do the variable importance measures differ?

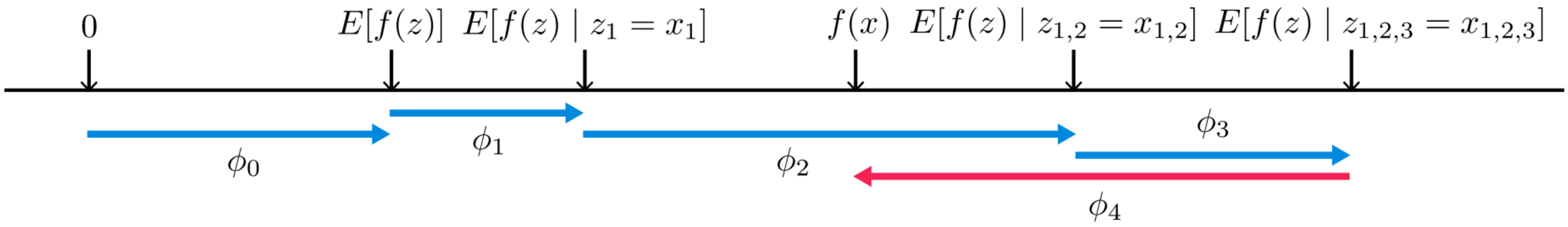
- Consider this example: Suppose there are two highly predictive variables X_1, X_2 that are also highly correlated. Then…
  - Gradient boosted trees will just choose one at random and ignore the other variable.
  - Random forests will choose both of them with similar probabilities.

SHAP values

- Shapley Additive exPlanation (SHAP) values summarize the variable importance **for a given prediction from a given model.**

- SHAP values measure how much the prediction changes when a feature is removed. Because there are many possible orders for removing the features, it averages over all possible orders.





SHAP values: calculations

Feature set	Prediction
{}	0.5
{1}	0.5
{2}	0.7
{3}	0.4
{1,2}	0.8
{1,3}	0.45
{2,3}	0.65
{1,2,3}	0.75

To calculate the SHAP value for X_1:

1. Determine how much X_1 changed our prediction with respect to possible feature subsets:
    - $\hat{f}(X_1) - \hat{f}(\{\,\}) = 0$
    - $\hat{f}(X_1, X_2) - \hat{f}(X_2) = 0.1$
    - $\hat{f}(X_1, X_3) - \hat{f}(X_3) = 0.05$
    - $\hat{f}(X_1, X_2, X_3) - \hat{f}(X_2, X_3) = 0.1$
2. Take a weighted average (for a special choice of weights) to get the SHAP value of X_1

Note: *In a linear model, the SHAP value for variable X_i is equal to $\beta_i x_i$.*

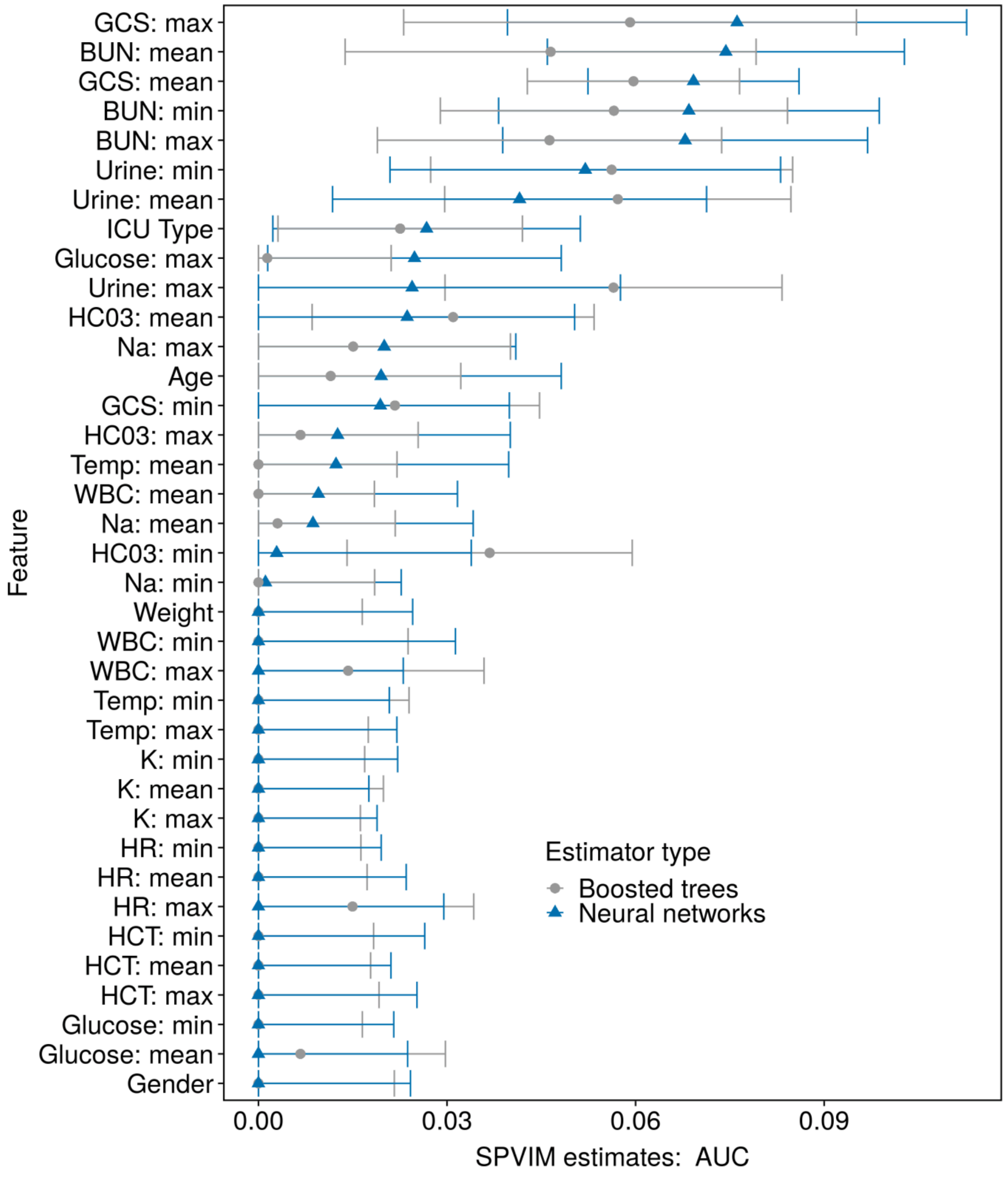
Population variable importance

- Warning: Getting in research territory!

- Not only is population variable importance helpful for answering scientific questions, population variable importance should be agnostic to which ML algorithm is used to estimate its value.

- Confidence intervals are only meaningful for population variable importance!

Shapley-based population variable importance measure



Williamson and Feng 2020

Outline

- Decision Trees
- Ensembling
    - Bagging
    - Random Forests
    - Gradient boosted trees
- Variable Importance

Outline

- Decision Trees
- Ensembling
  - Bagging
  - Random Forests
  - Gradient boosted trees
- Variable Importance
- **Bonus: Ensembling in general**

Ensemble methods

- Bagging and random forests are examples of ensemble methods.

- Ensemble methods construct a set of prediction models and take a weighted average to make a final prediction.

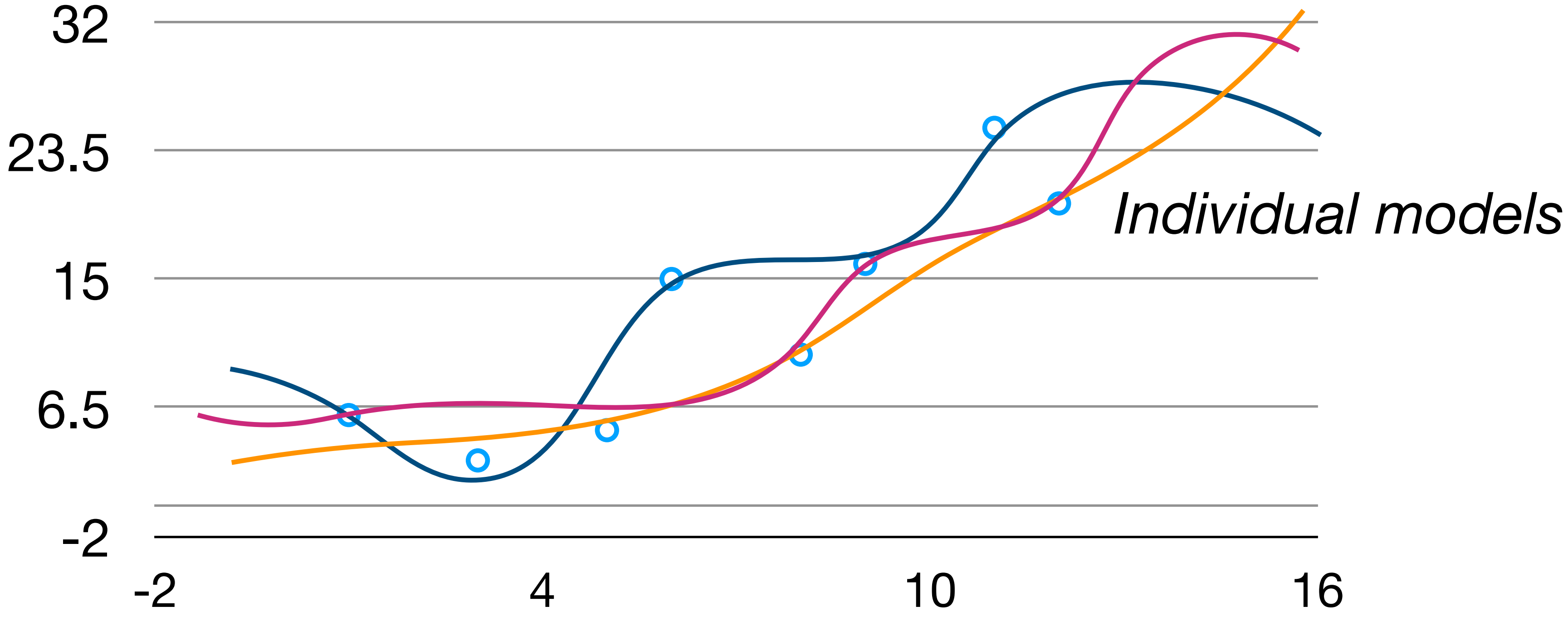
- “Wisdom of the crowds”: Ensembles tend to be more accurate than the individual prediction models.

Benefits of ensembling

1. Protection against overfitting:

   - When the underlying models have low bias but high variance, ensemble methods reduce the variance and selects a more appropriate bias-variance tradeoff.

   - We can also get a sense of model uncertainty.

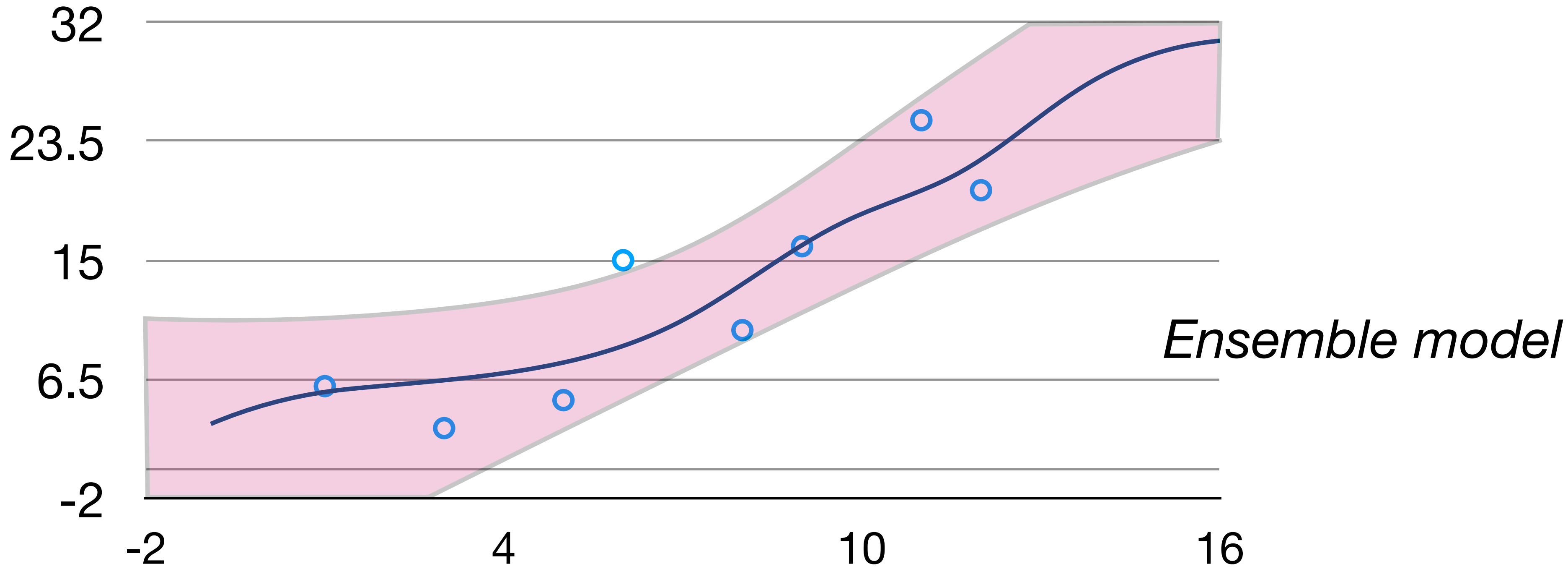


Benefits of ensembling

1. Protection against overfitting:

- When the underlying models have low bias but high variance, ensemble methods reduce the variance and selects a more appropriate bias-variance tradeoff.

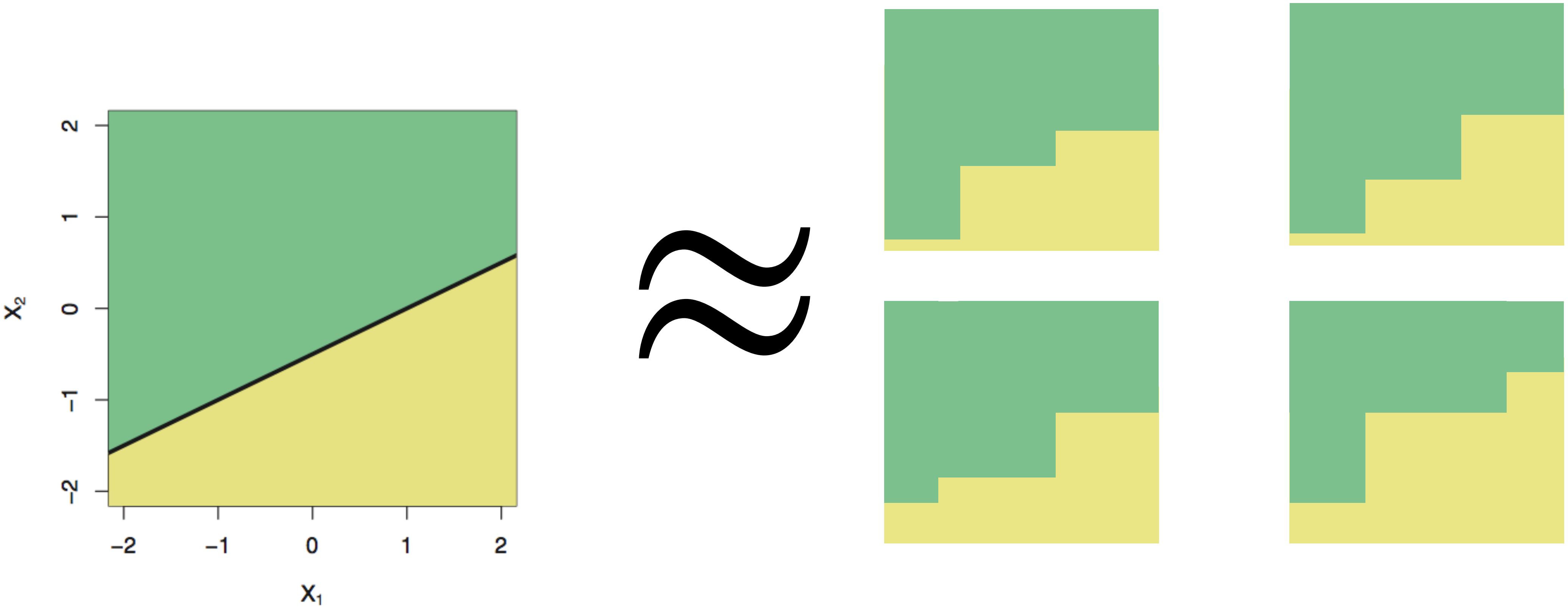
- We can also get a sense of model uncertainty.



Benefits of ensembling

2. The model is misspecified, but their average is a good approximation:

- Even if the individual models are highly restricted and the true data cannot be fully represented by the individual models, we can expand the space of functions that we can estimate by taking an average of the individual models.

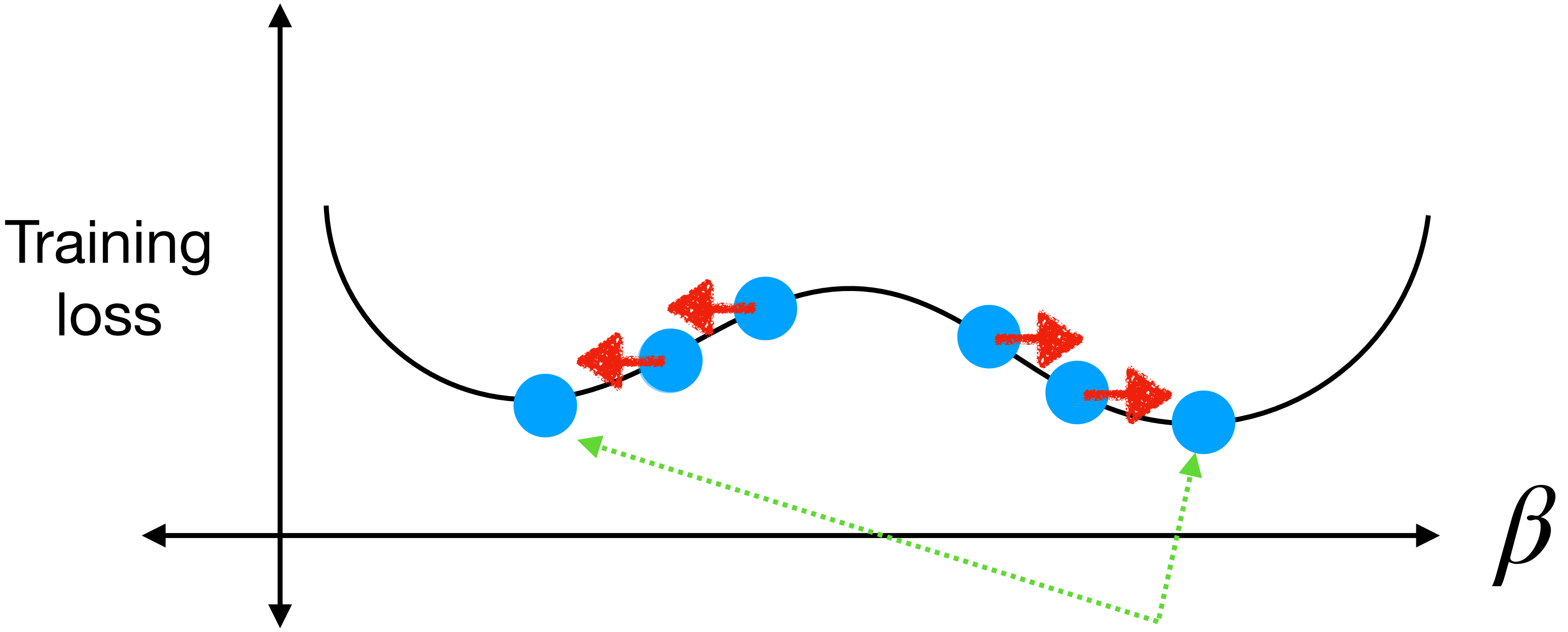


Benefits of ensembling

3. Ensembling can help us overcome computational difficulties:

- For ML algorithms like random forests and neural networks, the model is fit by attempting to solve a non-convex optimization problem.

- Because we are very likely to get stuck at a local optima, we should try multiple start points. For models that look equally good, taking an average can help.



Take the average prediction from these models