Tree pruning and bagging

STAT 4710

November 1, 2022

Where we are

✓ **Unit 1:** R for data mining

✓ **Unit 2:** Prediction fundamentals

✓ **Unit 3:** Regression-based methods

Unit 4: Tree-based methods

Unit 5: Deep learning

Lecture 1: Growing decision trees

Lecture 2: Tree pruning and bagging

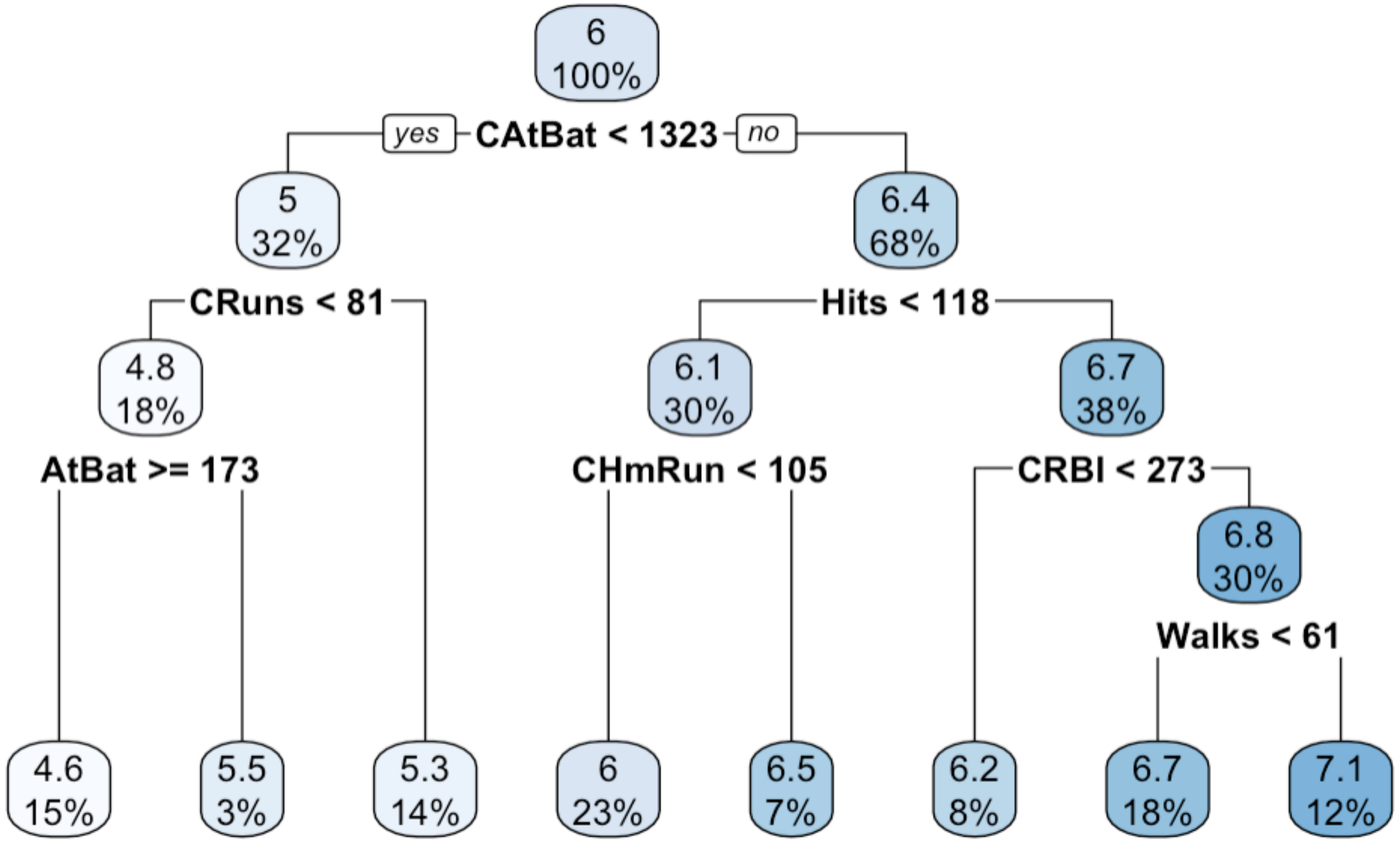
Lecture 3: Random forests

Lecture 4: Boosting

Lecture 5: Unit review and quiz in class

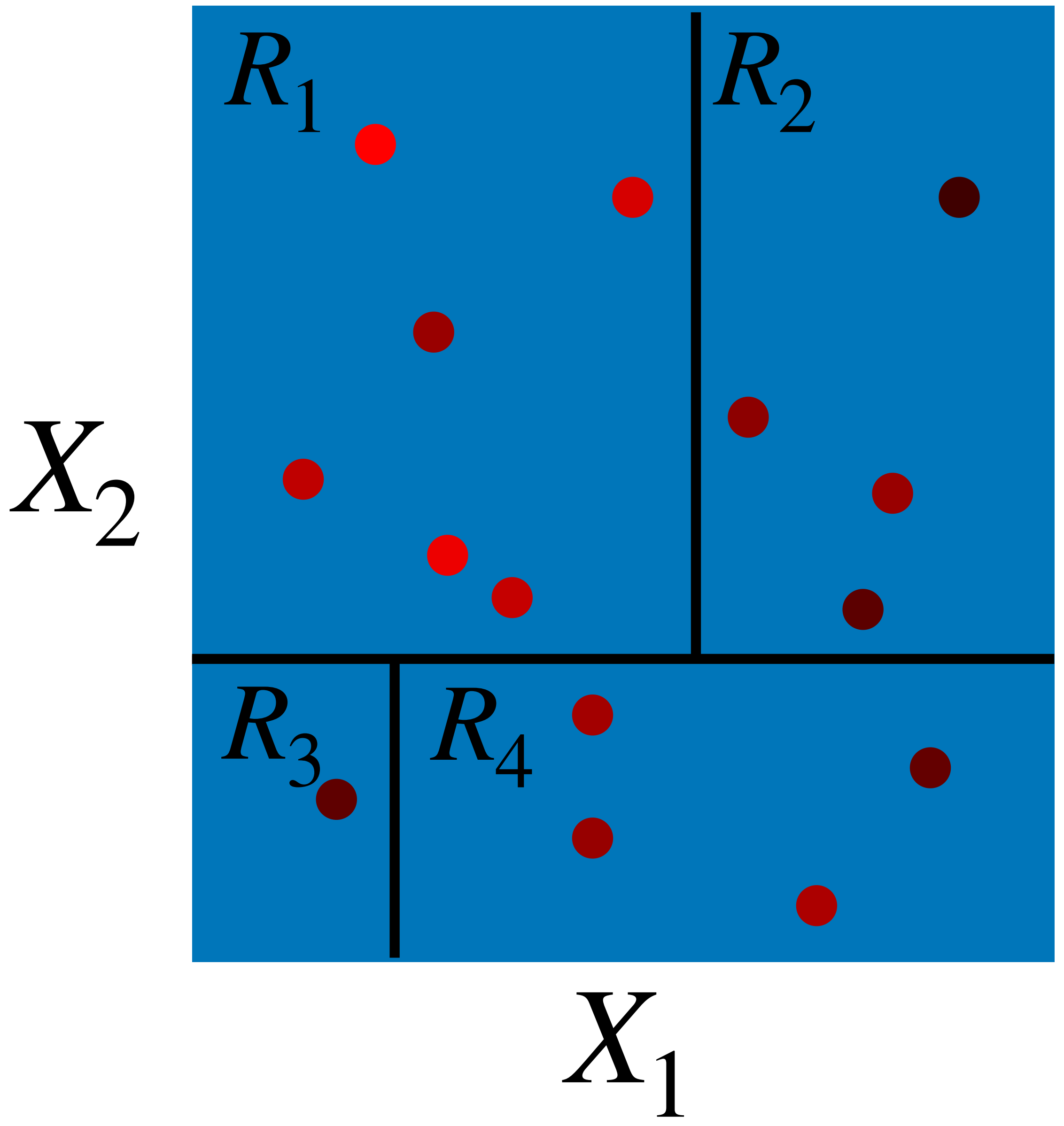
Recall: Decision trees

- Create a partition of feature space by recursively splitting on different features
- Regression and classification trees
- Terminal nodes in the tree correspond to the rectangles in the partition
- Predict a single number (category) for each terminal node in a regression (classification) tree



Clarifications from last time

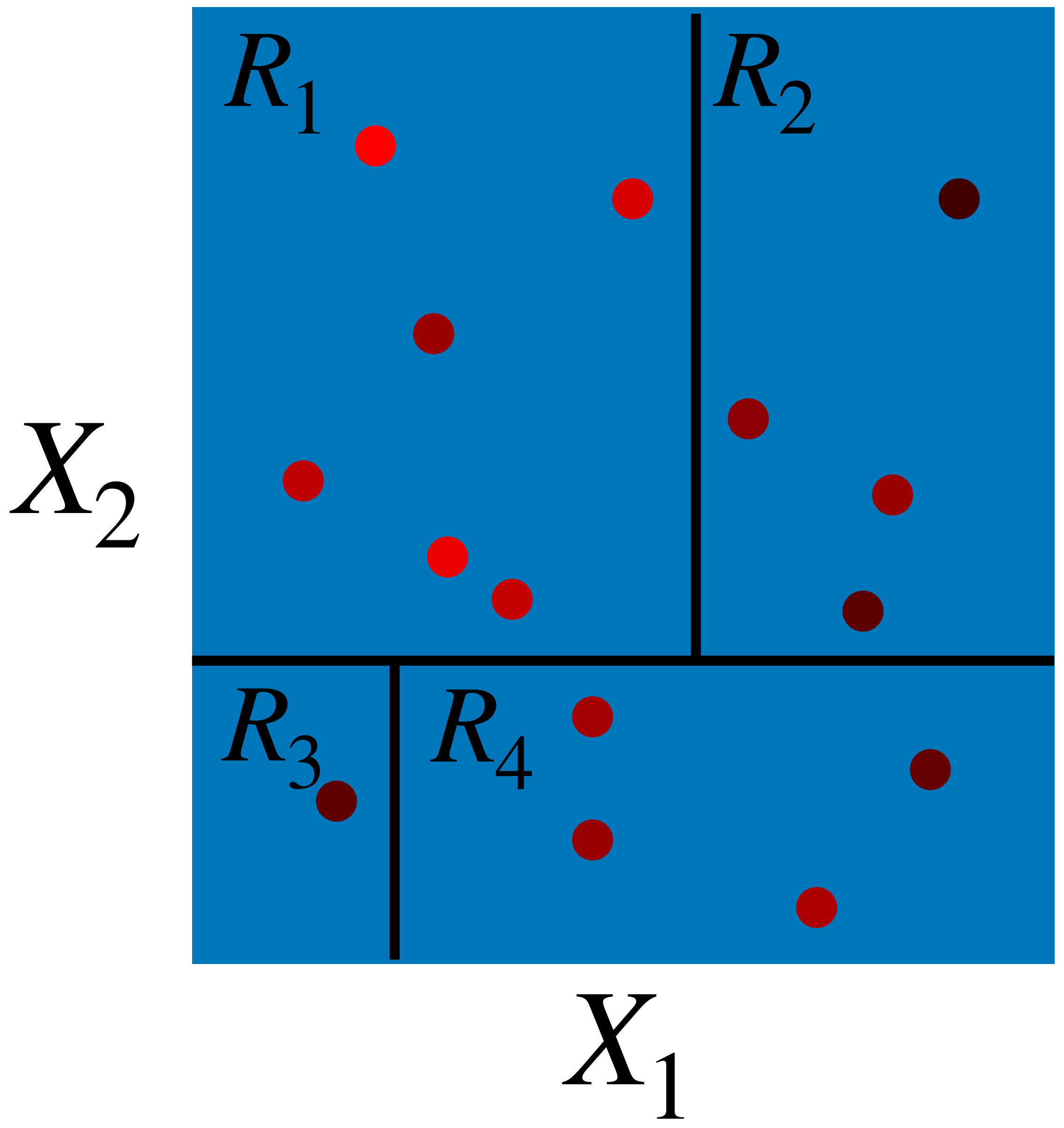
Regression tree



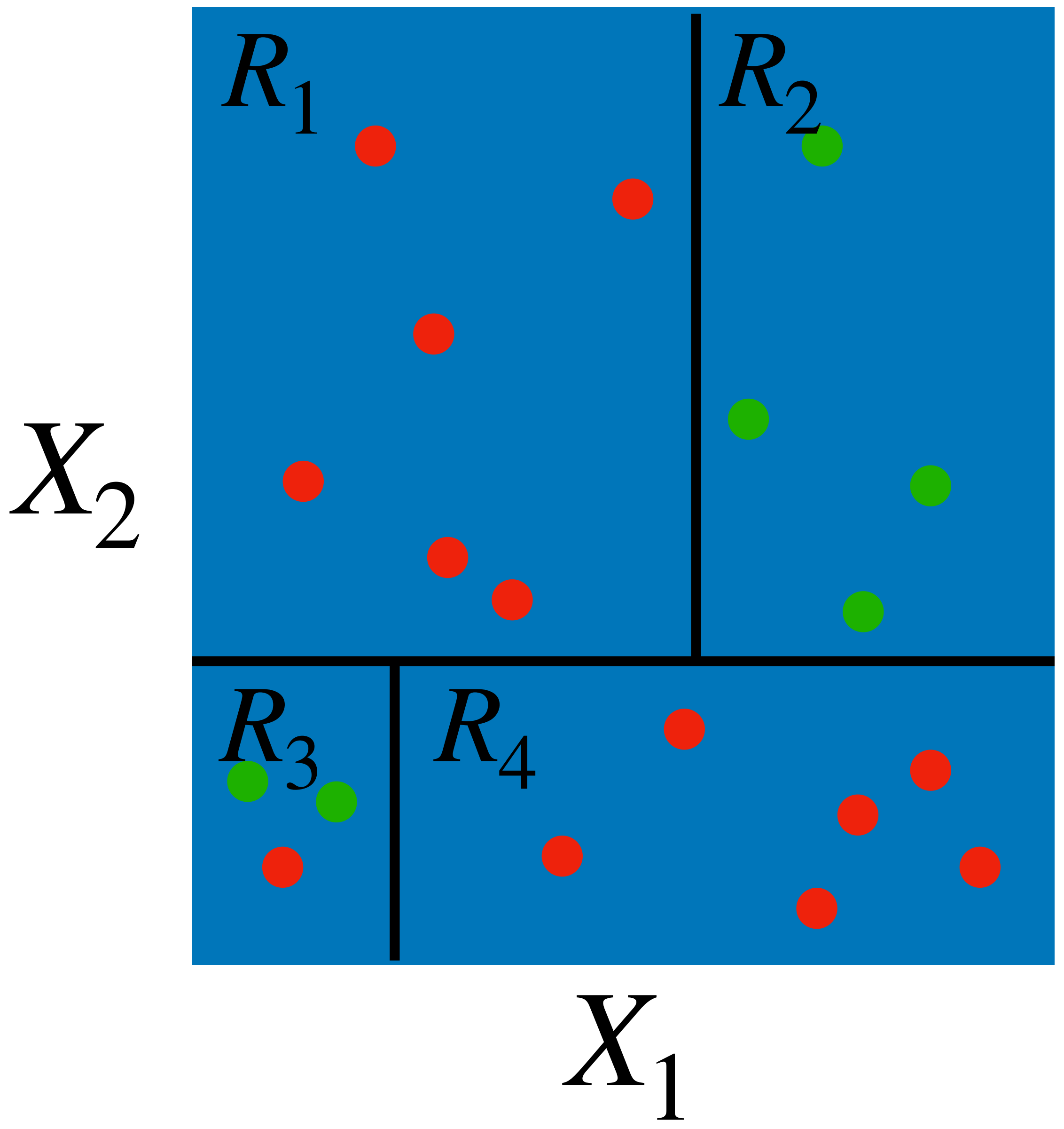
Regression: Total RSS $= \sum_{i=1}^{n} (Y_i - \widehat{Y}_i)^2 = \sum_{i \in R_1} (Y_i - \widehat{Y}_i)^2 + \sum_{i \in R_2} (Y_i - \widehat{Y}_i)^2 + \sum_{i \in R_3} (Y_i - \widehat{Y}_i)^2 + \sum_{i \in R_4} (Y_i - \widehat{Y}_i)^2$

Clarifications from last time

Regression tree



Classification tree

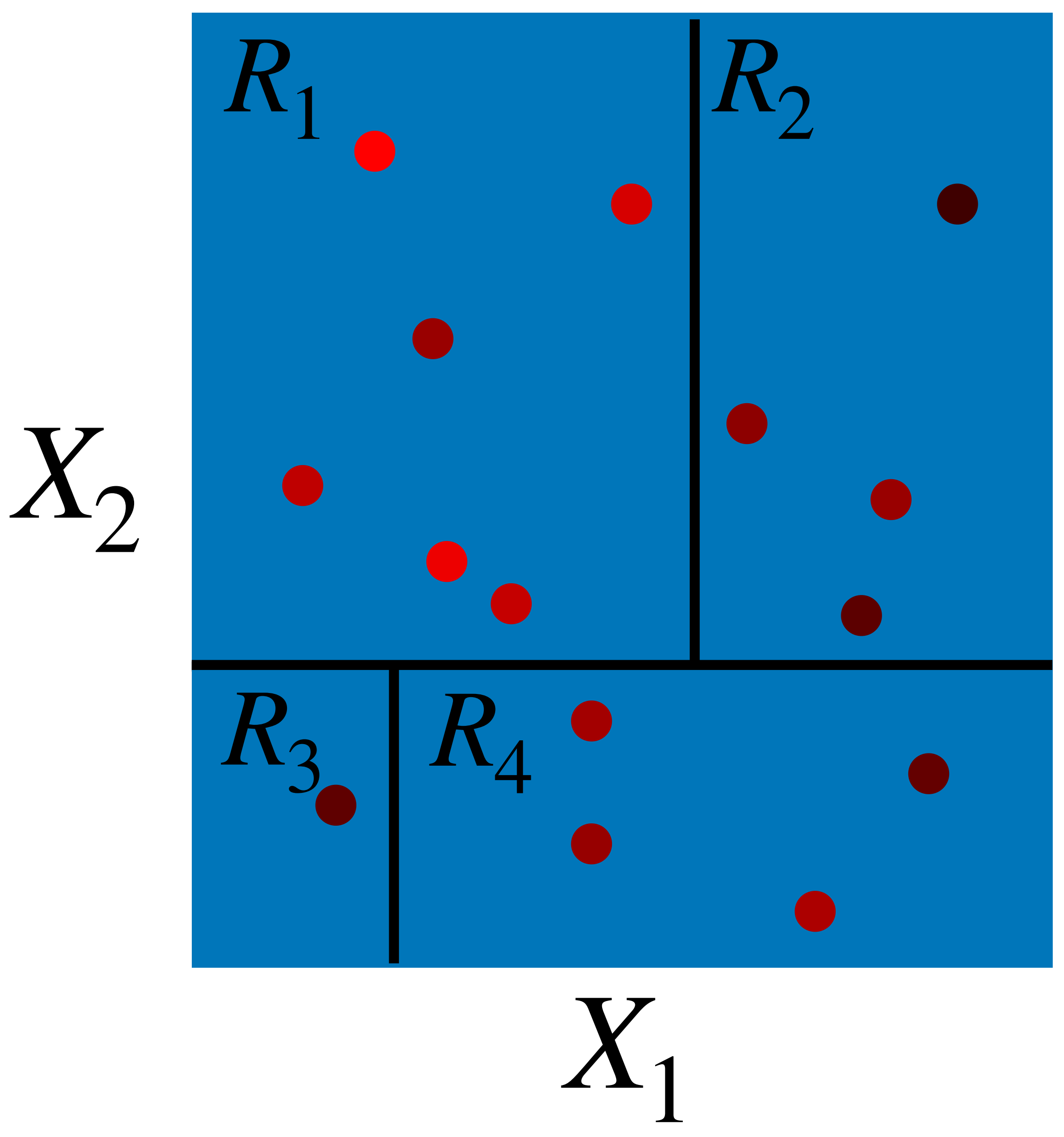


Regression: Total RSS $= \sum_{i=1}^{n} (Y_i - \widehat{Y}_i)^2 = \sum_{i \in R_1} (Y_i - \widehat{Y}_i)^2 + \sum_{i \in R_2} (Y_i - \widehat{Y}_i)^2 + \sum_{i \in R_3} (Y_i - \widehat{Y}_i)^2 + \sum_{i \in R_4} (Y_i - \widehat{Y}_i)^2$

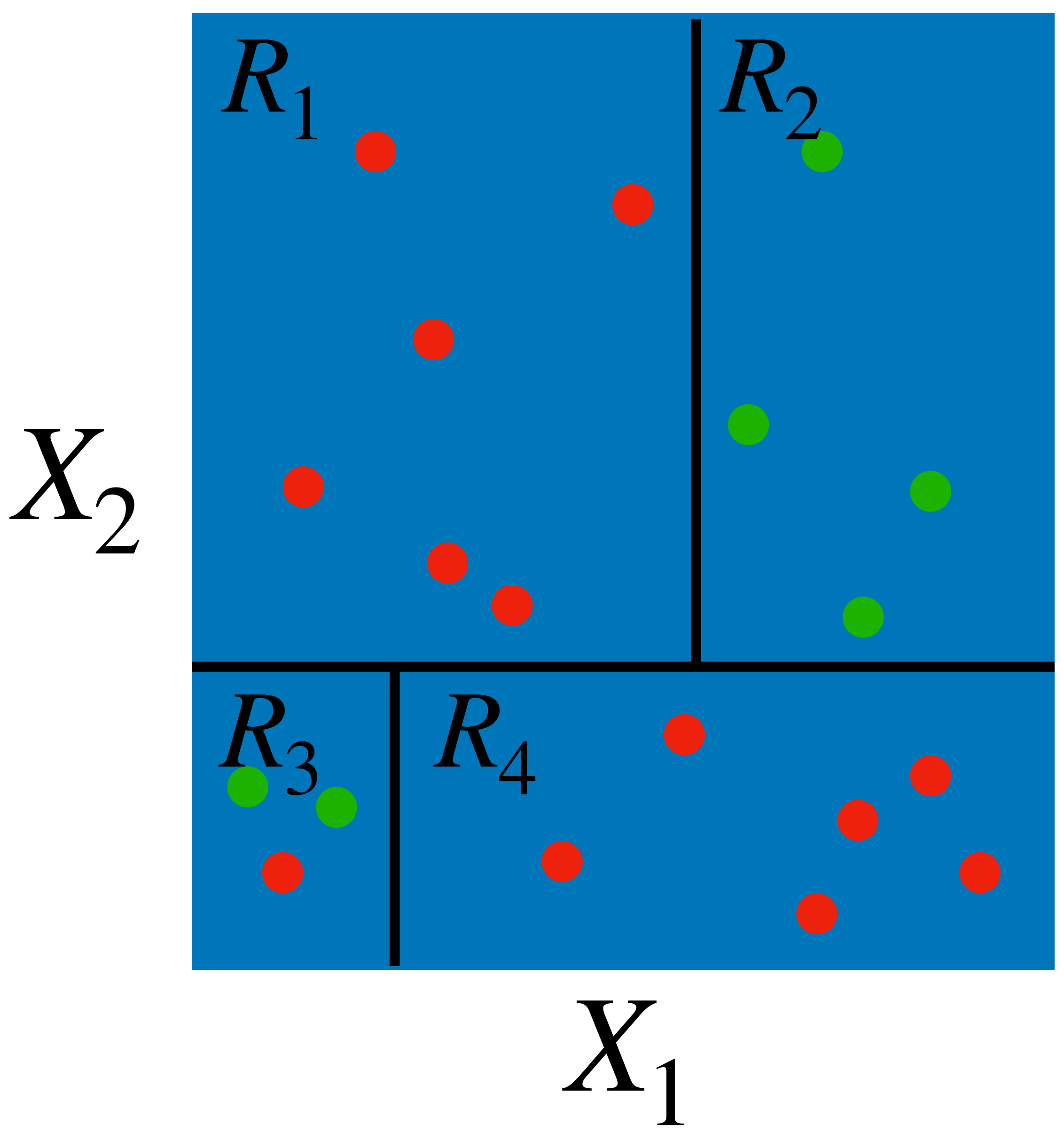
Classification: Total Gini $= n_1 \cdot 2\widehat{p}_1(1 - \widehat{p}_1) + n_2 \cdot 2\widehat{p}_2(1 - \widehat{p}_2) + n_3 \cdot 2\widehat{p}_3(1 - \widehat{p}_3) + n_4 \cdot 2\widehat{p}_4(1 - \widehat{p}_4)$

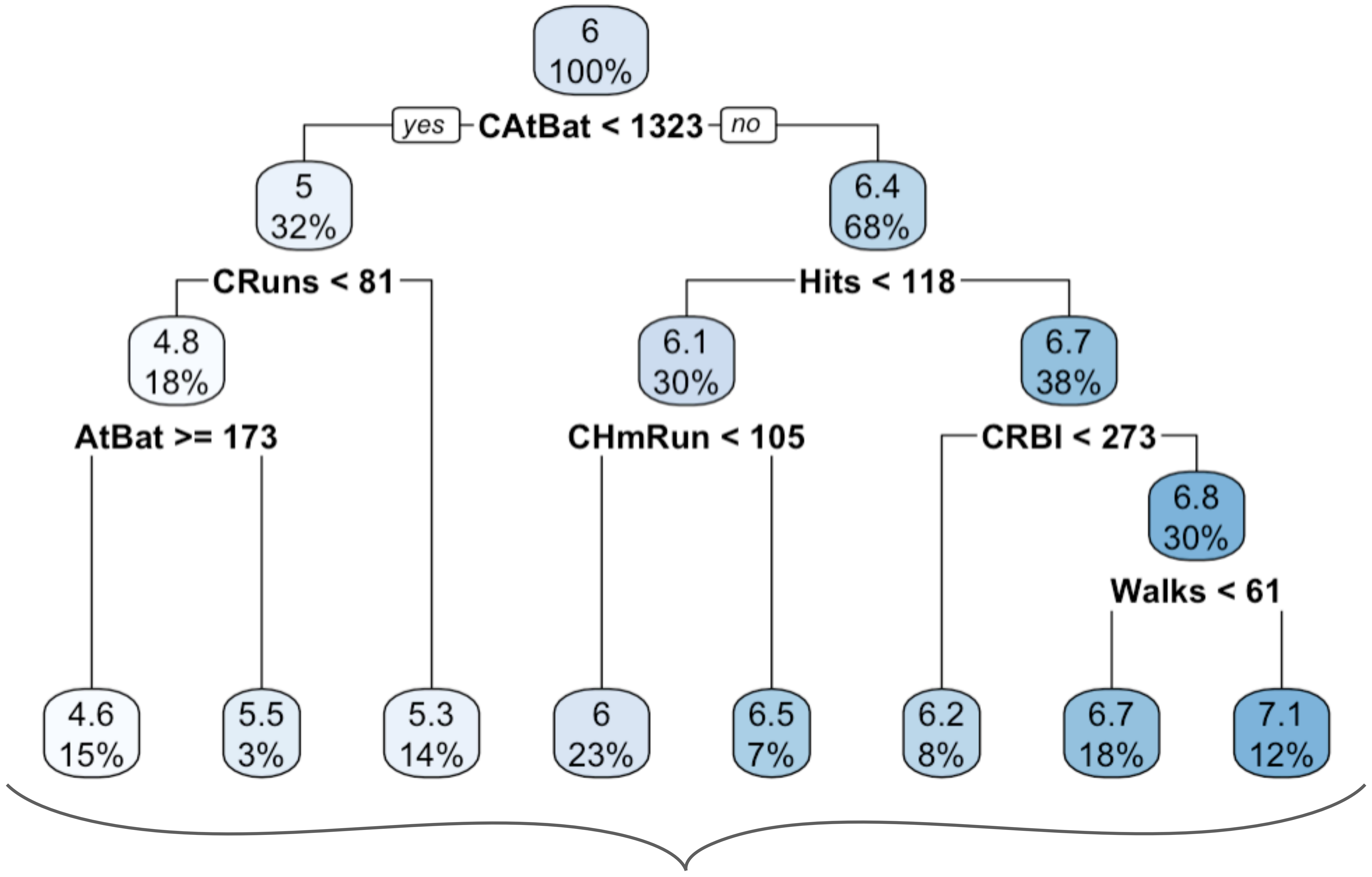
Clarifications from last time

Regression tree



Classification tree



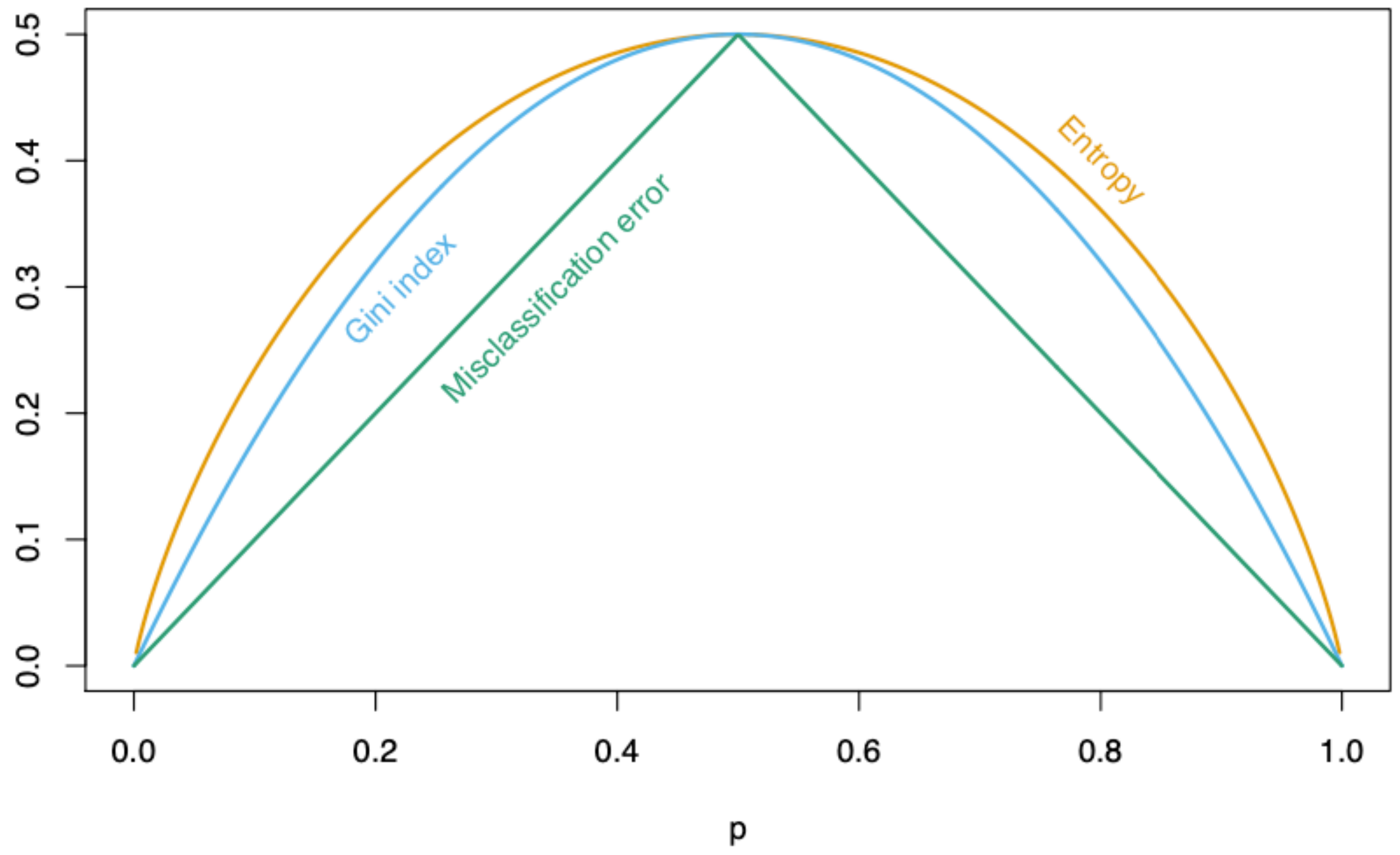


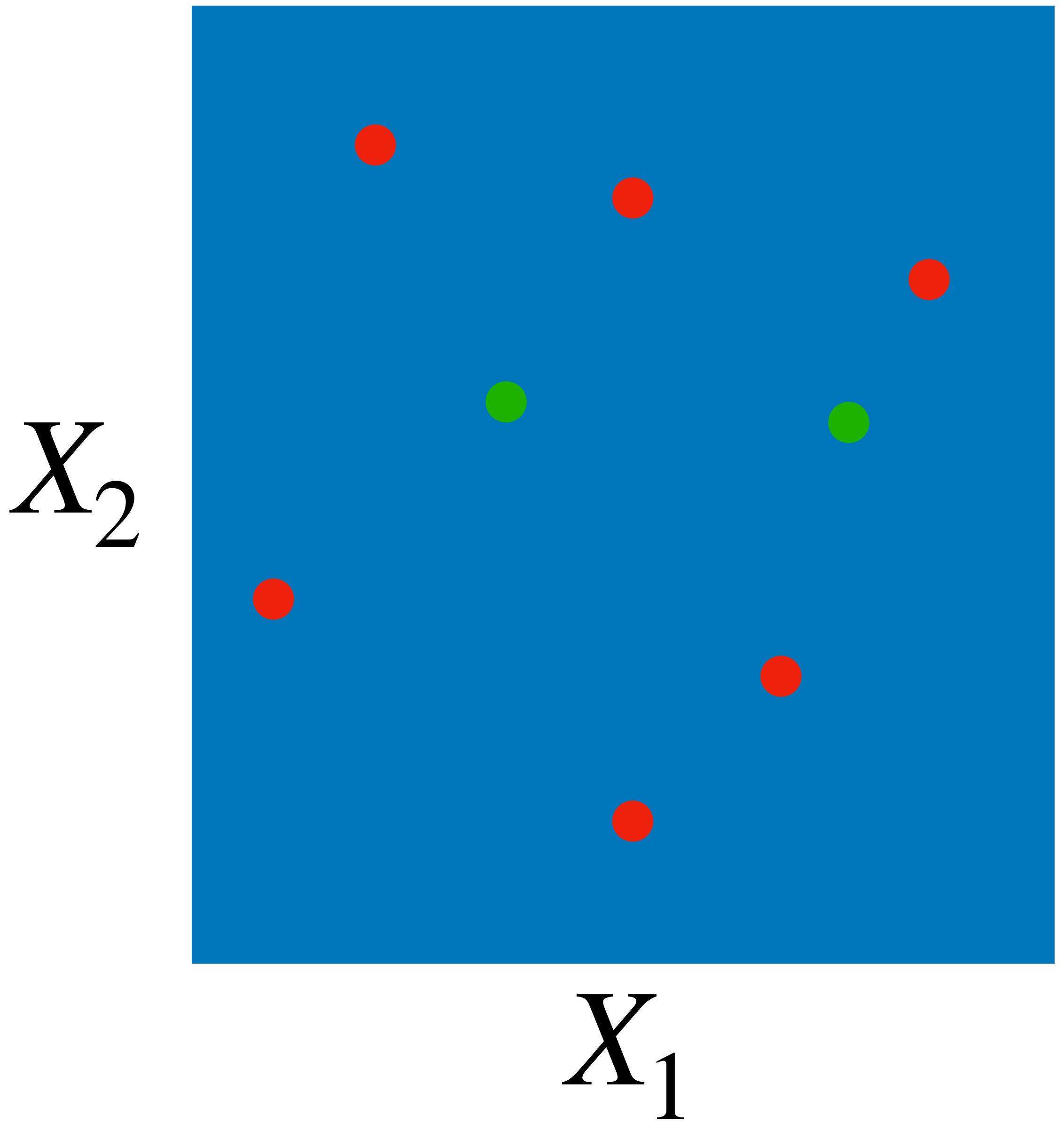
Leaf nodes or terminal nodes (equivalent)

Regression: Total RSS $= \sum_{i=1}^{n} (Y_i - \widehat{Y}_i)^2 = \sum_{i \in R_1} (Y_i - \widehat{Y}_i)^2 + \sum_{i \in R_2} (Y_i - \widehat{Y}_i)^2 + \sum_{i \in R_3} (Y_i - \widehat{Y}_i)^2 + \sum_{i \in R_4} (Y_i - \widehat{Y}_i)^2$

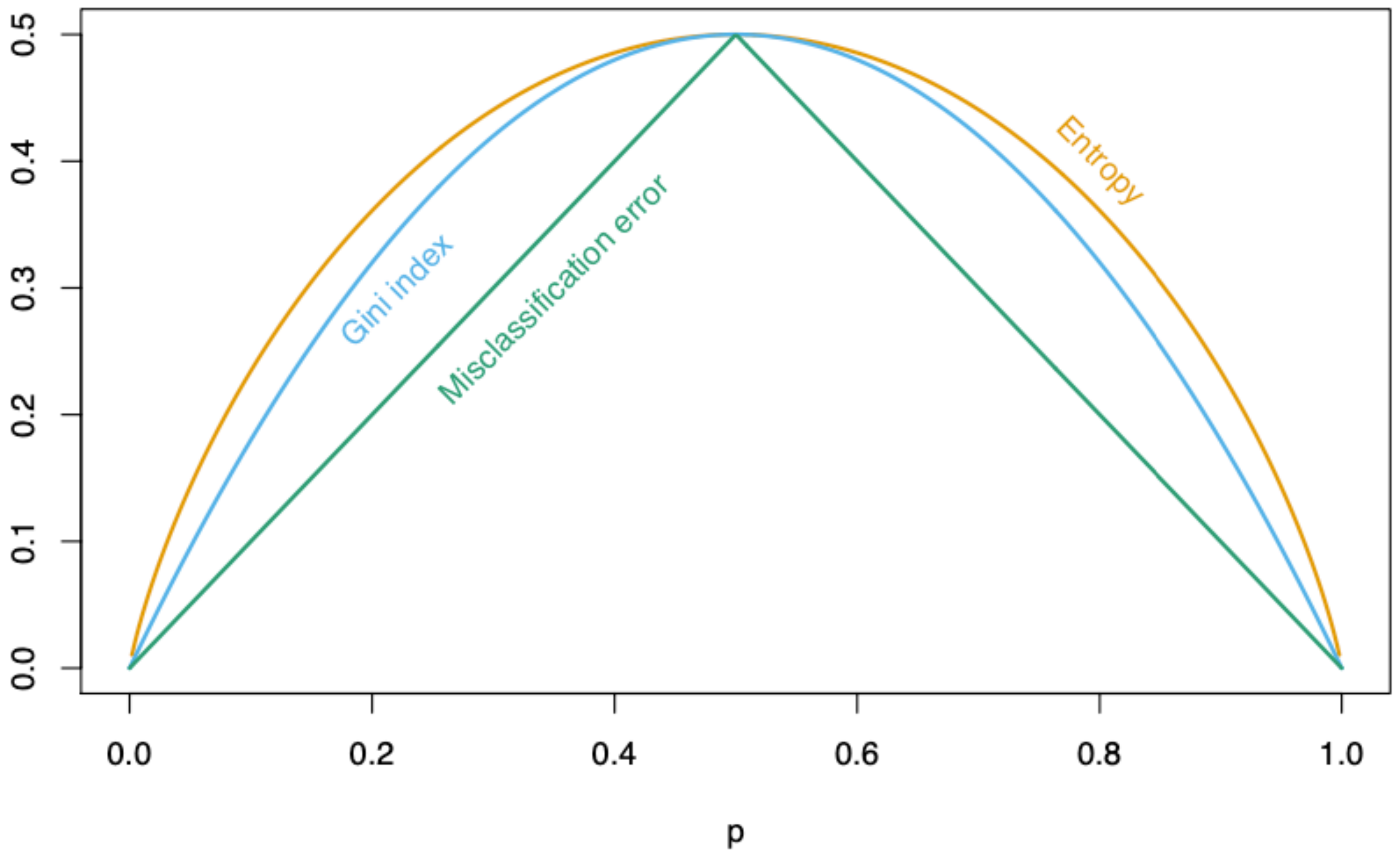
Classification: Total Gini $= n_1 \cdot 2\widehat{p}_1(1 - \widehat{p}_1) + n_2 \cdot 2\widehat{p}_2(1 - \widehat{p}_2) + n_3 \cdot 2\widehat{p}_3(1 - \widehat{p}_3) + n_4 \cdot 2\widehat{p}_4(1 - \widehat{p}_4)$

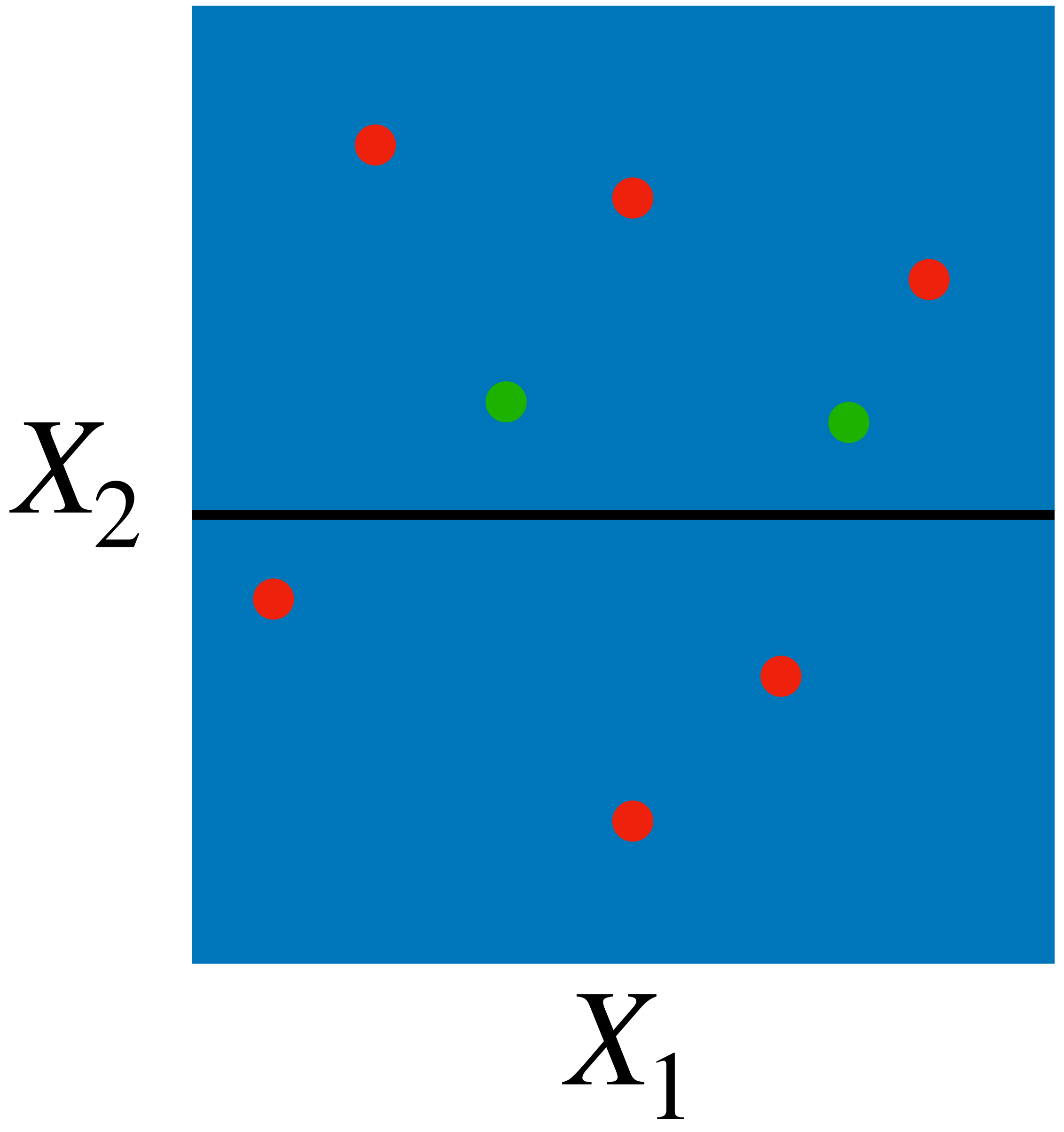
Clarifications from last time: Why Gini index?



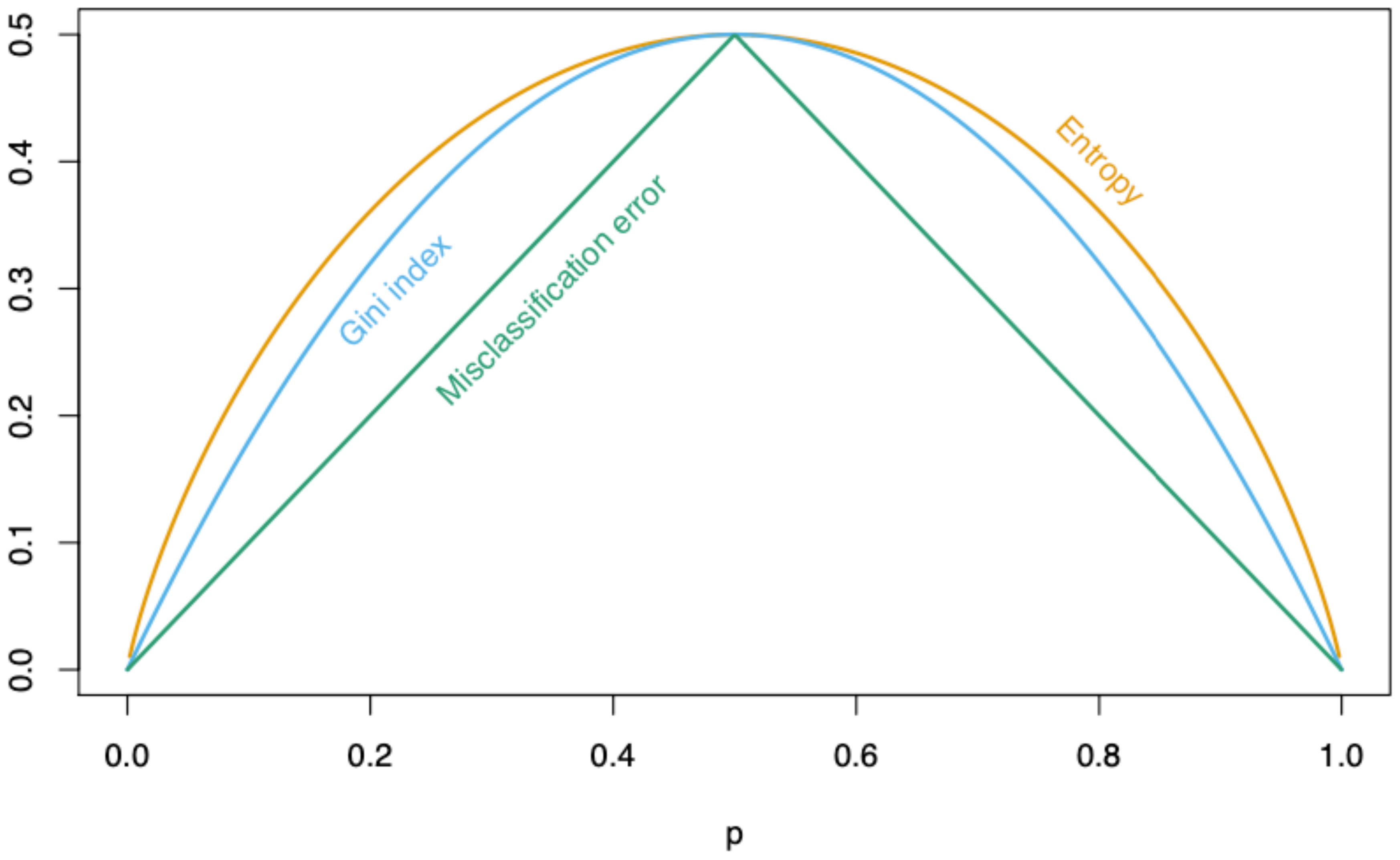


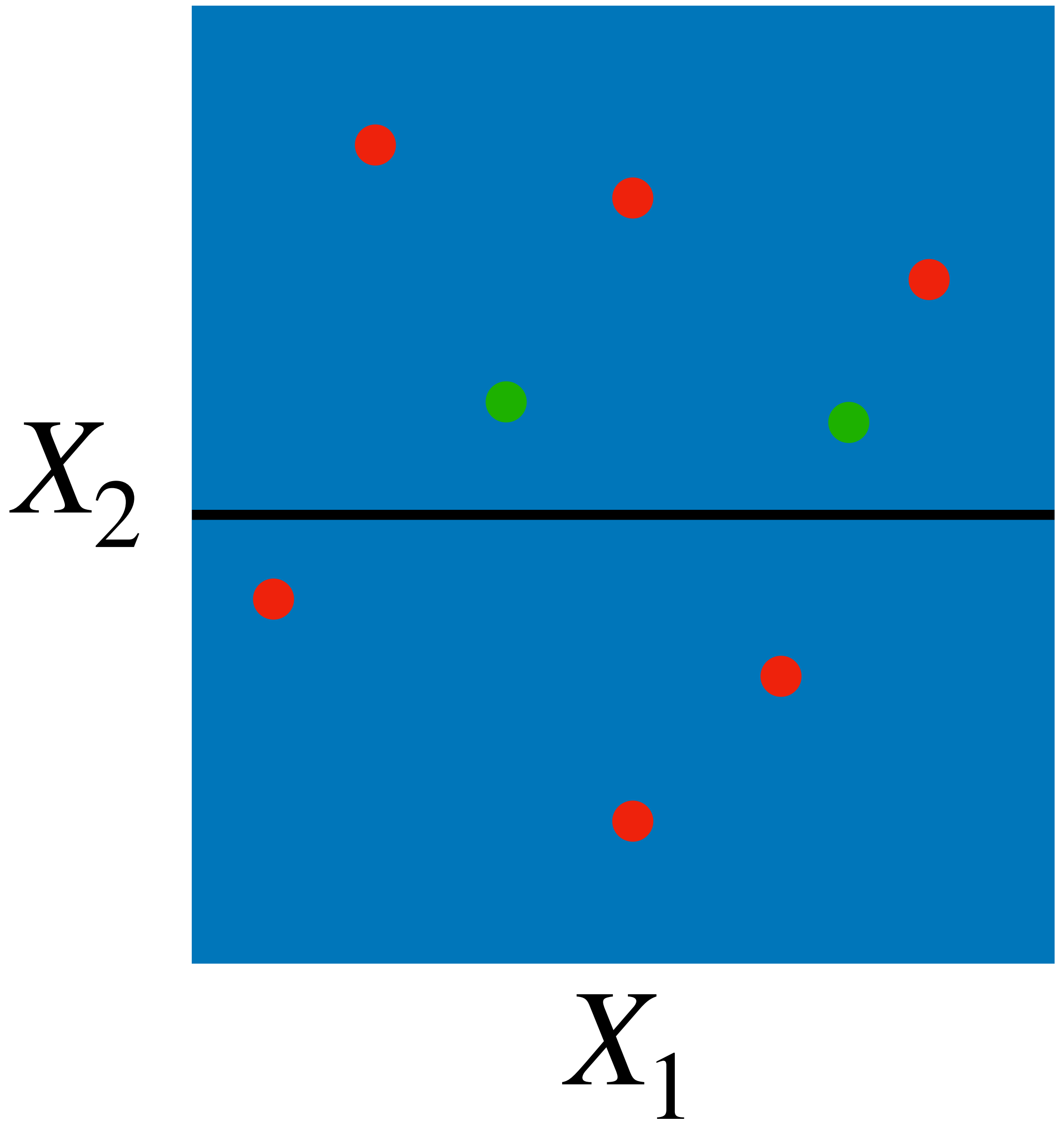
Clarifications from last time: Why Gini index?





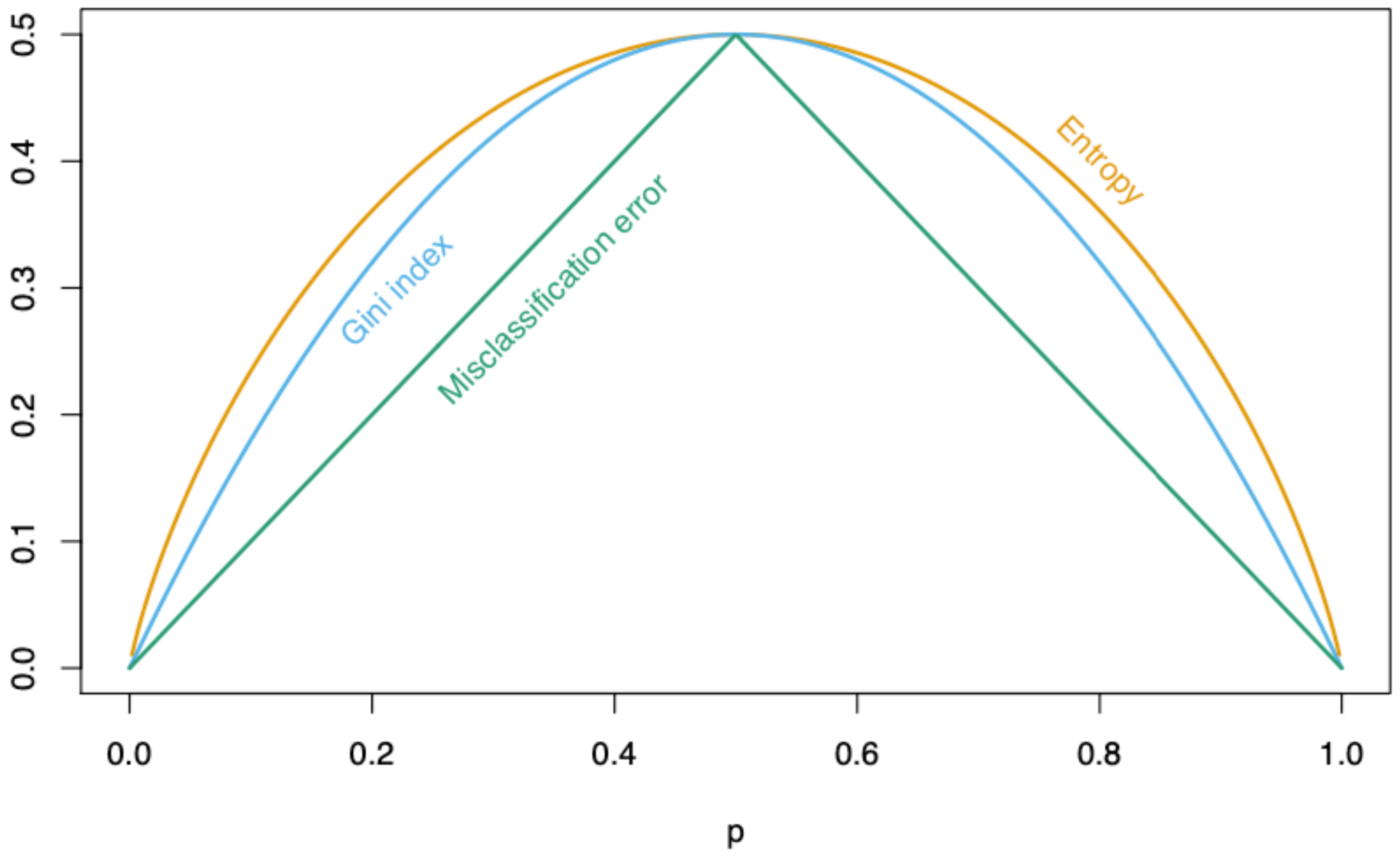
Clarifications from last time: Why Gini index?

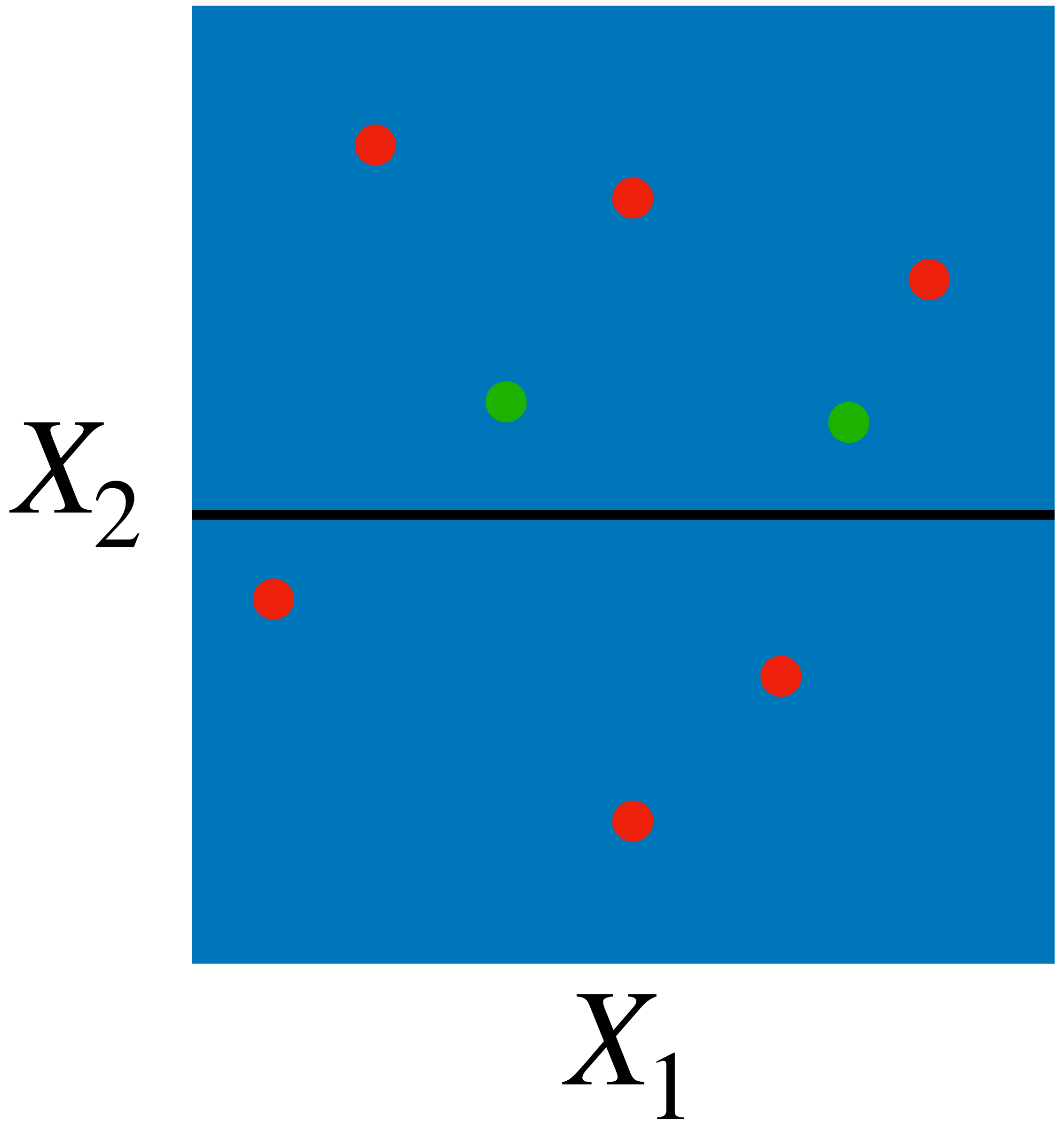




Tree	Misclass. error

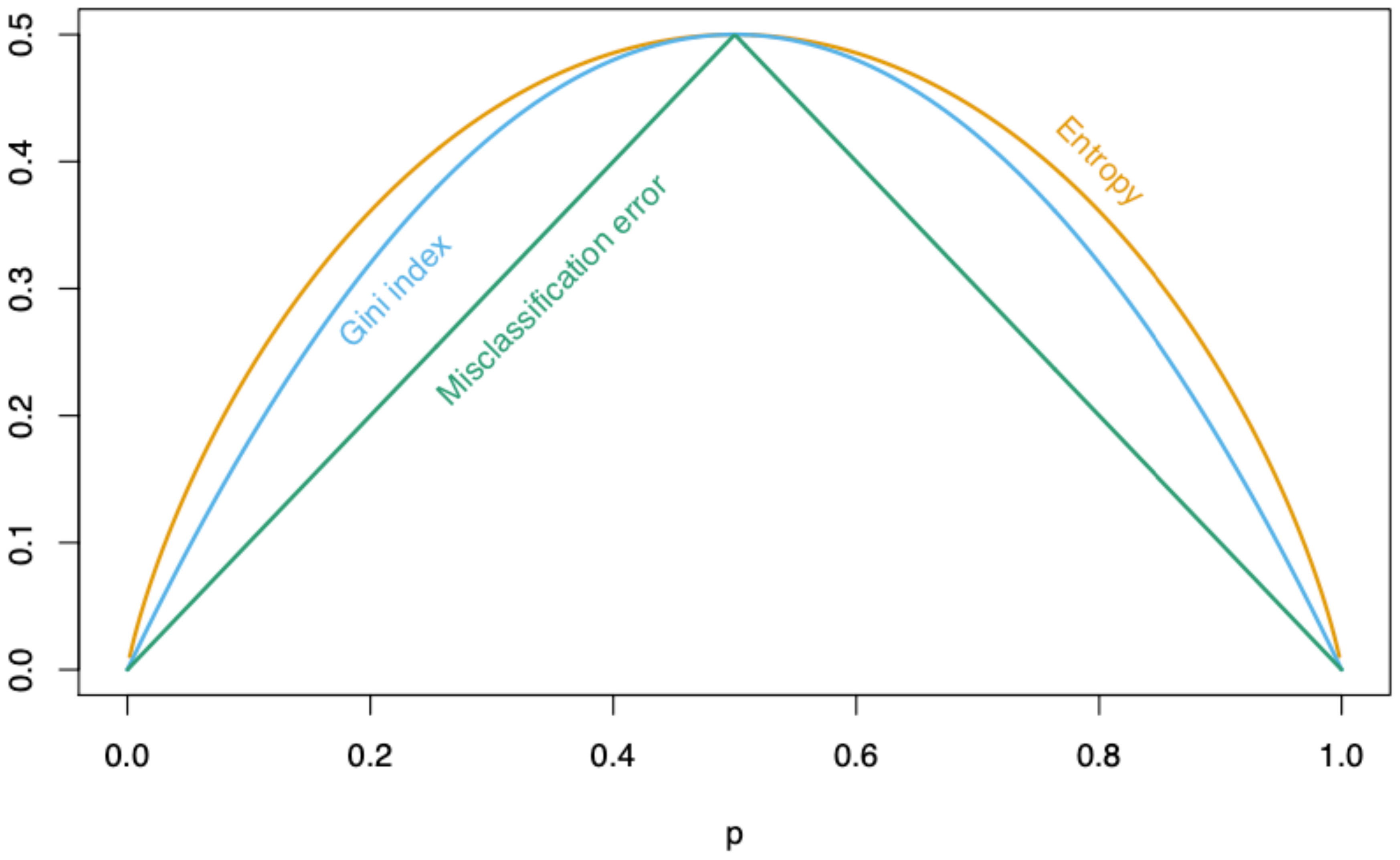
Clarifications from last time: Why Gini index?

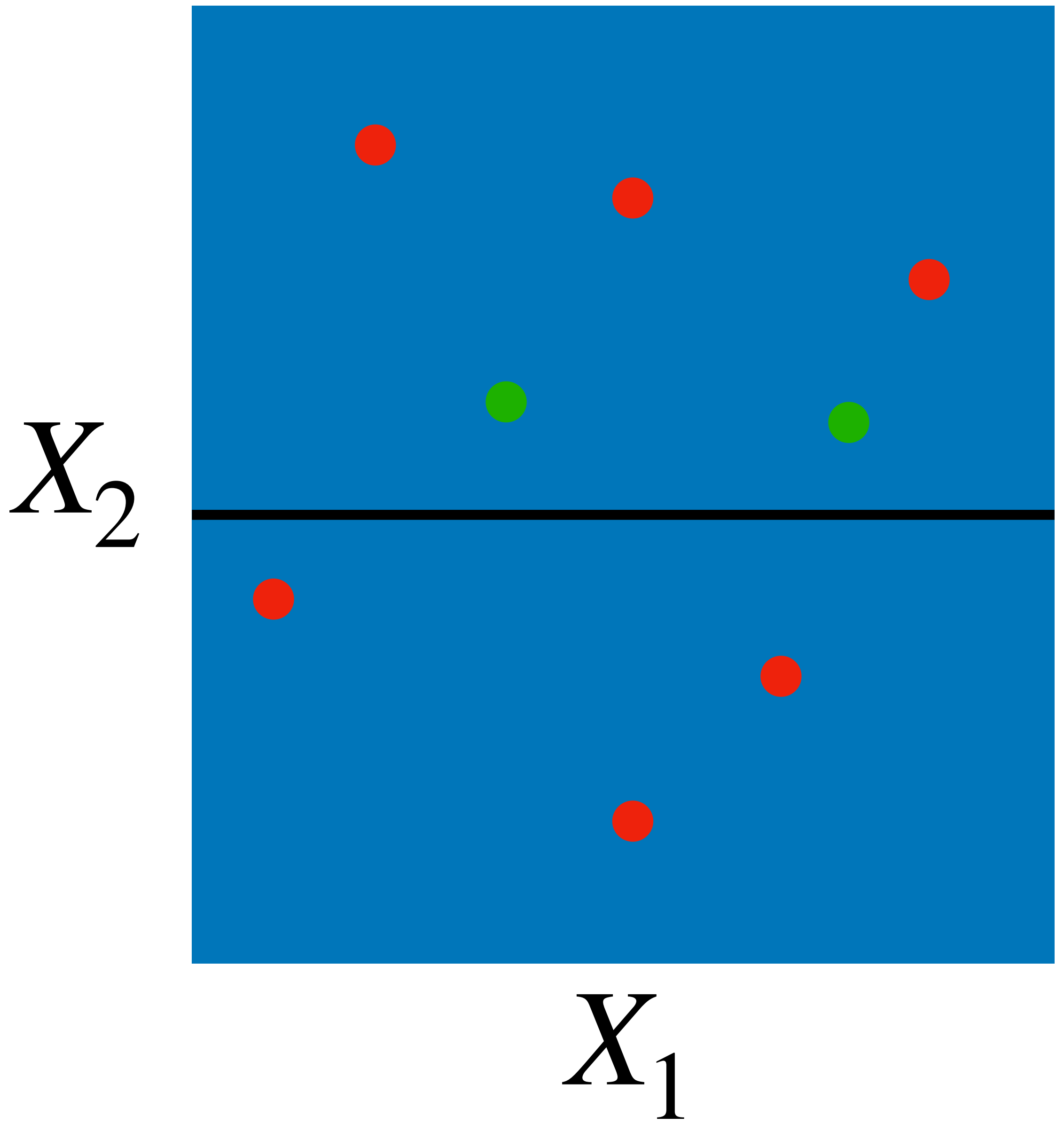




Tree	Misclass. error
No splits	

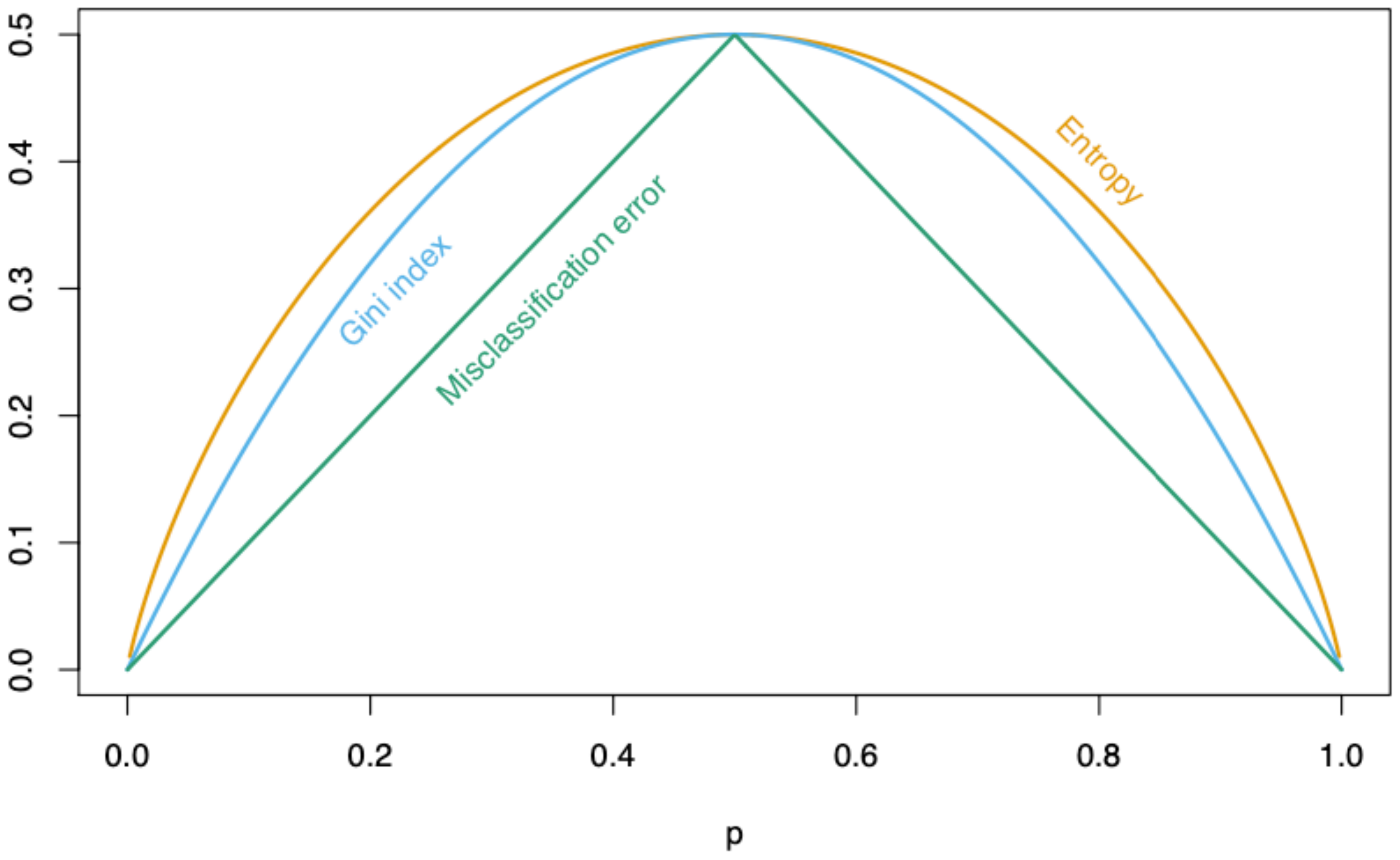
Clarifications from last time: Why Gini index?

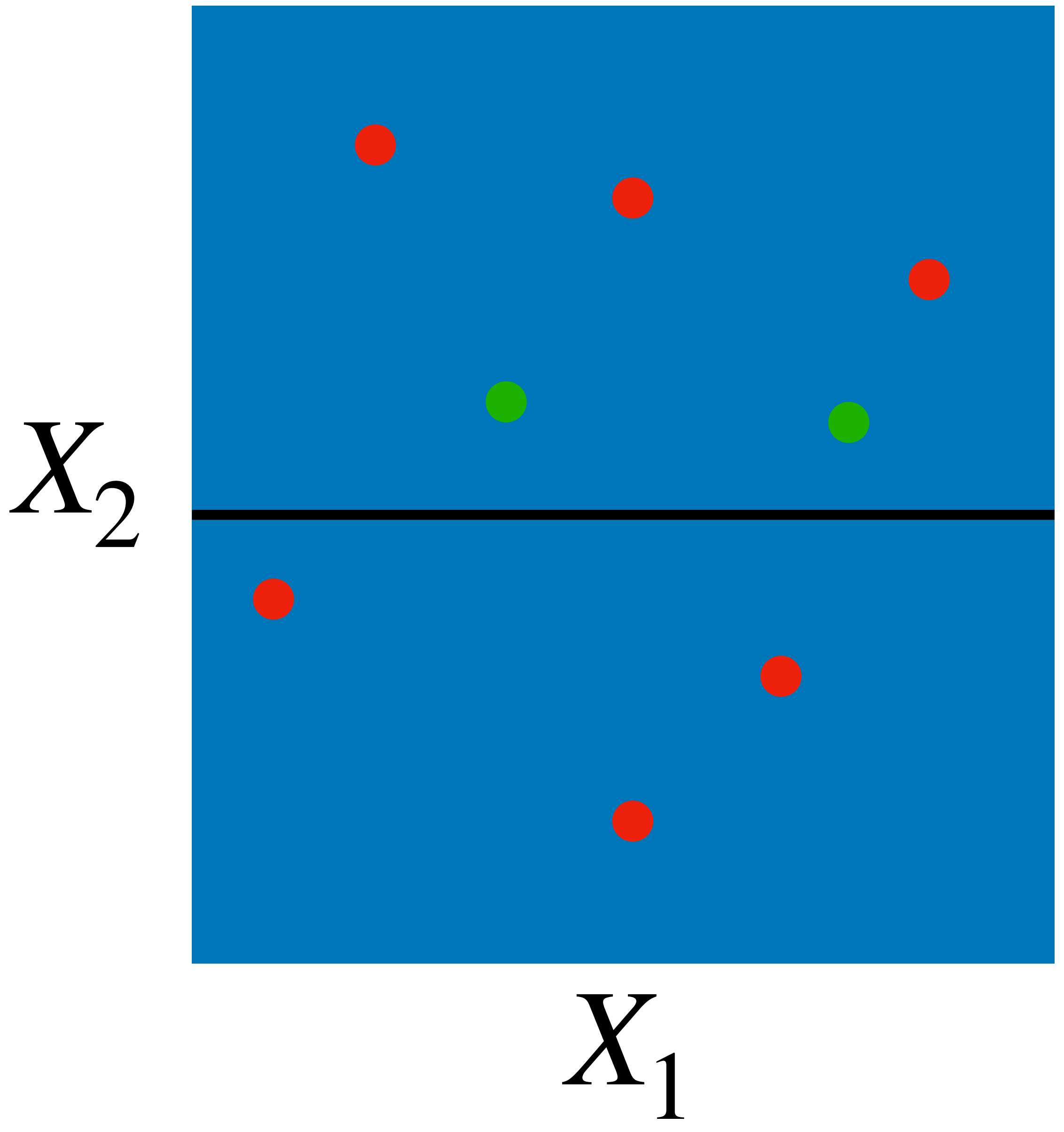




Tree	Misclass. error
No splits	$\frac{1}{4}$

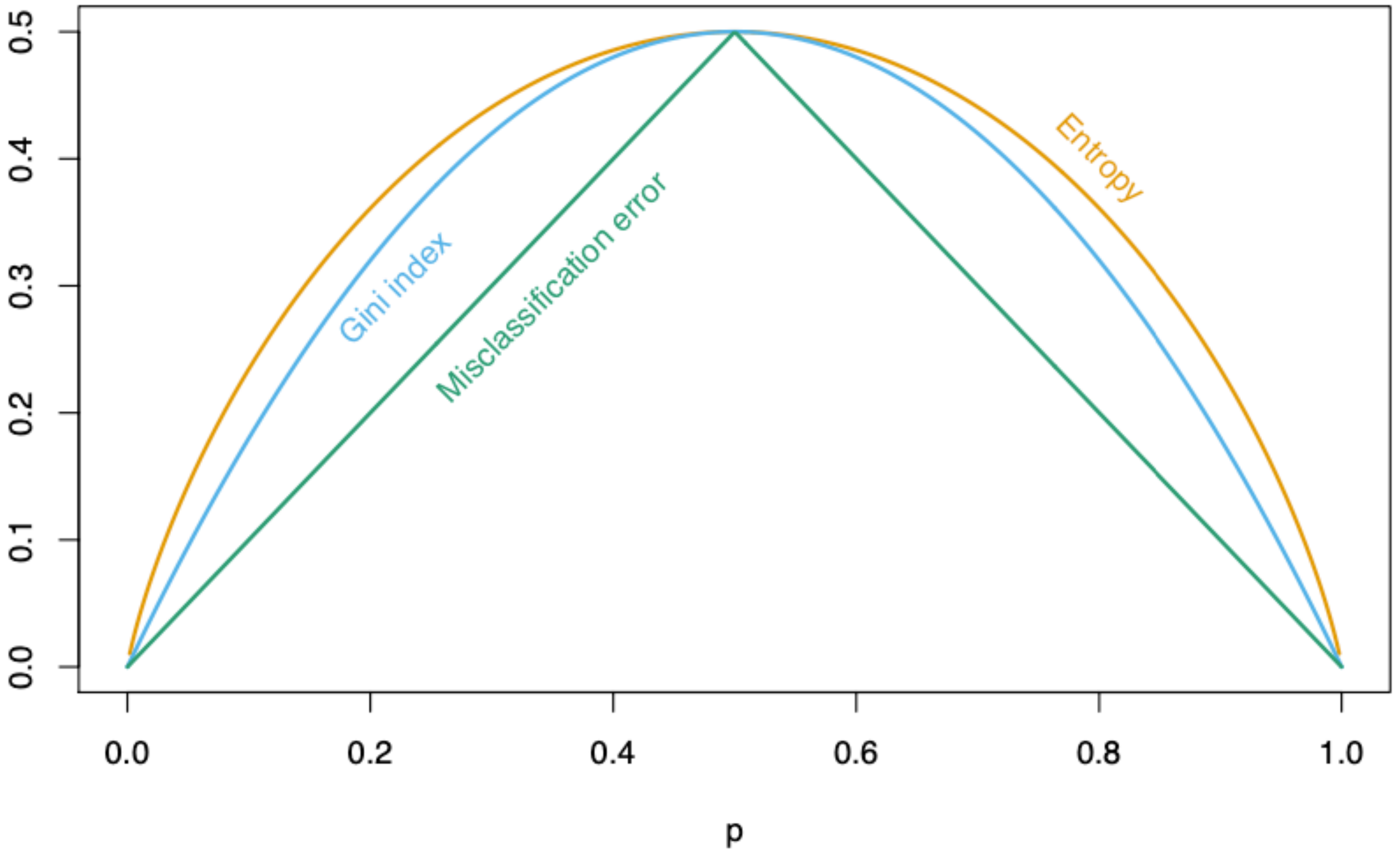
Clarifications from last time: Why Gini index?

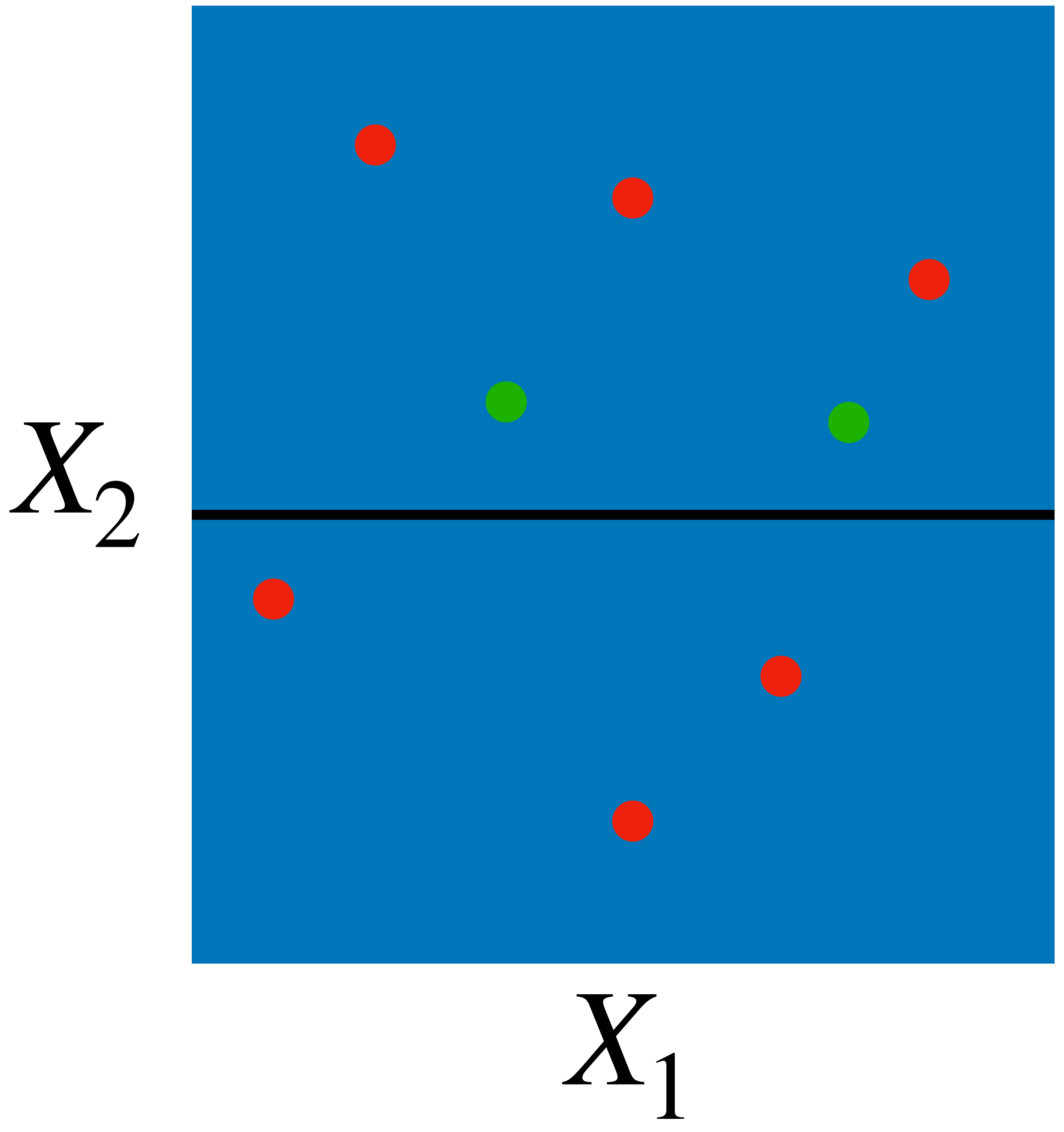




Tree	Misclass. error
No splits	$\frac{1}{4}$
One split	

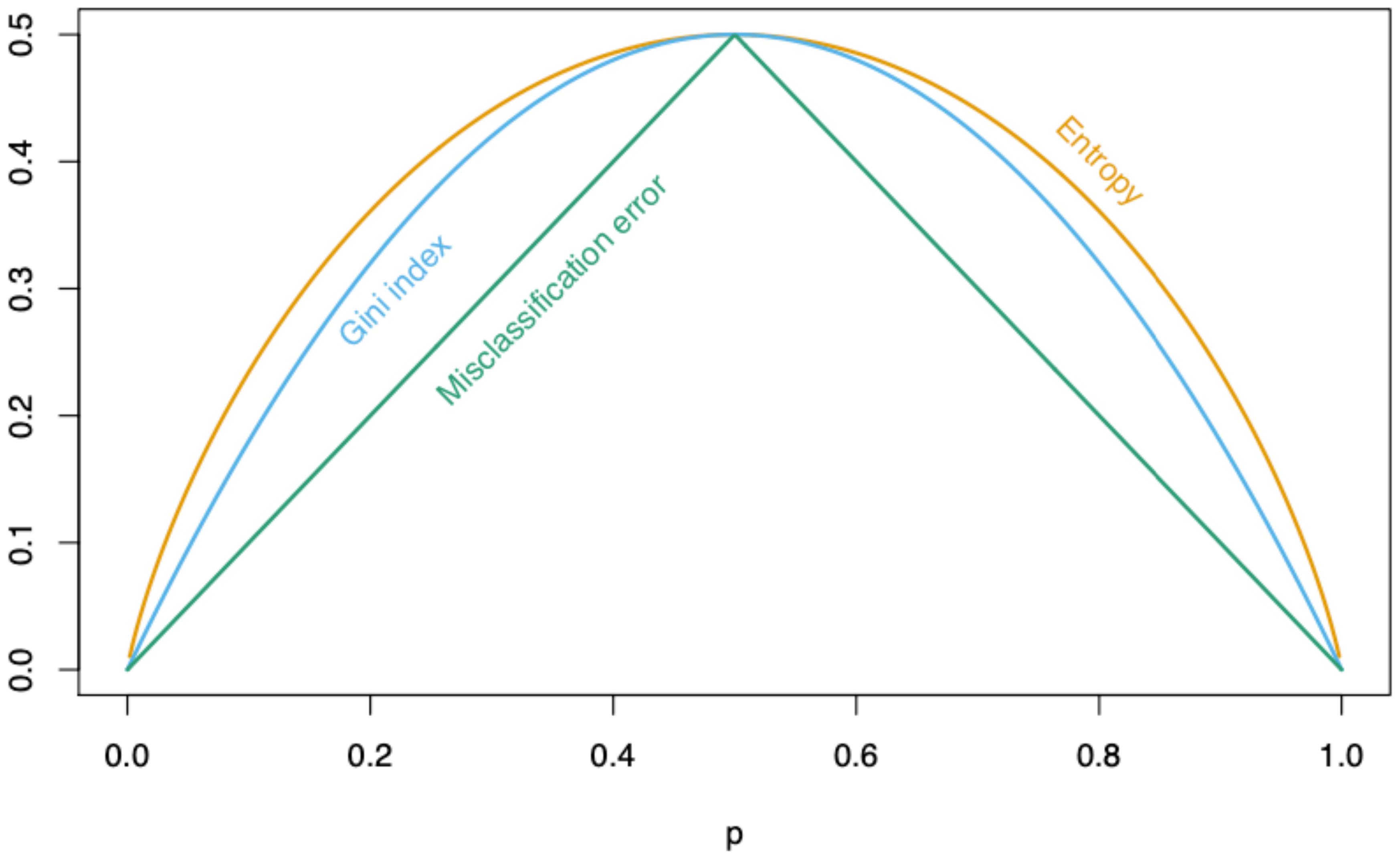
Clarifications from last time: Why Gini index?

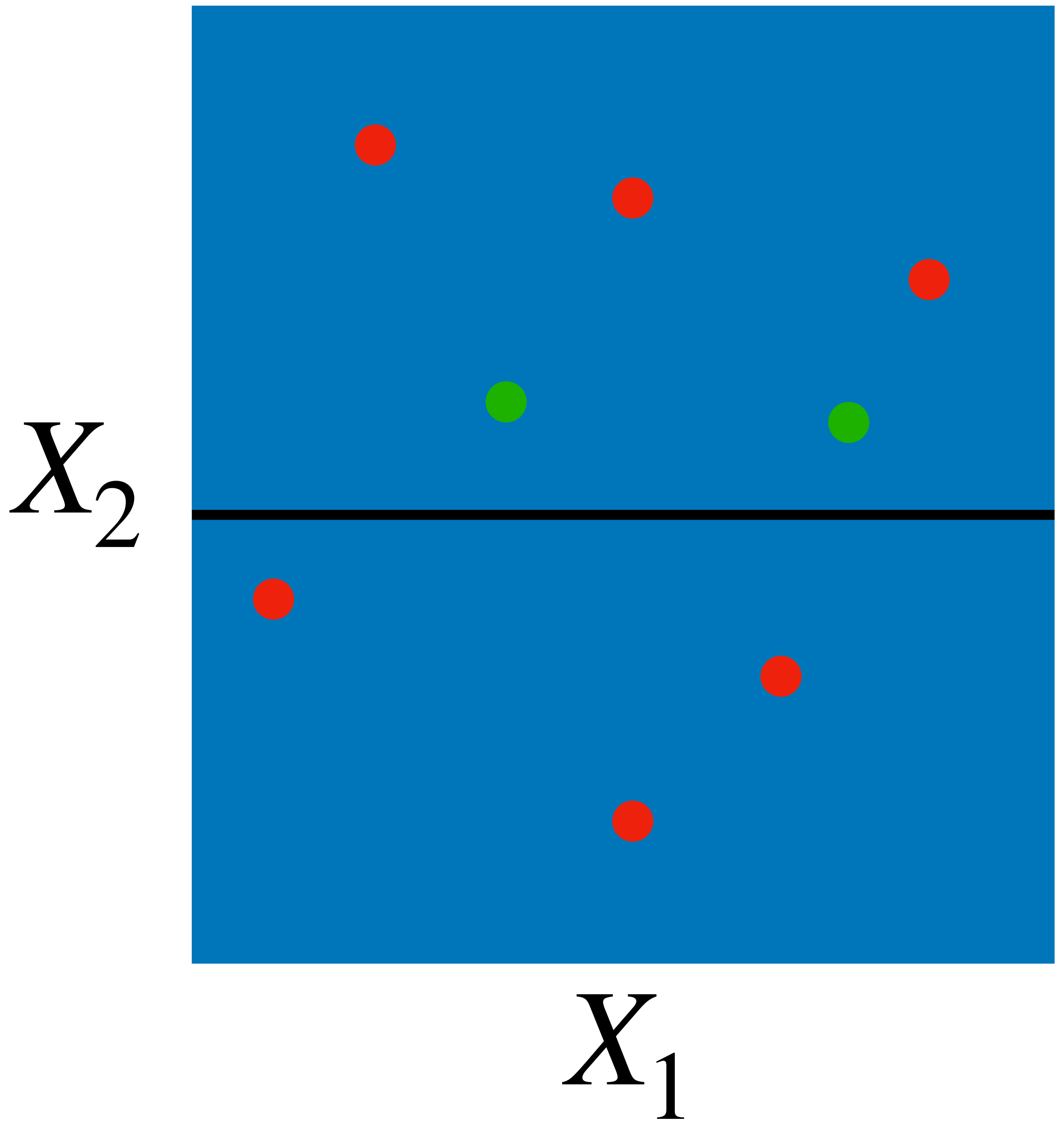




Tree	Misclass. error
No splits	$\frac{1}{4}$
One split	$\frac{1}{4}$

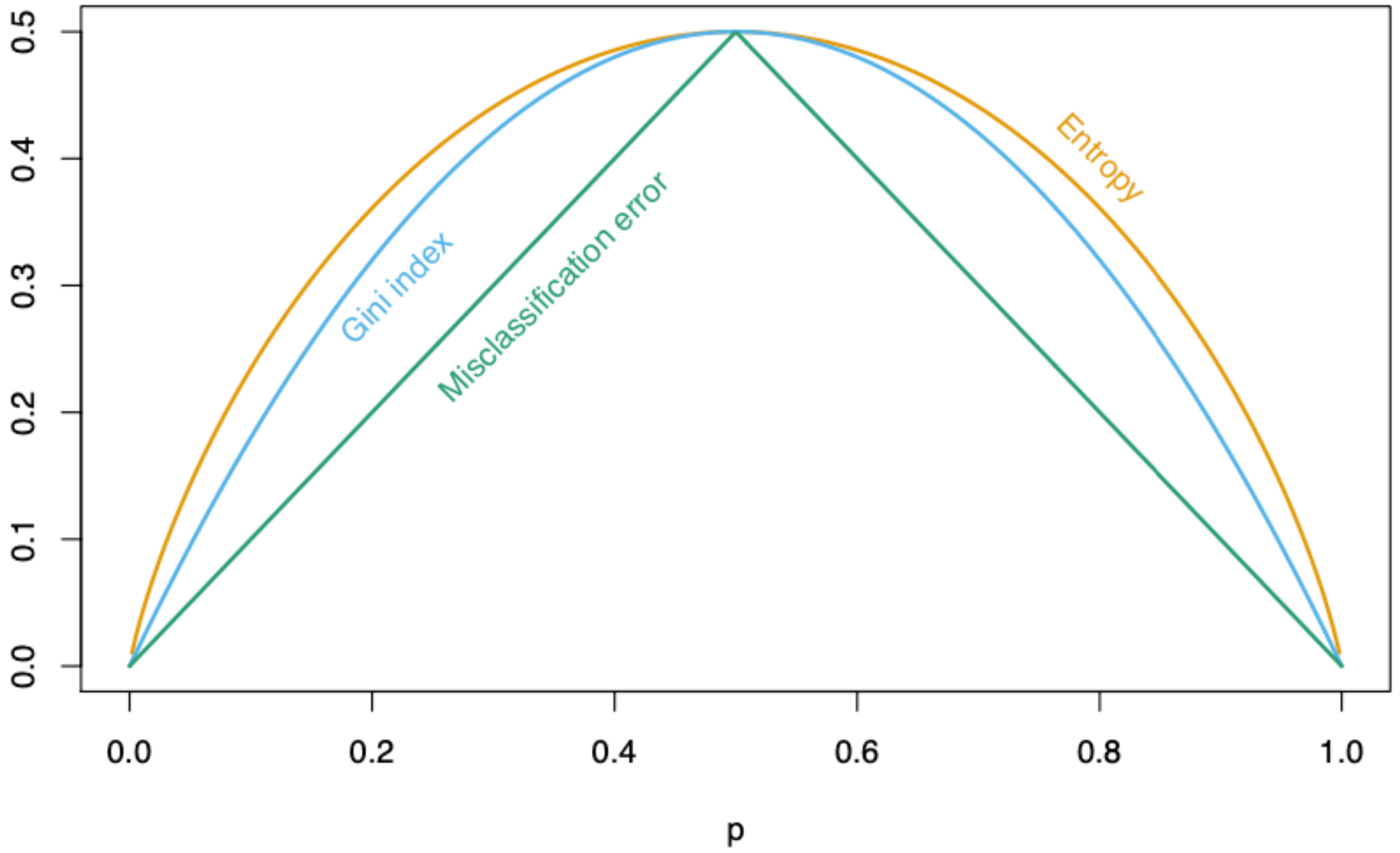
Clarifications from last time: Why Gini index?

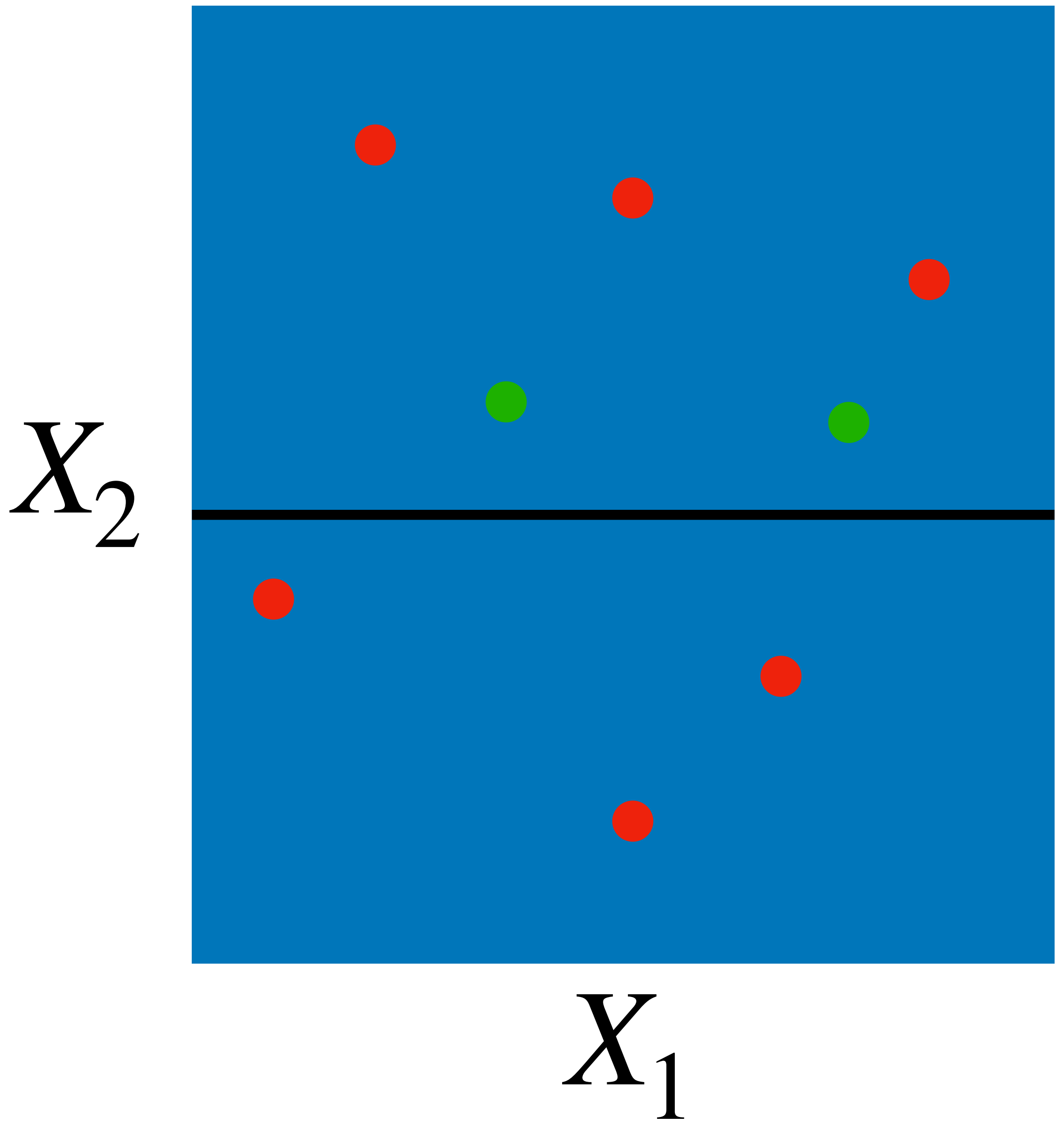




Tree	Misclass. error	Total Gini index
No splits	$\frac{1}{4}$	
One split	$\frac{1}{4}$	

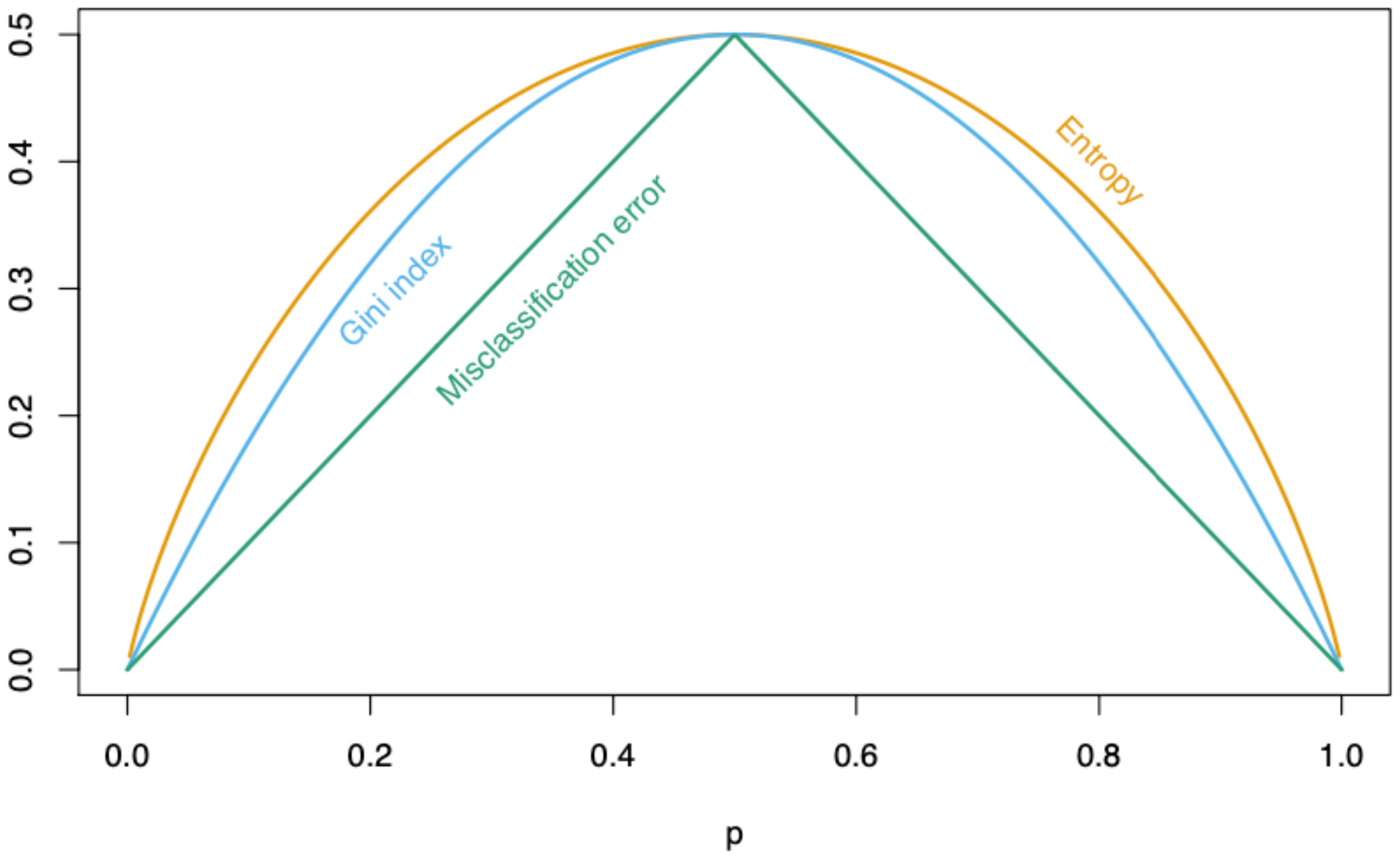
Clarifications from last time: Why Gini index?

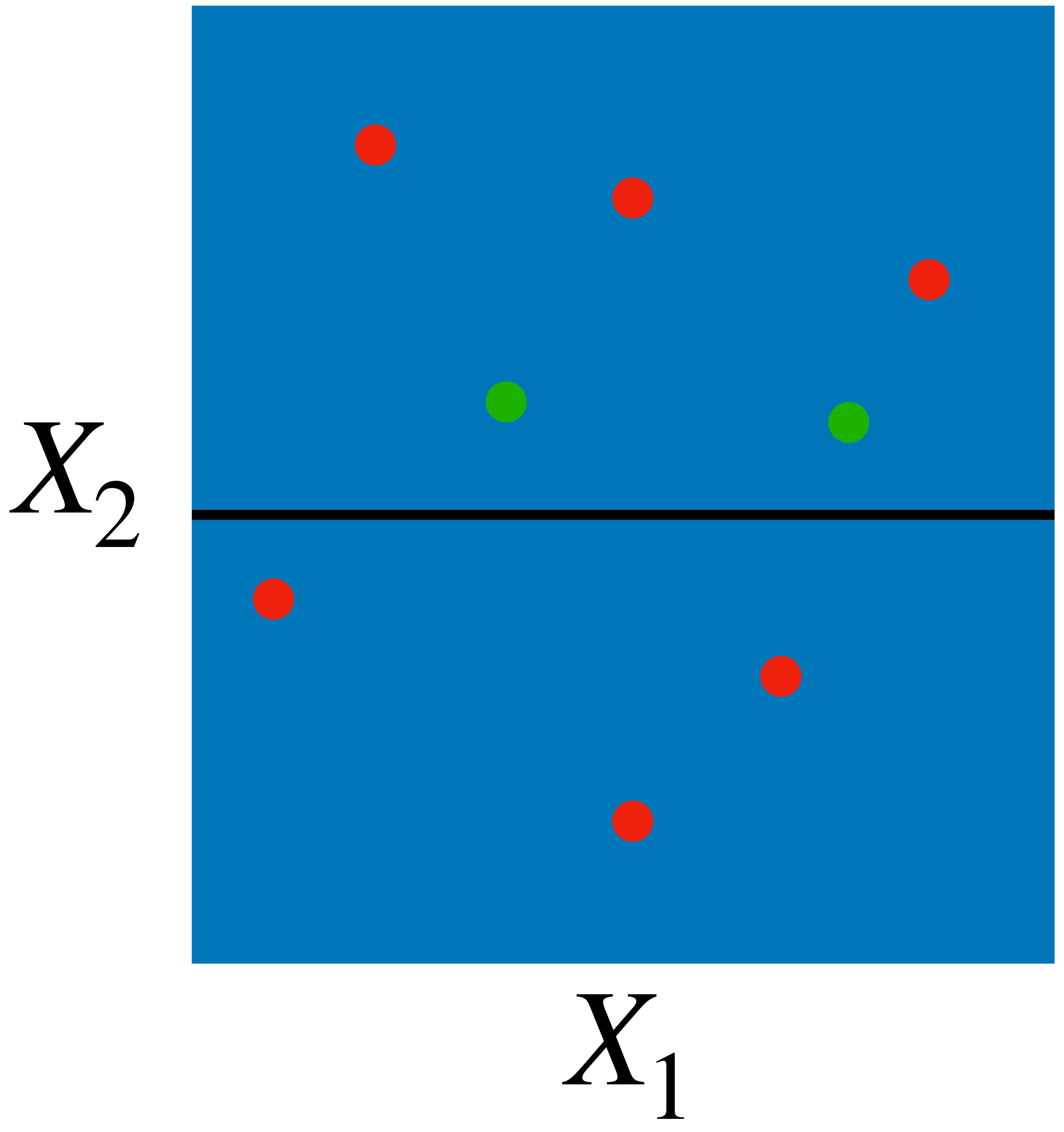




Tree	Misclass. error	Total Gini index
No splits	$\frac{1}{4}$	$8 \cdot 2 \cdot \frac{1}{4} \cdot \frac{3}{4} = 3$
One split	$\frac{1}{4}$	

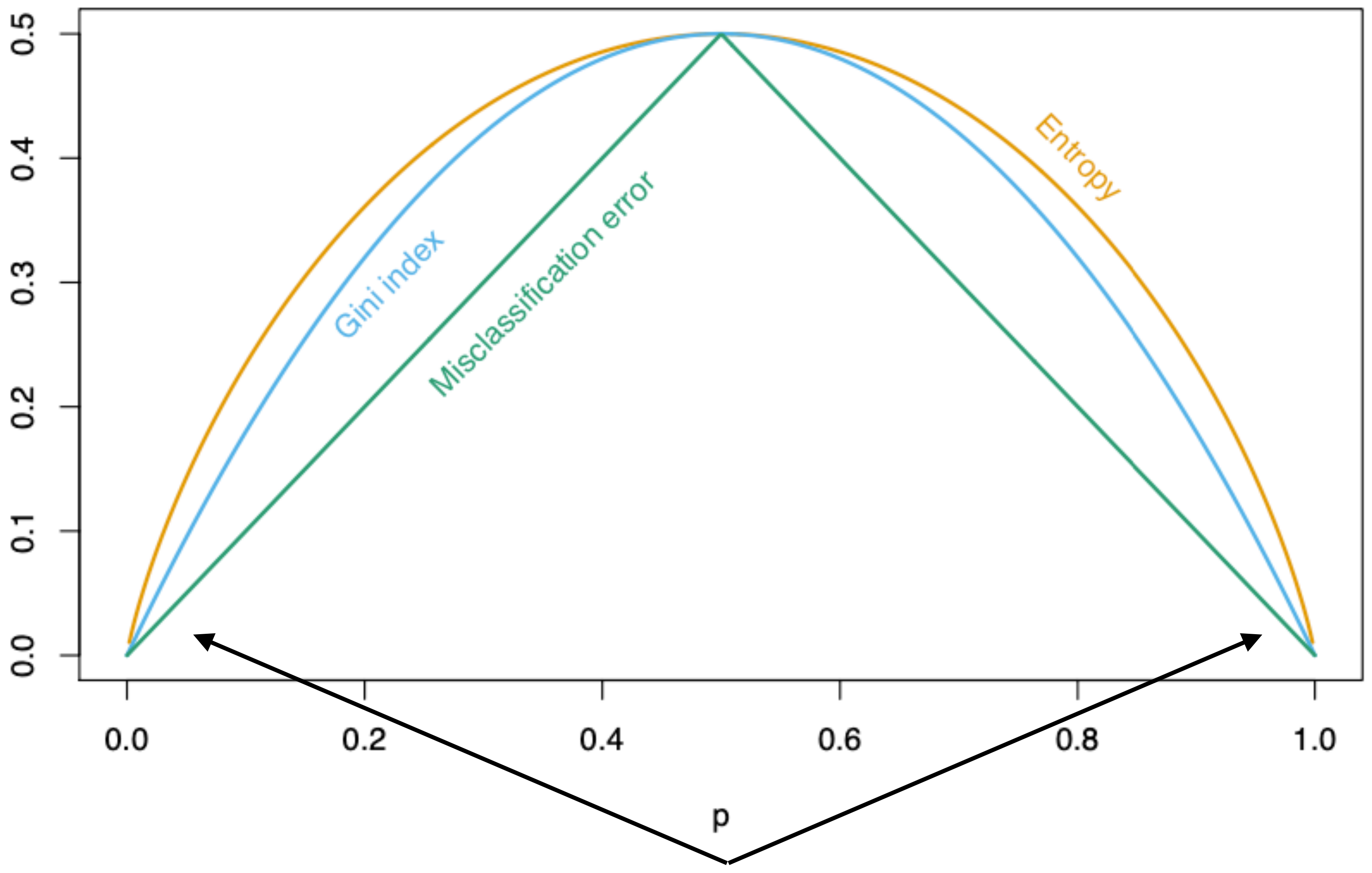
Clarifications from last time: Why Gini index?



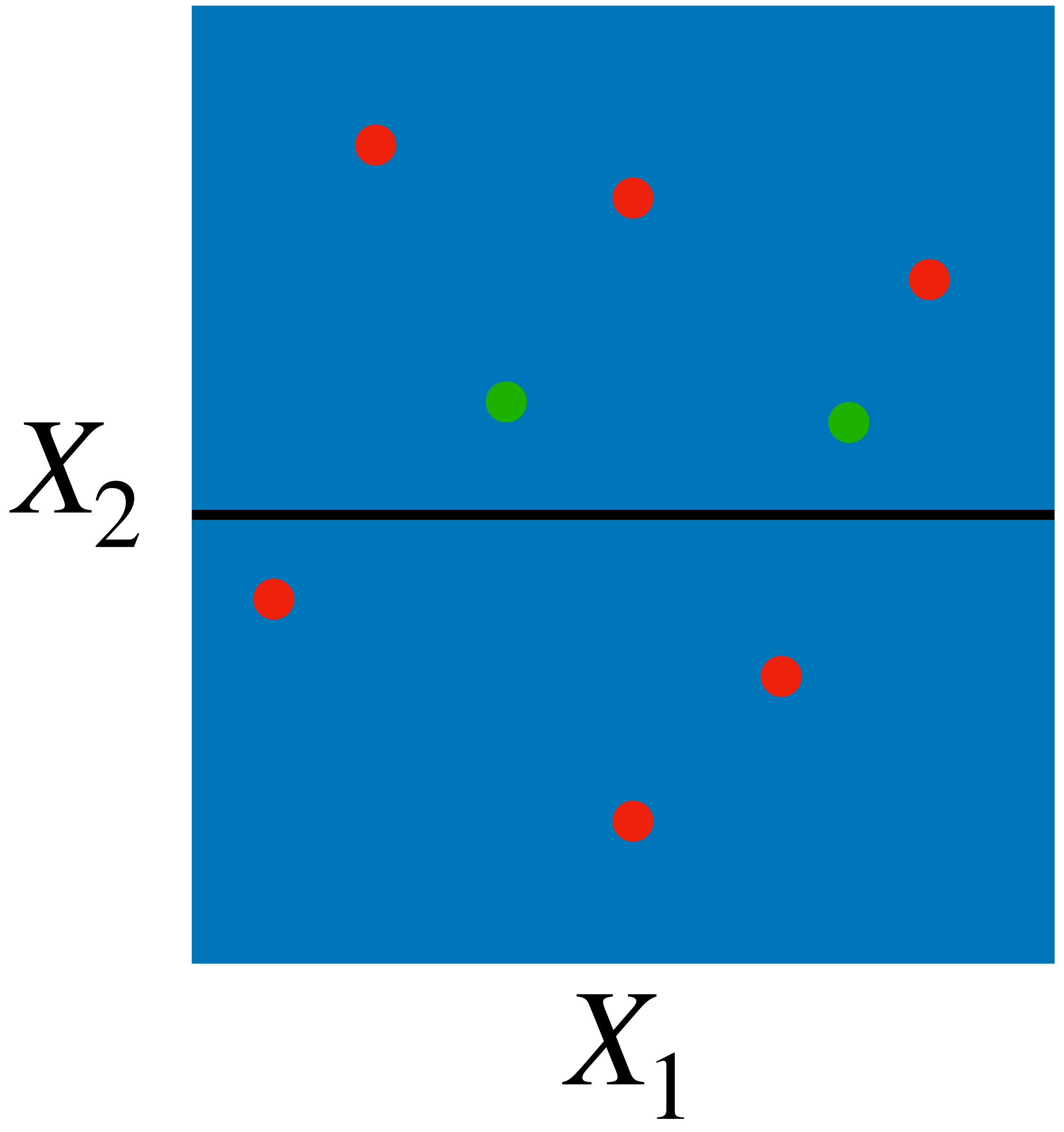


Tree	Misclass. error	Total Gini index
No splits	$\frac{1}{4}$	$8 \cdot 2 \cdot \frac{1}{4} \cdot \frac{3}{4} = 3$
One split	$\frac{1}{4}$	$5 \cdot 2 \cdot \frac{2}{5} \cdot \frac{3}{5} + 3 \cdot 0 = 2.4$

Clarifications from last time: Why Gini index?



Rises more sharply away from 0 and 1, promoting node purity



Tree	Misclass. error	Total Gini index
No splits	$\frac{1}{4}$	$8 \cdot 2 \cdot \frac{1}{4} \cdot \frac{3}{4} = 3$
One split	$\frac{1}{4}$	$5 \cdot 2 \cdot \frac{2}{5} \cdot \frac{3}{5} + 3 \cdot 0 = 2.4$

Complexity of a decision tree

The more terminal nodes (regions), the more flexibly the tree fits training data:

- if there are as many terminal nodes as training points, training error = 0
- If there is just one terminal node, we are fitting a constant model

As with any prediction method, there is a bias-variance tradeoff based on model complexity.

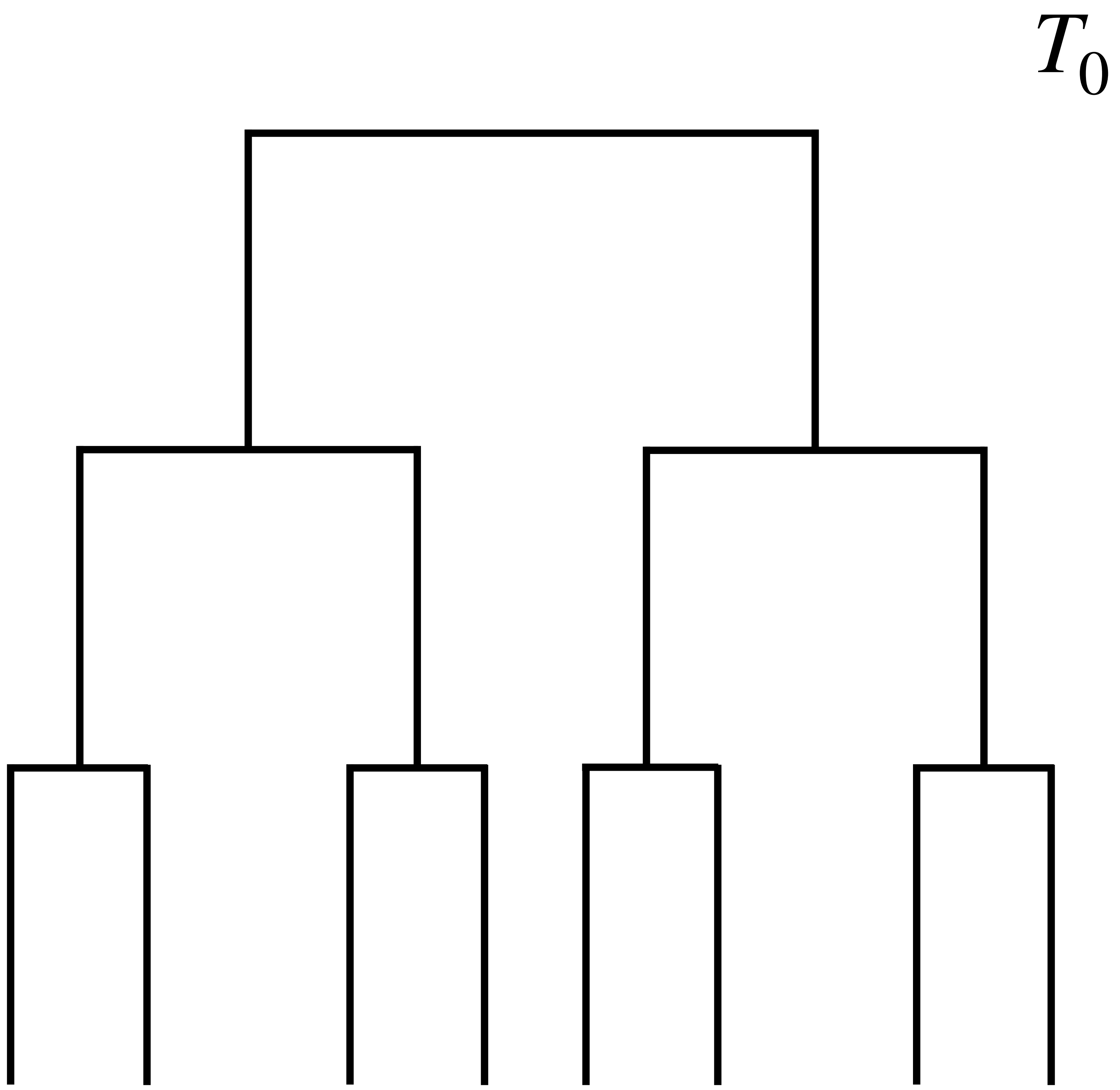
How to choose the best model complexity? Cross-validation.

A family of decision trees of varying complexity

- First grow out our tree about as far as we can to obtain a big tree T_0.
- We can then consider any subtree $T \subseteq T_0$.

Note: There are several subtrees T for each complexity value.

In other model selection scenarios, we have just had one model for each complexity.

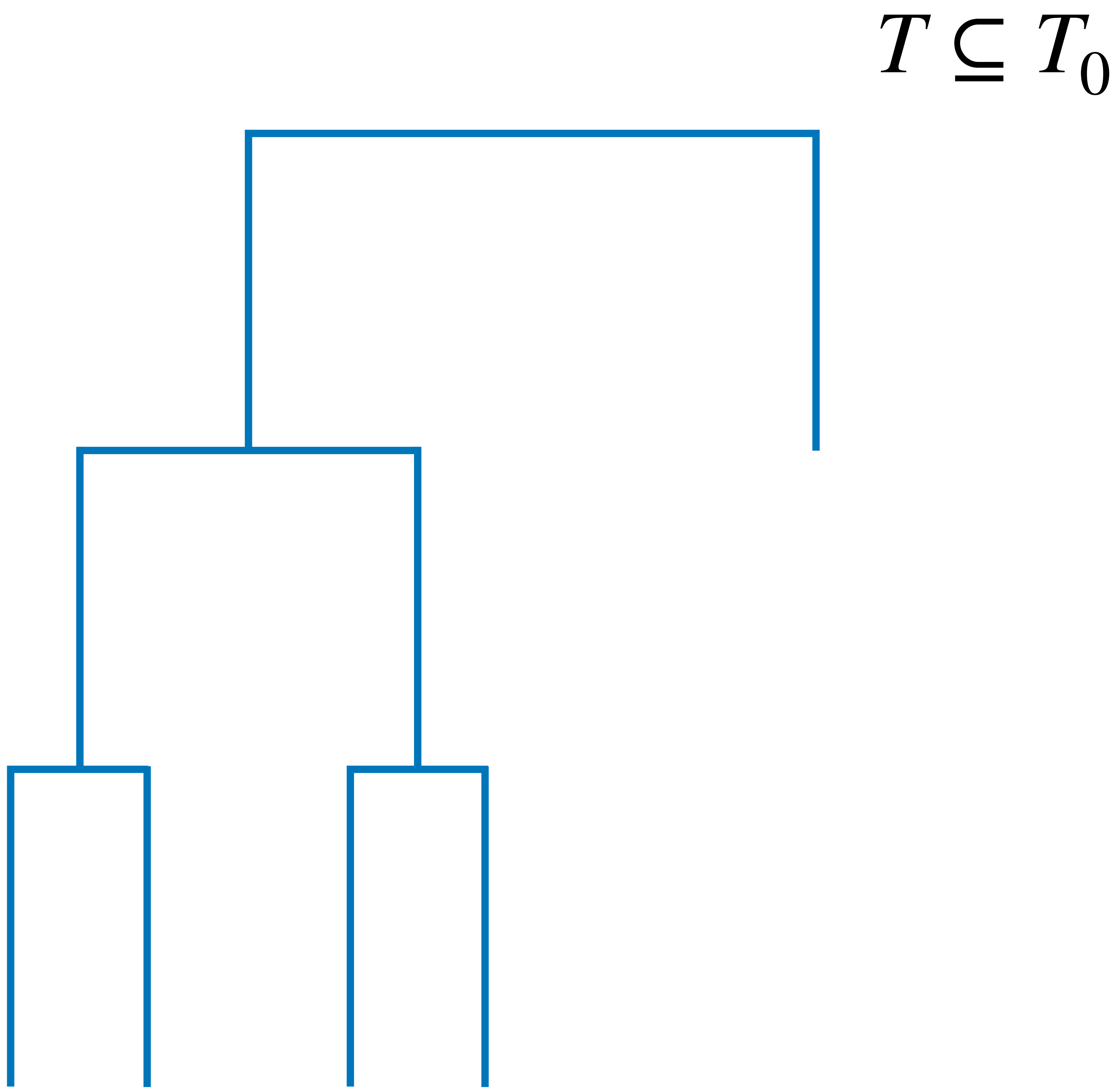


A family of decision trees of varying complexity

- First grow out our tree about as far as we can to obtain a big tree T_0.
- We can then consider any subtree $T \subseteq T_0$.

Note: There are several subtrees T for each complexity value.

In other model selection scenarios, we have just had one model for each complexity.

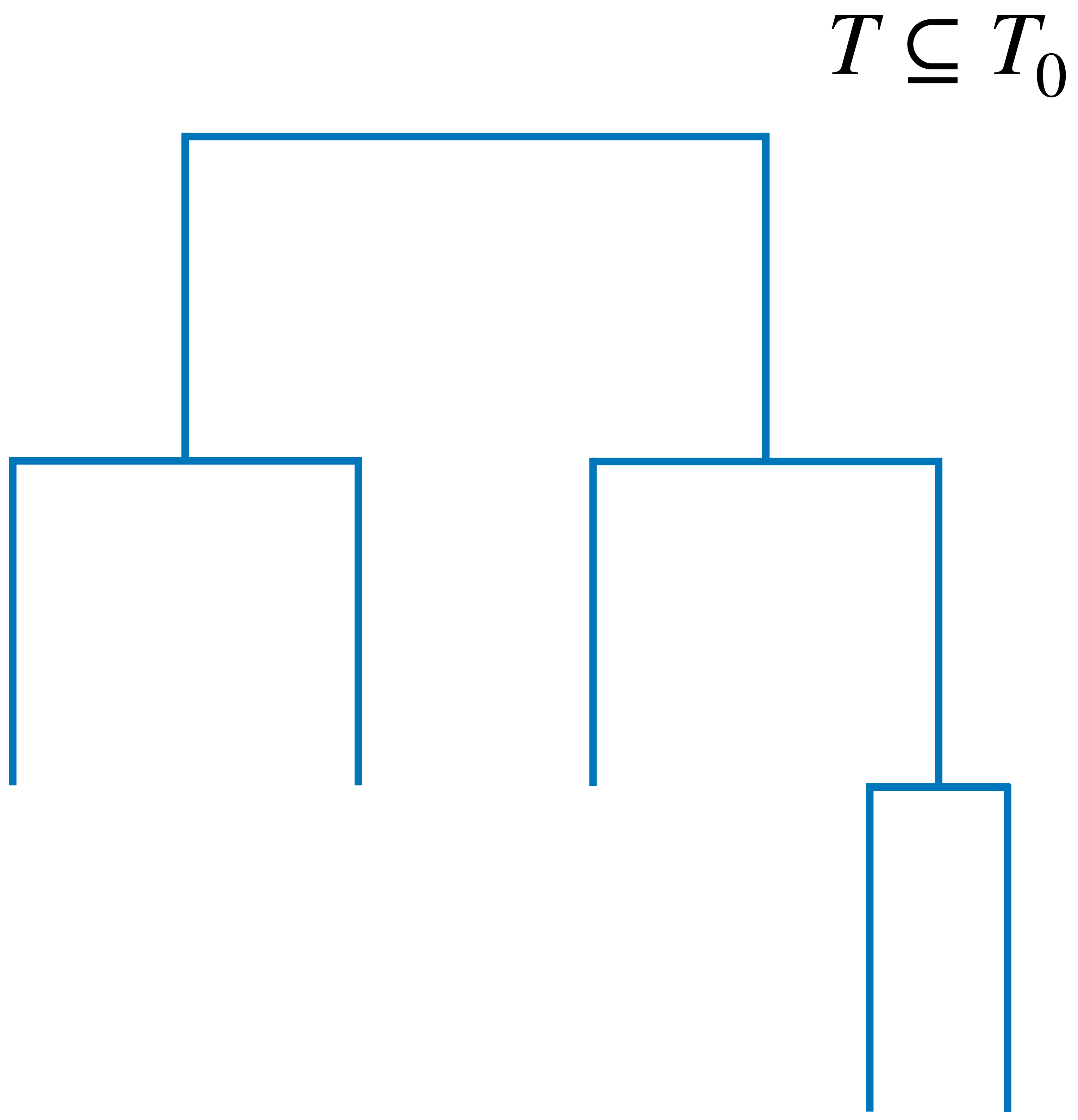


A family of decision trees of varying complexity

- First grow out our tree about as far as we can to obtain a big tree T_0.
- We can then consider any subtree $T \subseteq T_0$.

Note: There are several subtrees T for each complexity value.

In other model selection scenarios, we have just had one model for each complexity.

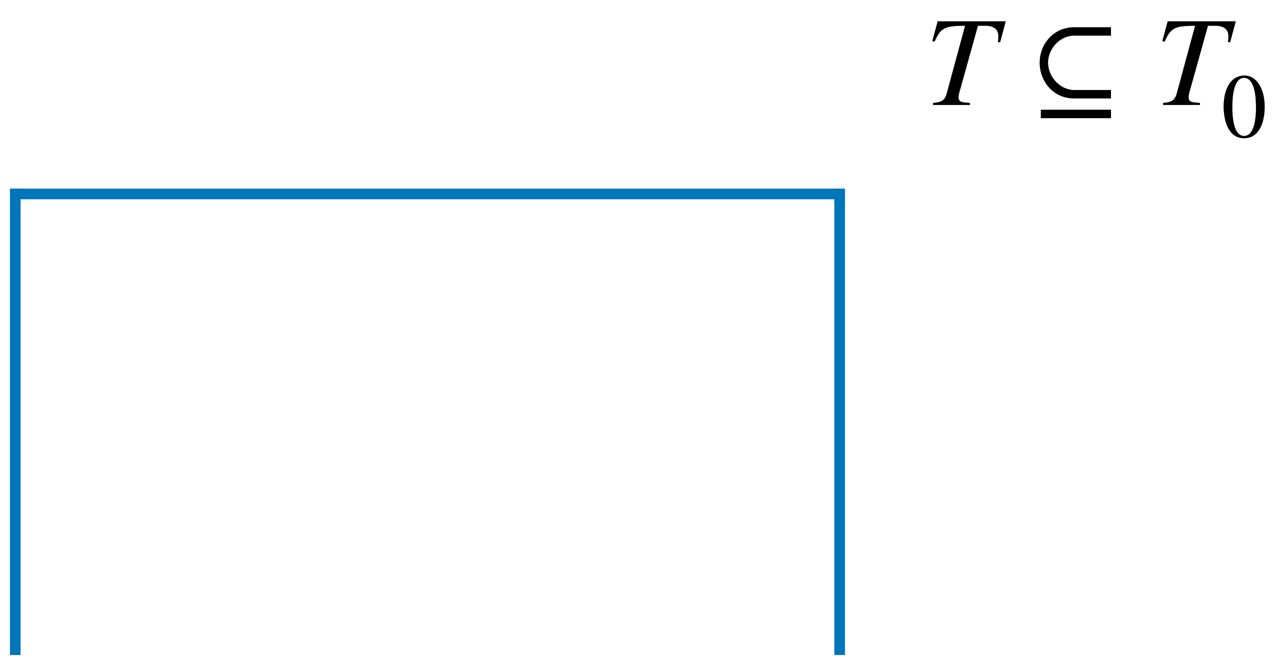


A family of decision trees of varying complexity

- First grow out our tree about as far as we can to obtain a big tree T_0.
- We can then consider any subtree $T \subseteq T_0$.

Note: There are several subtrees T for each complexity value.

In other model selection scenarios, we have just had one model for each complexity.

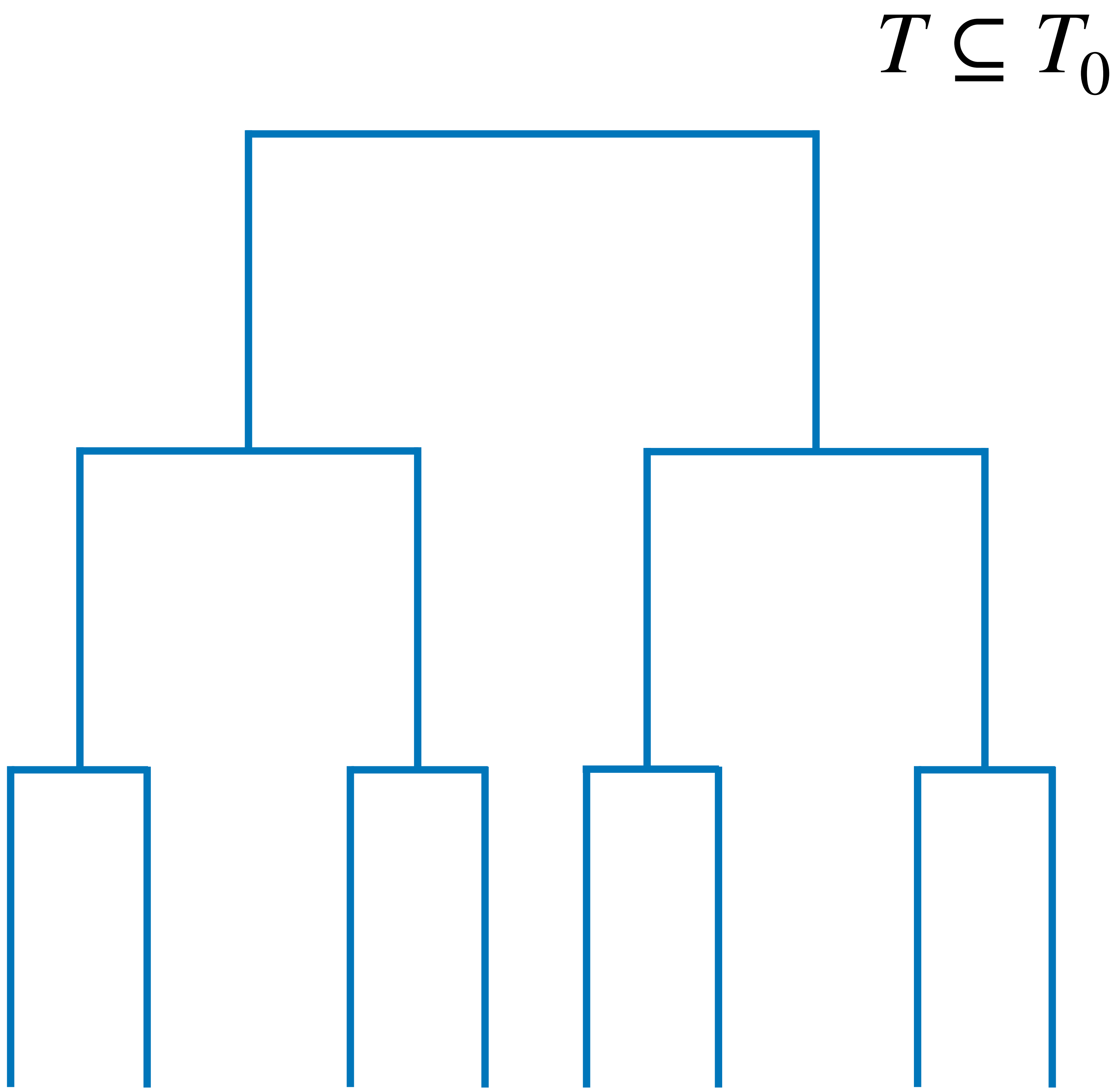


A family of decision trees of varying complexity

- First grow out our tree about as far as we can to obtain a big tree T_0.
- We can then consider any subtree $T \subseteq T_0$.

Note: There are several subtrees T for each complexity value.

In other model selection scenarios, we have just had one model for each complexity.



Imposing a penalty for terminal nodes

Let $|T|$ be number of terminal nodes in tree T. Fixing some $\alpha \geq 0$, consider

$$T_\alpha = \underset{T \subseteq T_0}{\arg\min} \left\{ \text{RSS}(T) + \alpha |T| \right\}.$$

Like lasso, varying α leads to sequence of trees; higher α leads to smaller trees.

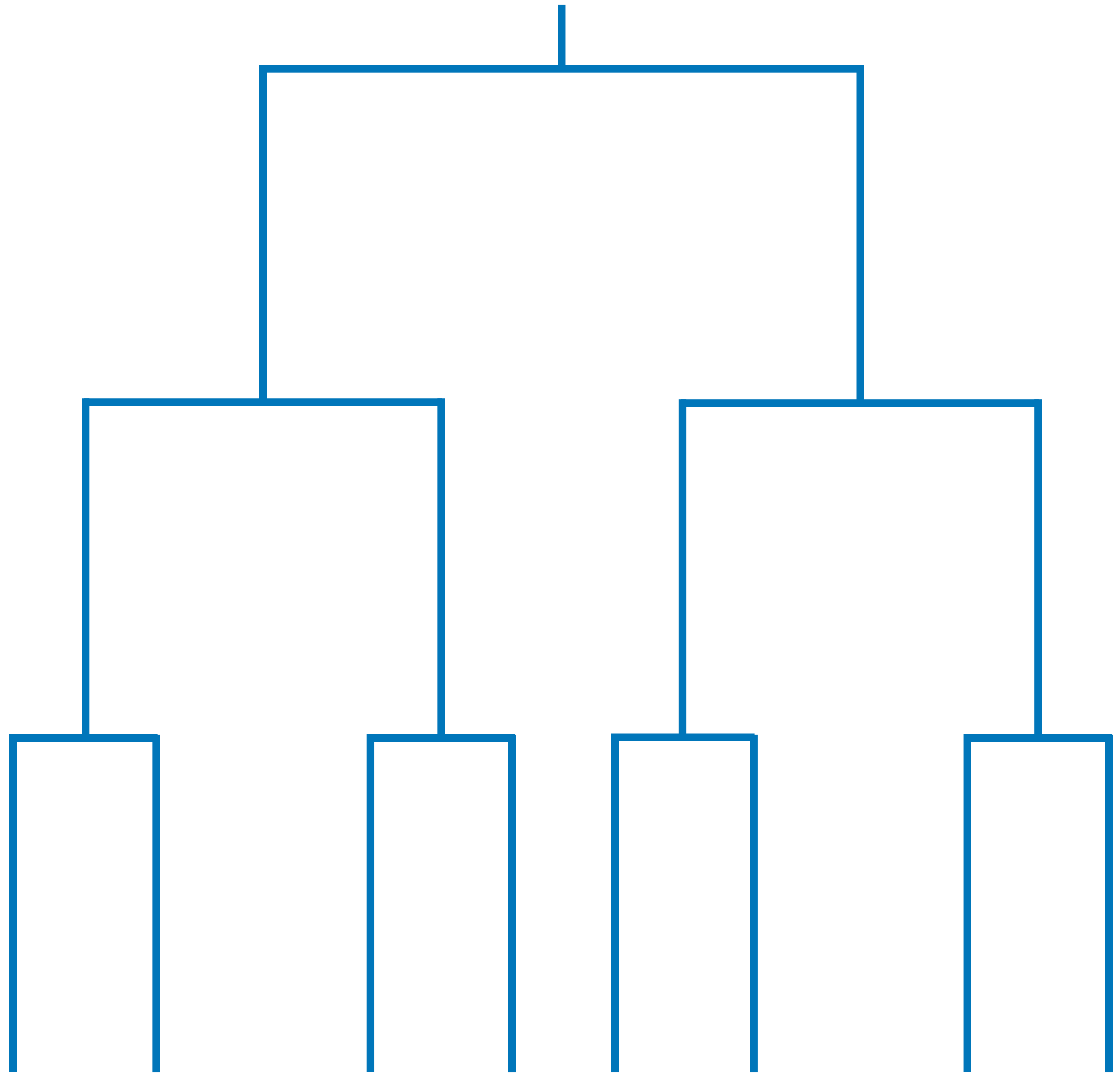
Unlike lasso, discrete set of α values gives all possible solutions as α varies.

Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune "weakest link" splits.

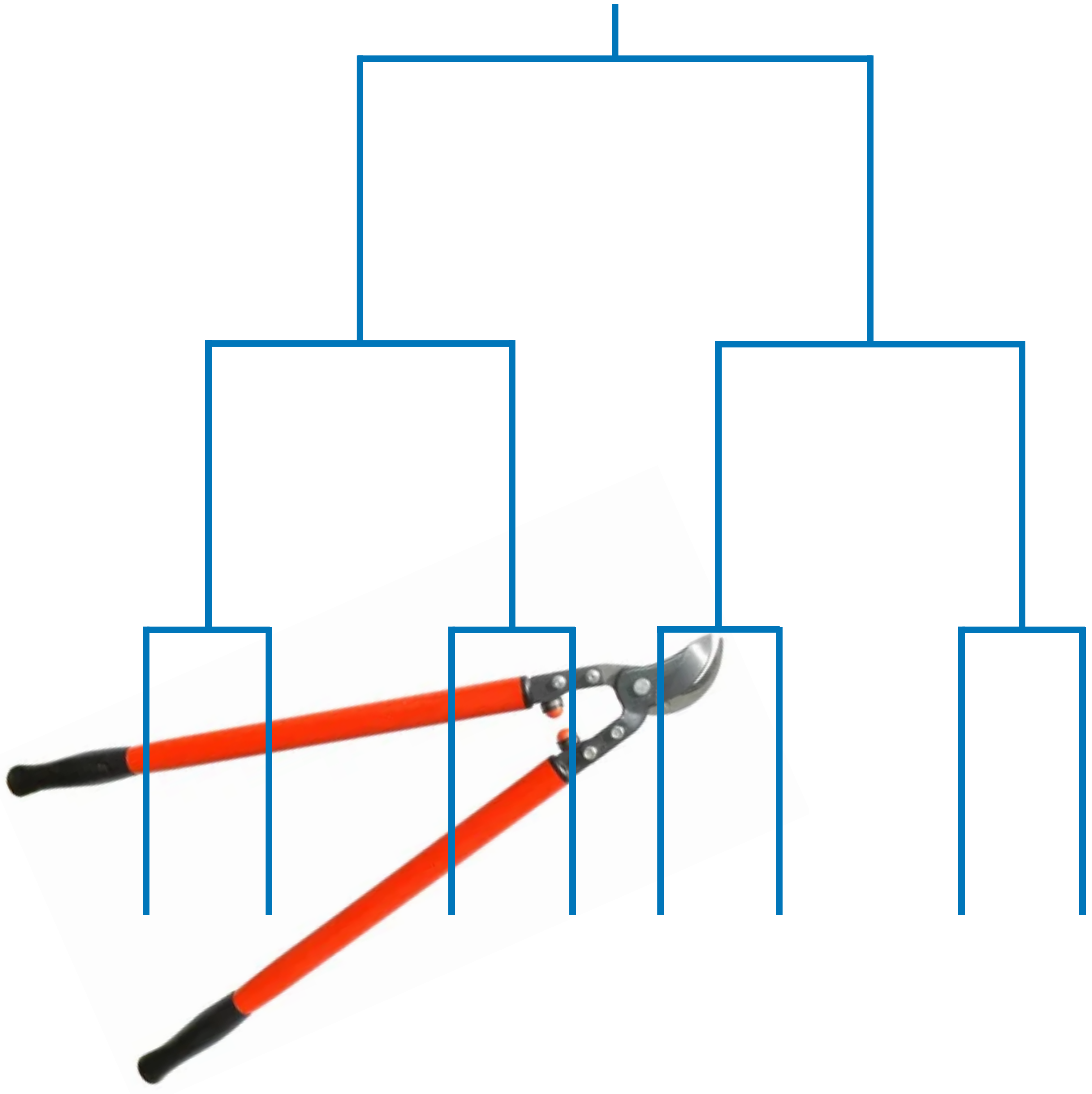


Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune "weakest link" splits.

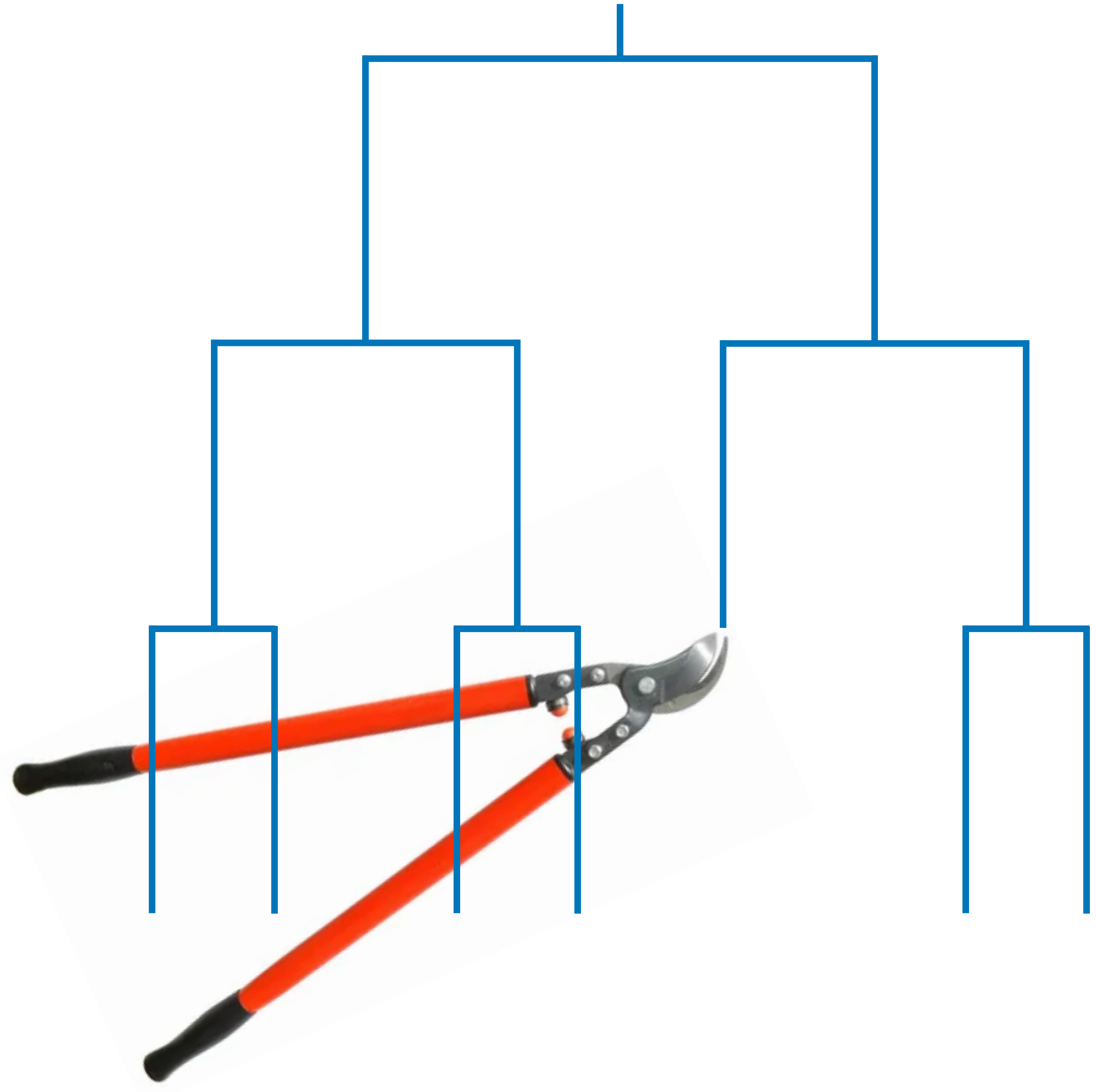


Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune "weakest link" splits.

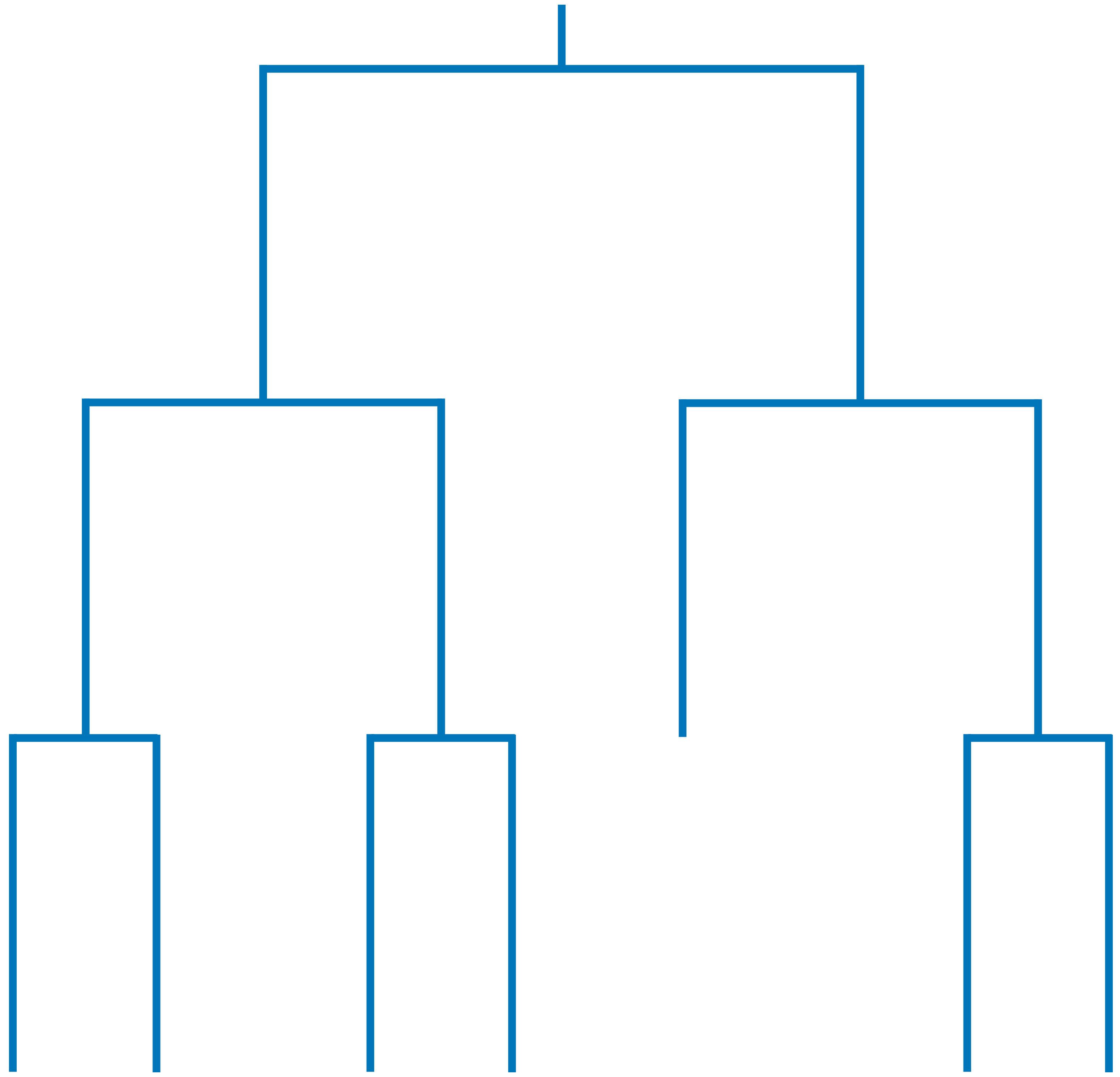


Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune “weakest link” splits.

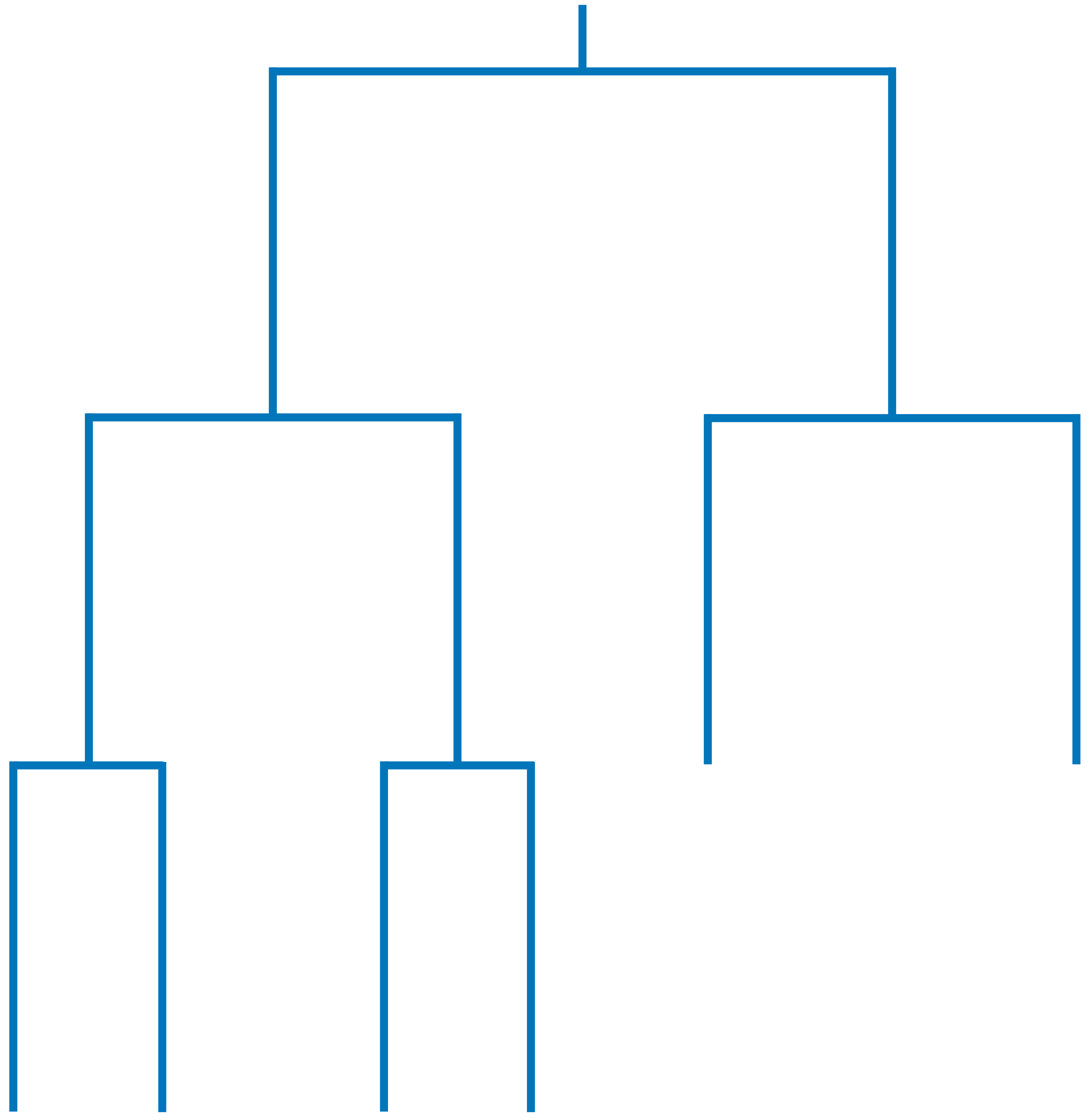


Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune "weakest link" splits.

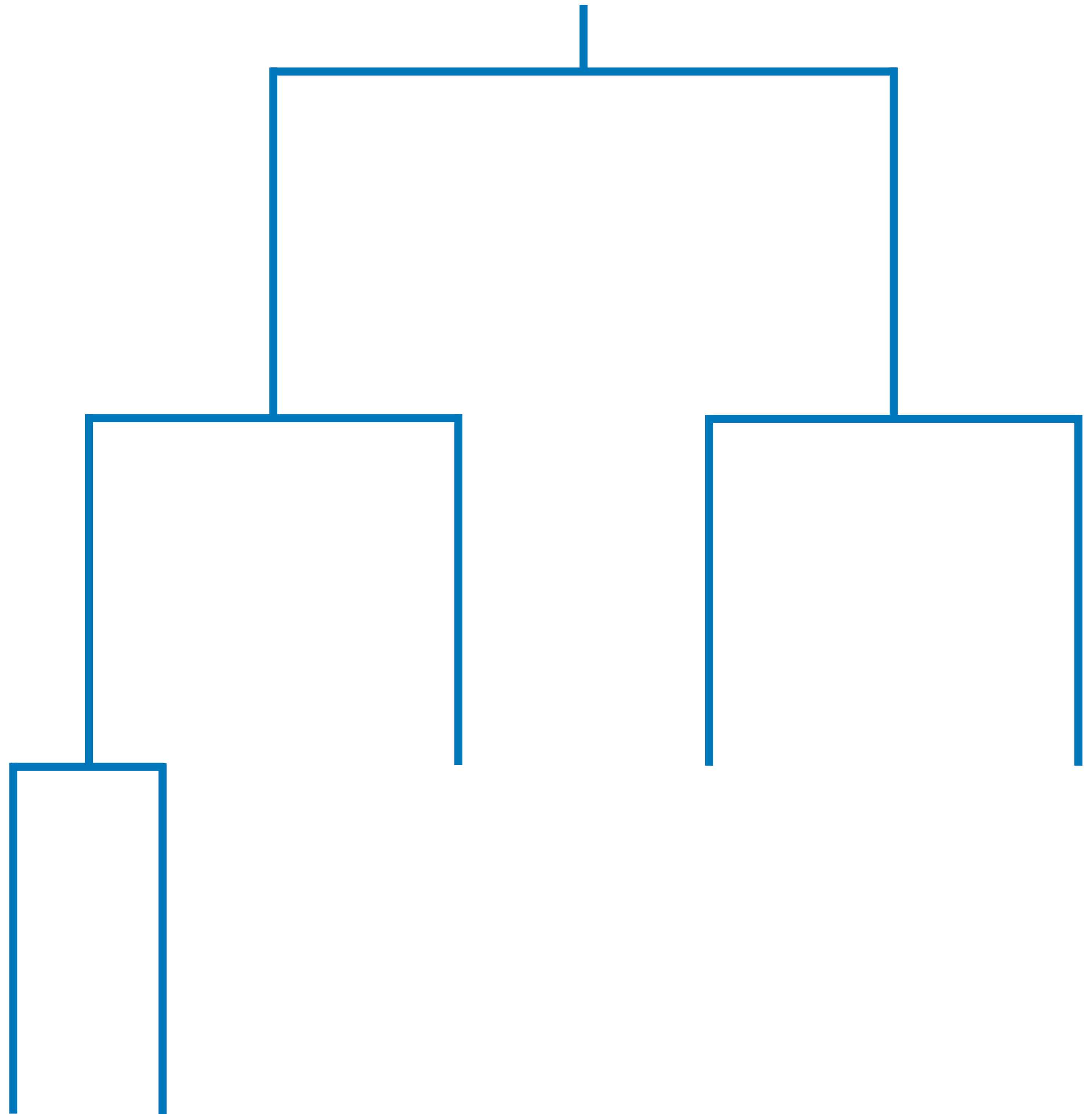


Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune "weakest link" splits.

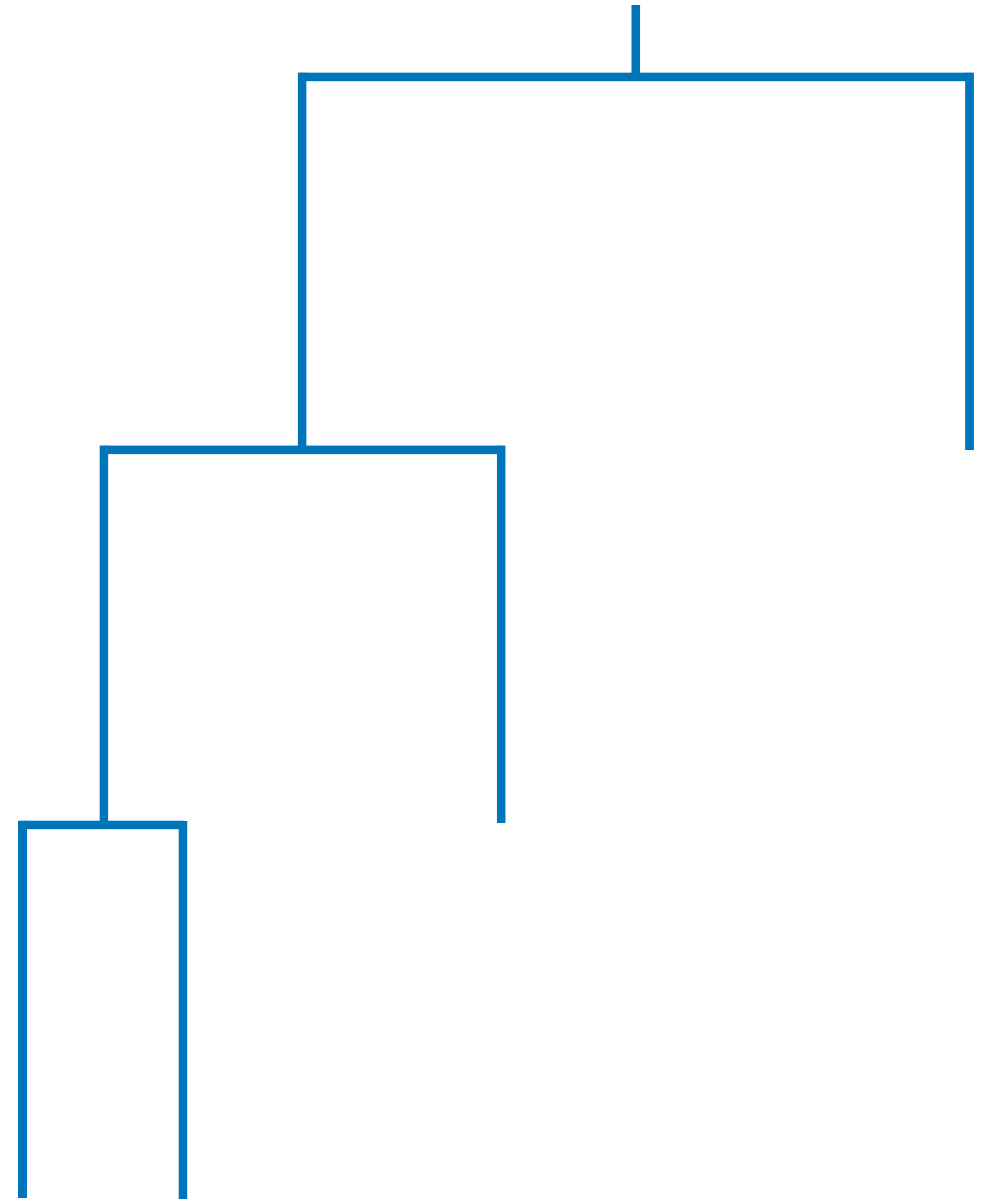


Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune “weakest link” splits.

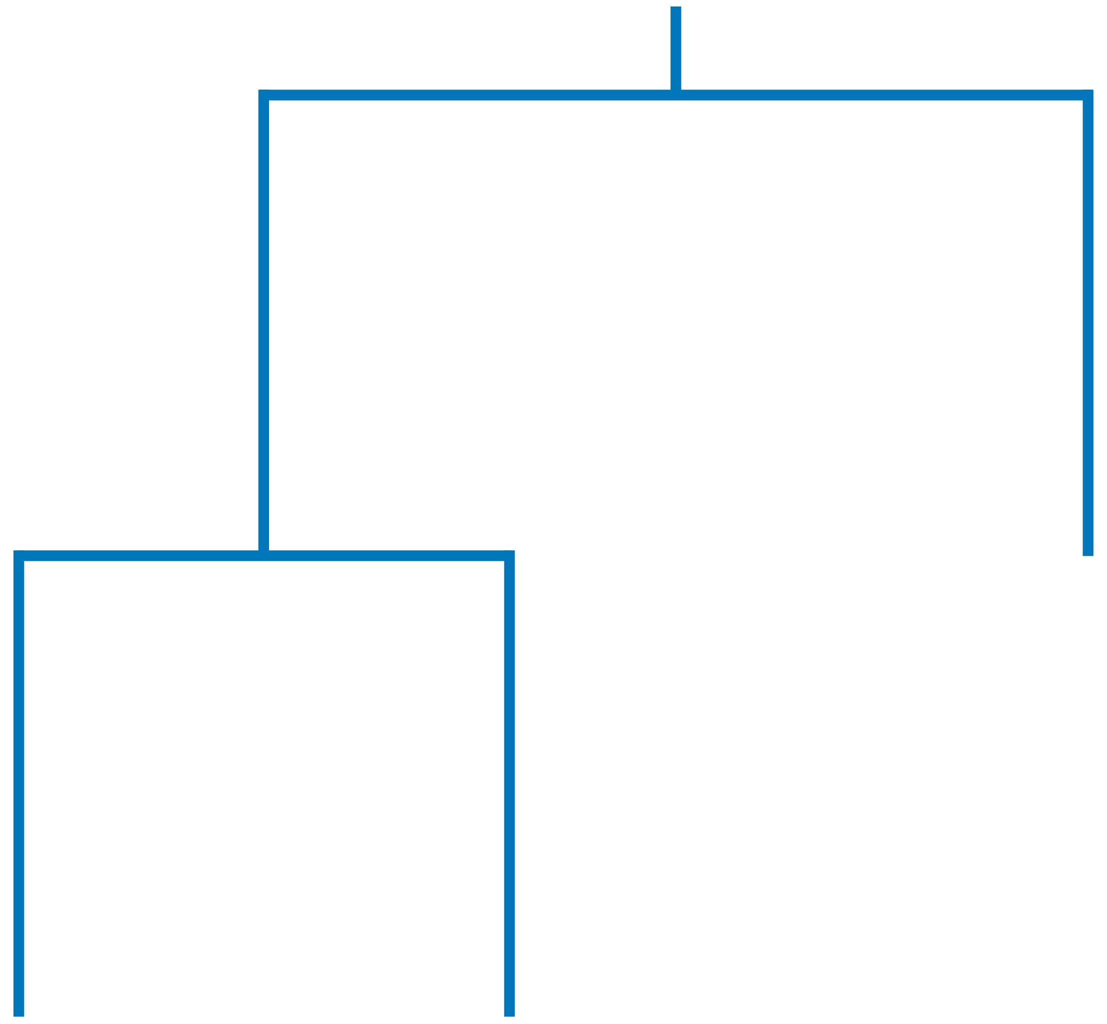


Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune "weakest link" splits.

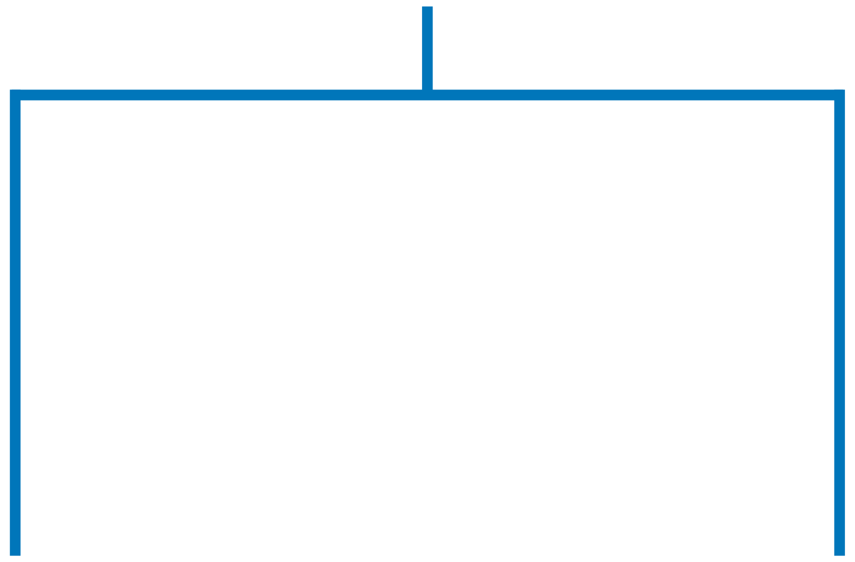


Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune "weakest link" splits.



Cost complexity pruning

Finding the sequence of trees T_α

Given a fully grown tree T_0, cost complexity pruning is an algorithm that finds optimal sequence T_α, which turns out to be nested.

Idea: Recursively prune “weakest link” splits.

Cross-validation

To find the optimal α for prediction

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α
- Split the training samples into K folds

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α
- Split the training samples into K folds
- For each fold k,

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α
- Split the training samples into K folds
- For each fold k,
    - grow a full tree T_0^{-k} on the out-of-fold data

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α
- Split the training samples into K folds
- For each fold k,
  - grow a full tree T_0^{-k} on the out-of-fold data
  - find the sequence of subtrees T_α^{-k} of T_0^{-k} by pruning (but using same α)

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α
- Split the training samples into K folds
- For each fold k,
    - grow a full tree T_0^{-k} on the out-of-fold data
    - find the sequence of subtrees T_α^{-k} of T_0^{-k} by pruning (but using same α)
    - using these trees, make predictions for each in-fold observation i and each α

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α
- Split the training samples into K folds
- For each fold k,
  - grow a full tree T_0^{-k} on the out-of-fold data
  - find the sequence of subtrees T_α^{-k} of T_0^{-k} by pruning (but using same α)
  - using these trees, make predictions for each in-fold observation i and each α
- Find CV estimates and standard errors as usual; choose $\widehat{\alpha}$ based on 1-standard-error rule

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α
- Split the training samples into K folds
- For each fold k,
  - grow a full tree T_0^{-k} on the out-of-fold data
  - find the sequence of subtrees T_α^{-k} of T_0^{-k} by pruning (but using same α)
  - using these trees, make predictions for each in-fold observation i and each α
- Find CV estimates and standard errors as usual; choose $\widehat{\alpha}$ based on 1-standard-error rule
- Output the final decision tree $T_{\widehat{\alpha}}$ (based on sequence of trees grown on full training data)

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α
- Split the training samples into K folds
- For each fold k,
  - grow a full tree T_0^{-k} on the out-of-fold data
  - find the sequence of subtrees T_α^{-k} of T_0^{-k} by pruning (but using same α)
  - using these trees, make predictions for each in-fold observation i and each α
- Find CV estimates and standard errors as usual; choose $\widehat{\alpha}$ based on 1-standard-error rule
- Output the final decision tree $T_{\hat{\alpha}}$ (based on sequence of trees grown on full training data)

Key insight: Cross-validating to find optimal α; trees with same α across CV folds may be different.

Cross-validation

To find the optimal α for prediction

- Grow a full tree T_0 on the whole training data
- Prune the tree to get a discrete sequence of trees T_α
- Split the training samples into K folds
- For each fold k,
  - grow a full tree T_0^{-k} on the out-of-fold data
  - find the sequence of subtrees T_α^{-k} of T_0^{-k} by pruning (but using same α)
  - using these trees, make predictions for each in-fold observation i and each α
- Find CV estimates and standard errors as usual; choose $\widehat{\alpha}$ based on 1-standard-error rule
- Output the final decision tree $T_{\widehat{\alpha}}$ (based on sequence of trees grown on full training data)

Key insight: Cross-validating to find optimal α; trees with same α across CV folds may be different.

Analogy with lasso: The variables selected for the same λ across different CV folds might be different.

Illustration: Tree growing and pruning

Step 1: Grow tree on whole training data in greedy fashion to get T_0.

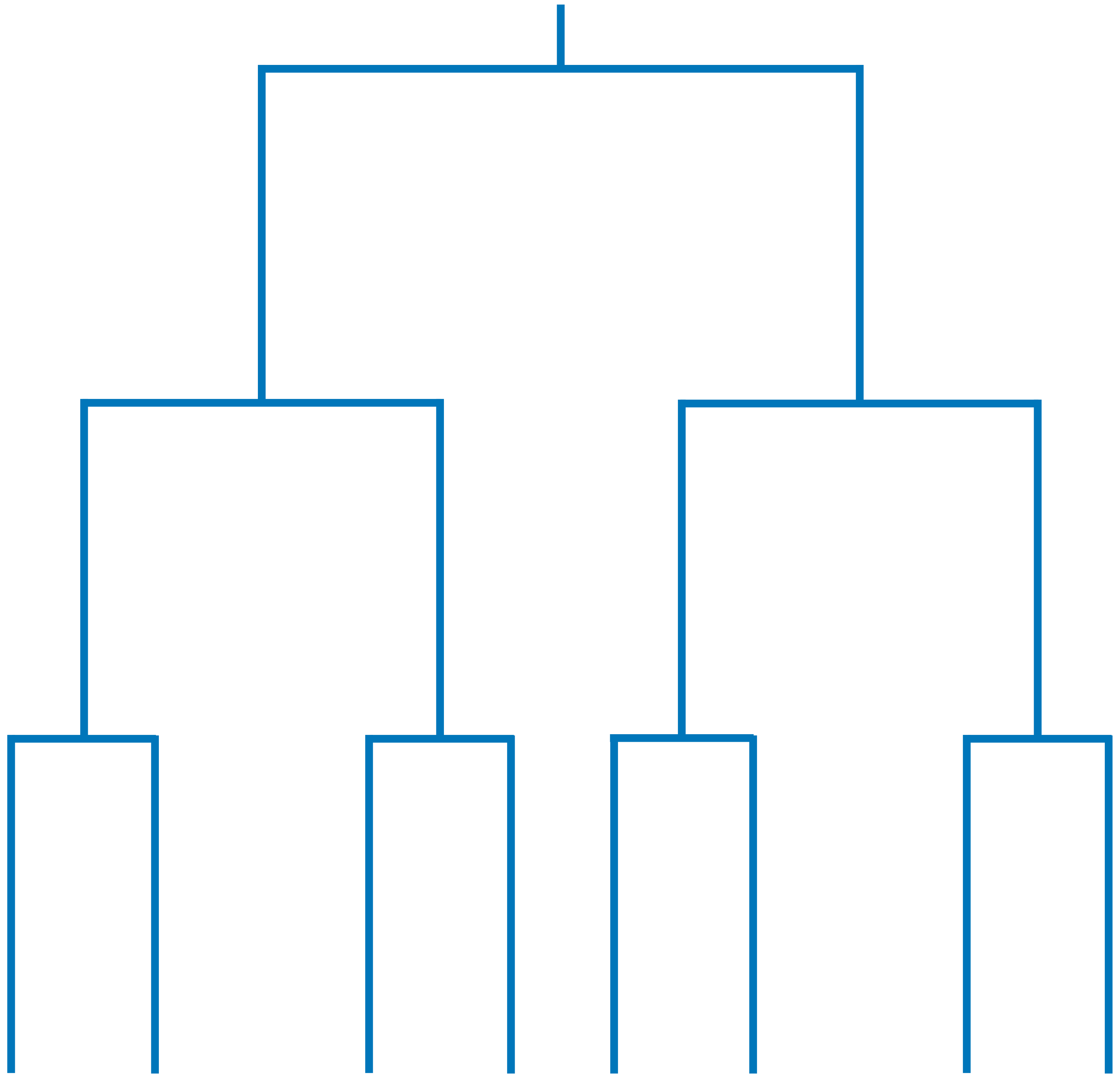
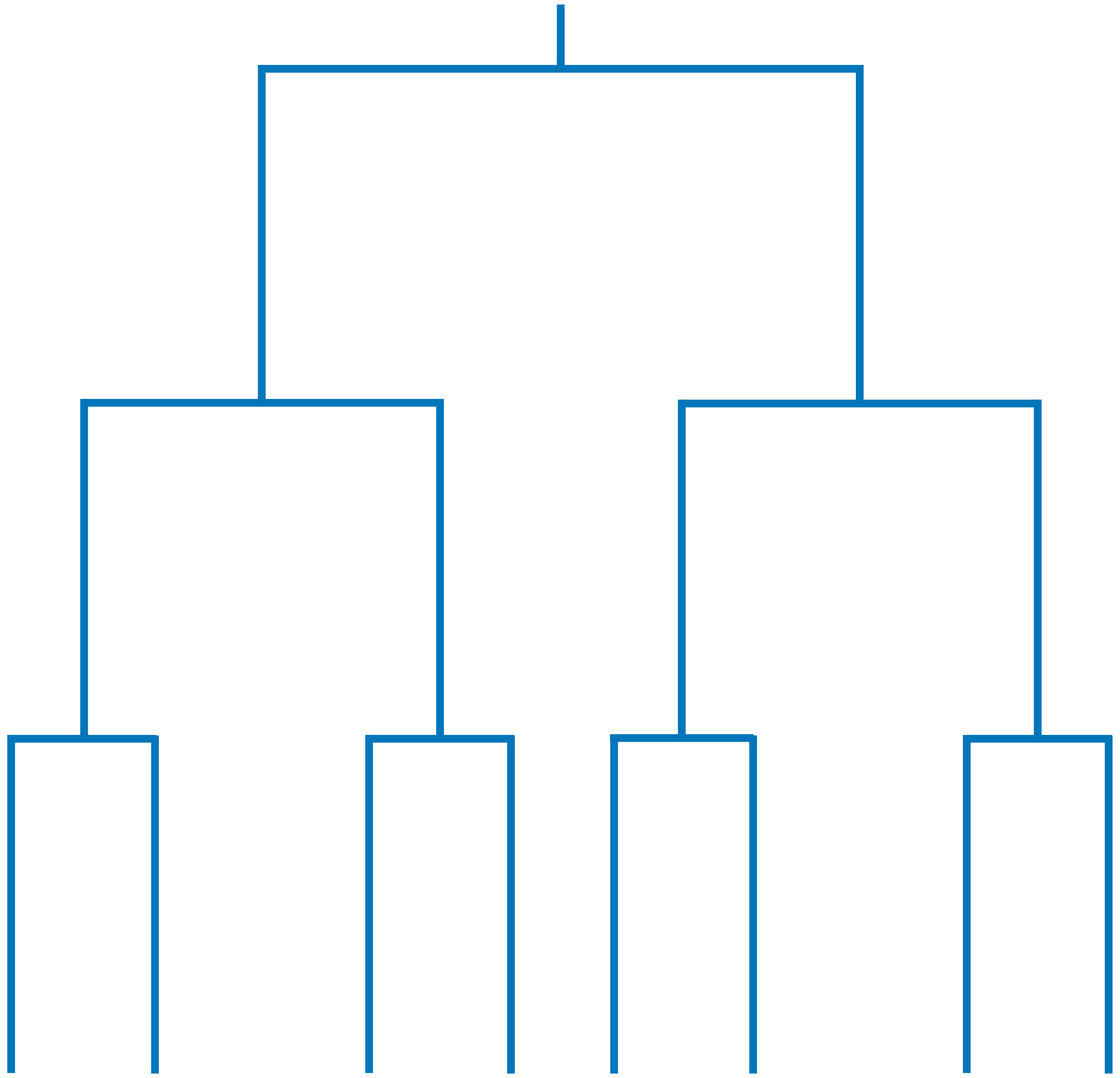


Illustration: Tree growing and pruning

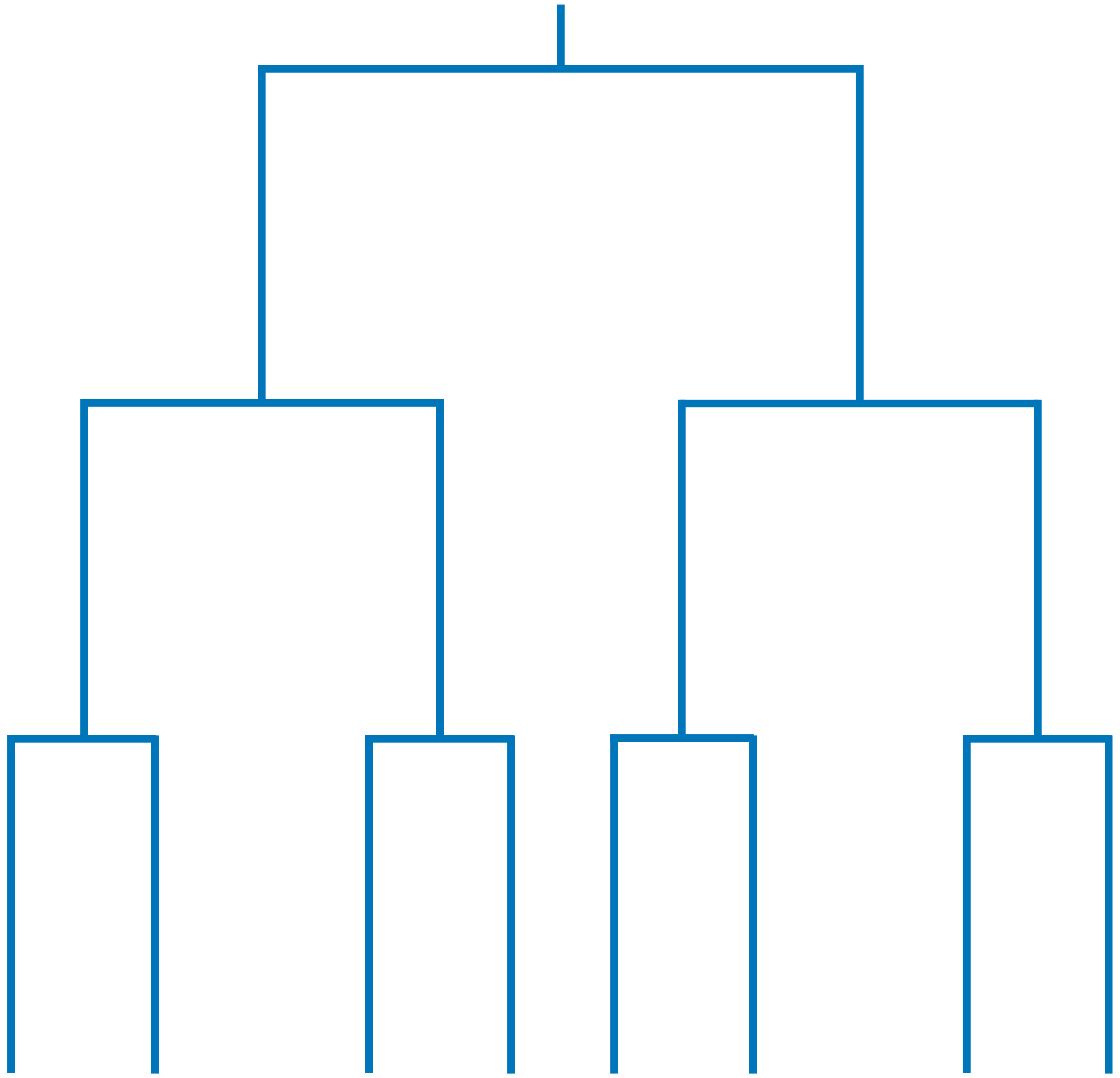
Step 2: Consider penalized objective function.



$$T_\alpha = \underset{T \subseteq T_0}{\arg\min} \left\{ \text{RSS}(T) + \alpha |T| \right\}$$

Illustration: Tree growing and pruning

Step 3: Sweep α from 0 to infinity, giving a nested sequence of trees.



$$T_\alpha = \underset{T \subseteq T_0}{\arg\min} \left\{ \text{RSS}(T) + \alpha |T| \right\}$$

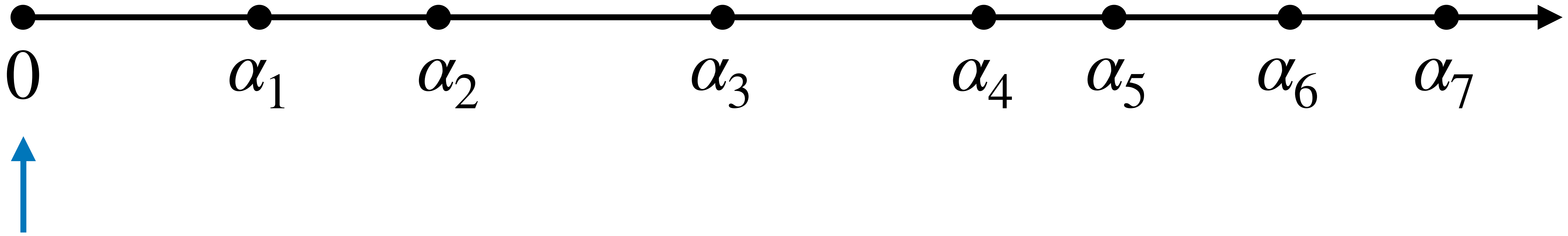


Illustration: Tree growing and pruning

Step 3: Sweep α from 0 to infinity, giving a nested sequence of trees.

$$T_\alpha = \underset{T \subseteq T_0}{\arg\min} \left\{ \mathsf{RSS}(T) + \alpha \, | \, T \, | \right\}$$

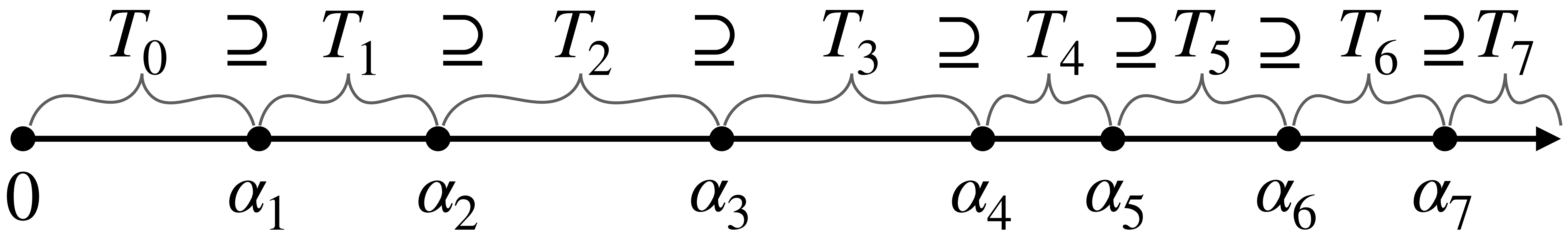

$T_0 \supseteq T_1 \supseteq T_2 \supseteq T_3 \supseteq T_4 \supseteq T_5 \supseteq T_6 \supseteq T_7$
0
α_1
α_2
α_3
α_4
α_5
α_6
α_7

Illustration: Tree growing and pruning

Step 4: Choose a representative value of α for each tree.

$$T_\alpha = \underset{T \subseteq T_0}{\arg\min} \left\{ \mathsf{RSS}(T) + \alpha |T| \right\}$$

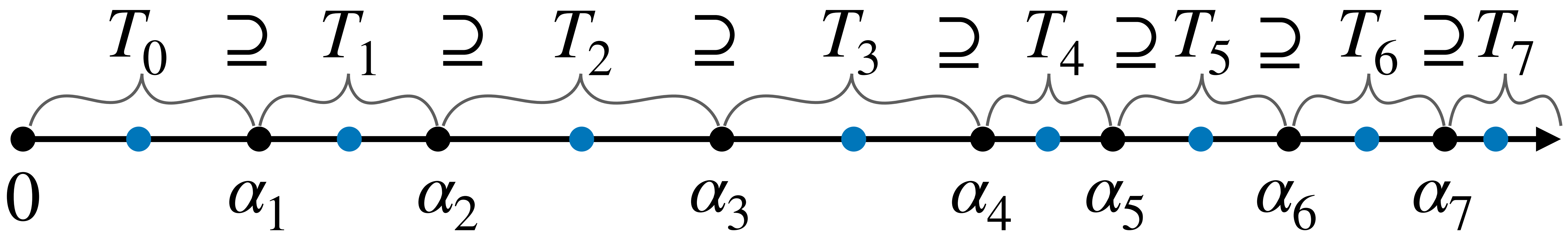
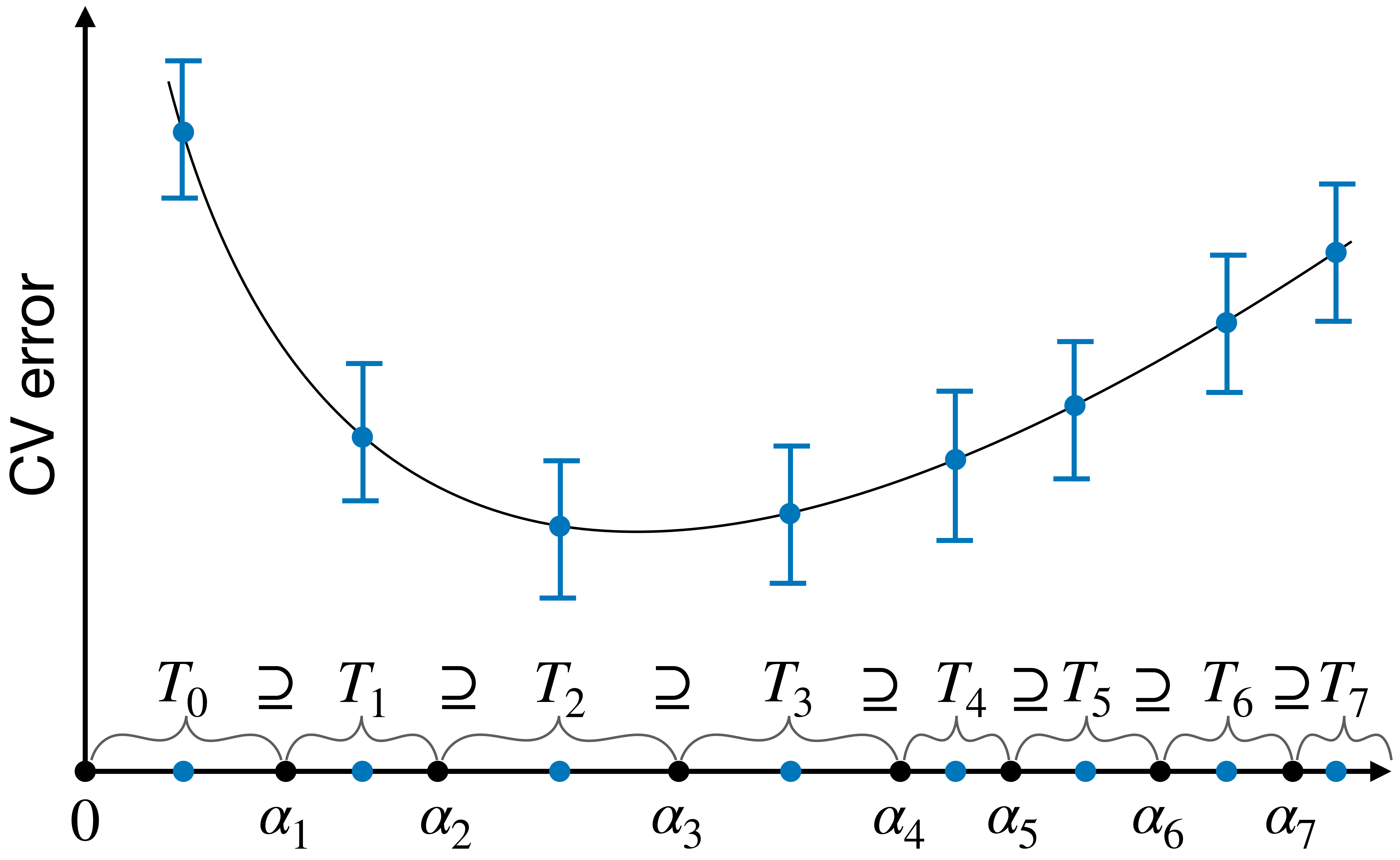

$T_0 \supseteq T_1 \supseteq T_2 \supseteq T_3 \supseteq T_4 \supseteq T_5 \supseteq T_6 \supseteq T_7$
0
α_1
α_2
α_3
α_4
α_5
α_6
α_7

Illustration: Tree growing and pruning

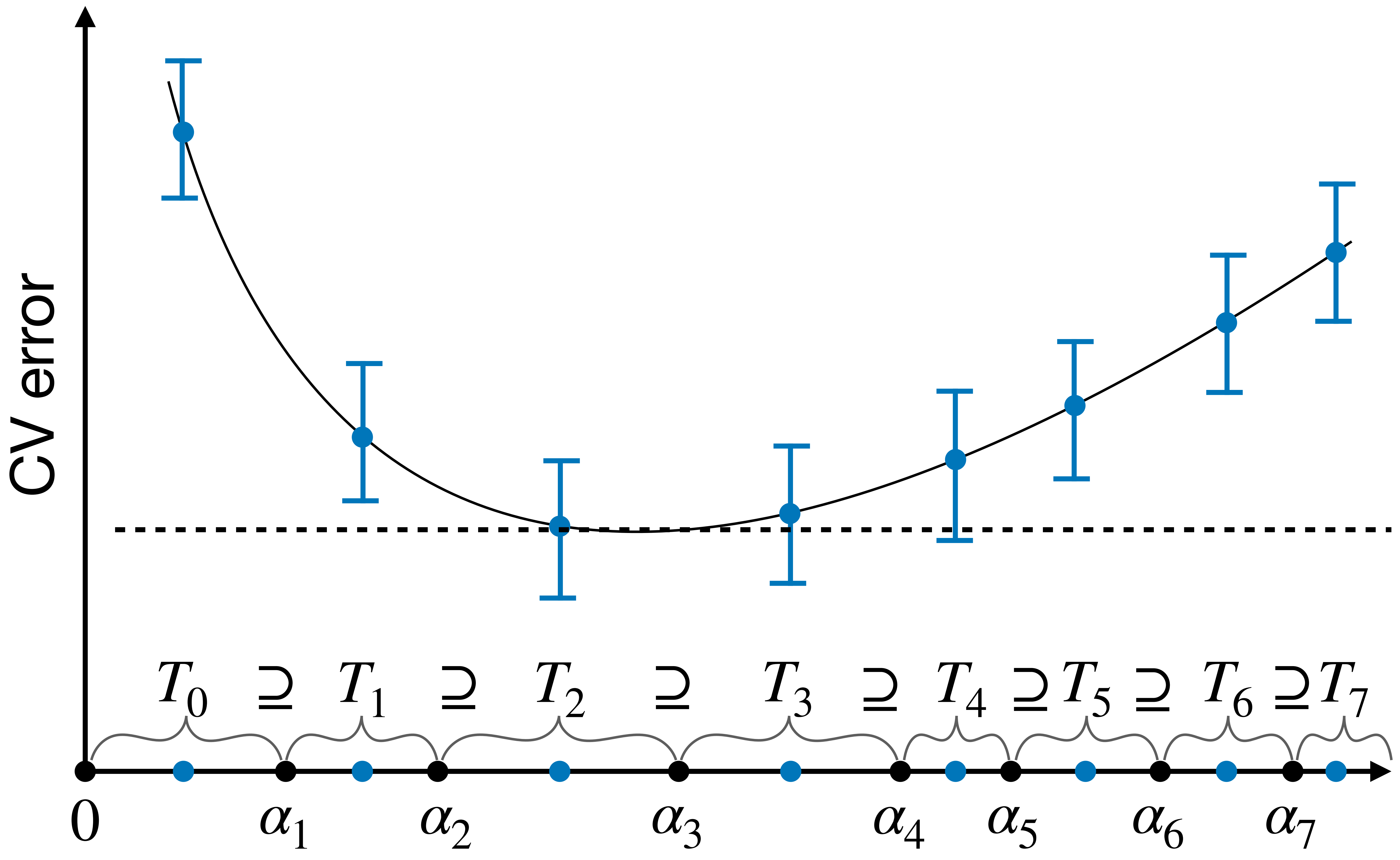
Step 5: Cross-validate over the representative values of α.



$$T_\alpha = \underset{T \subseteq T_0}{\arg\min} \left\{ \mathsf{RSS}(T) + \alpha \, | \, T \, | \right\}$$

Illustration: Tree growing and pruning

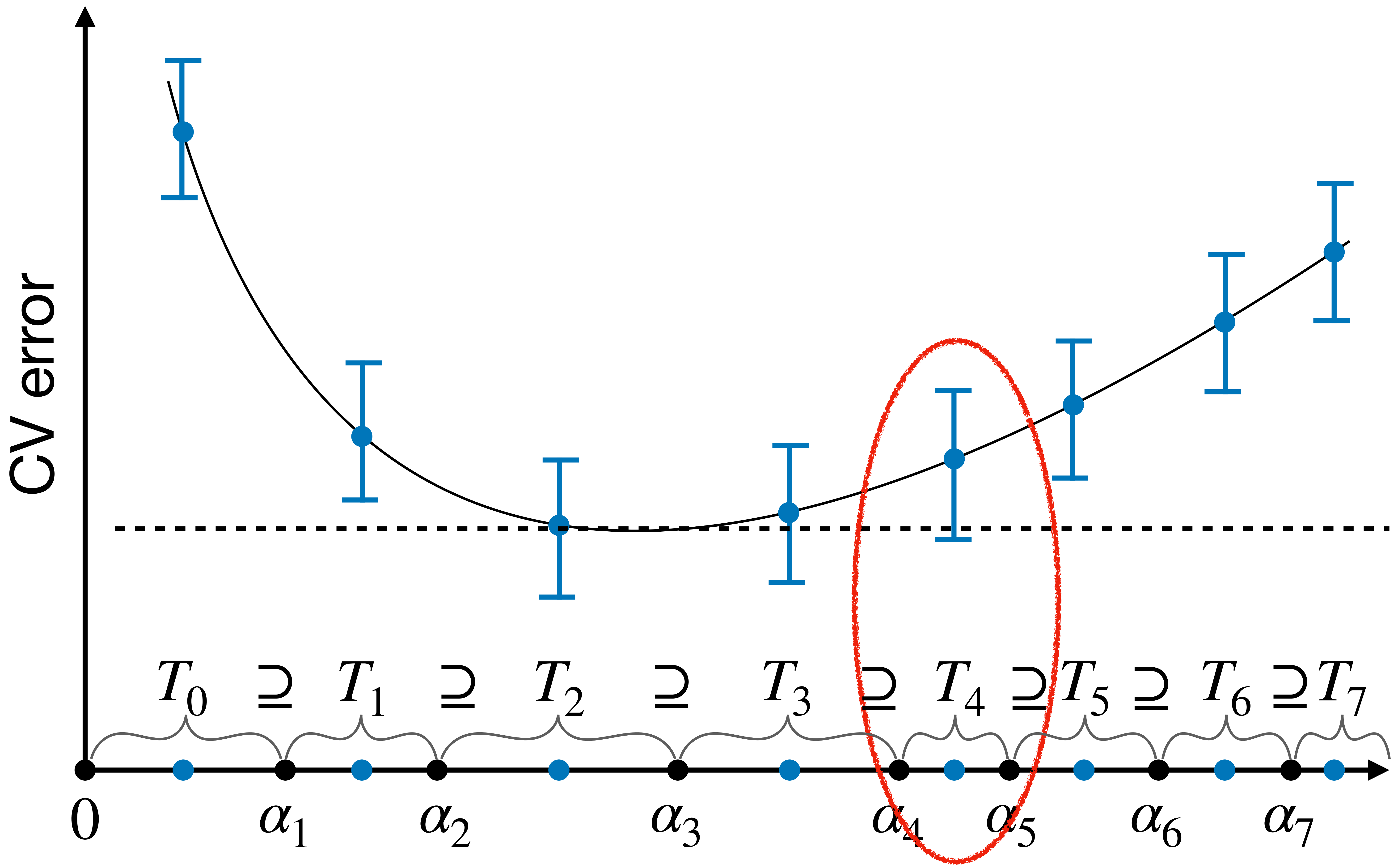
Step 6: Use one-standard-error rule to choose α.



$$T_\alpha = \underset{T \subseteq T_0}{\arg\min} \left\{ \mathsf{RSS}(T) + \alpha \,|\, T \,|\, \right\}$$

Illustration: Tree growing and pruning

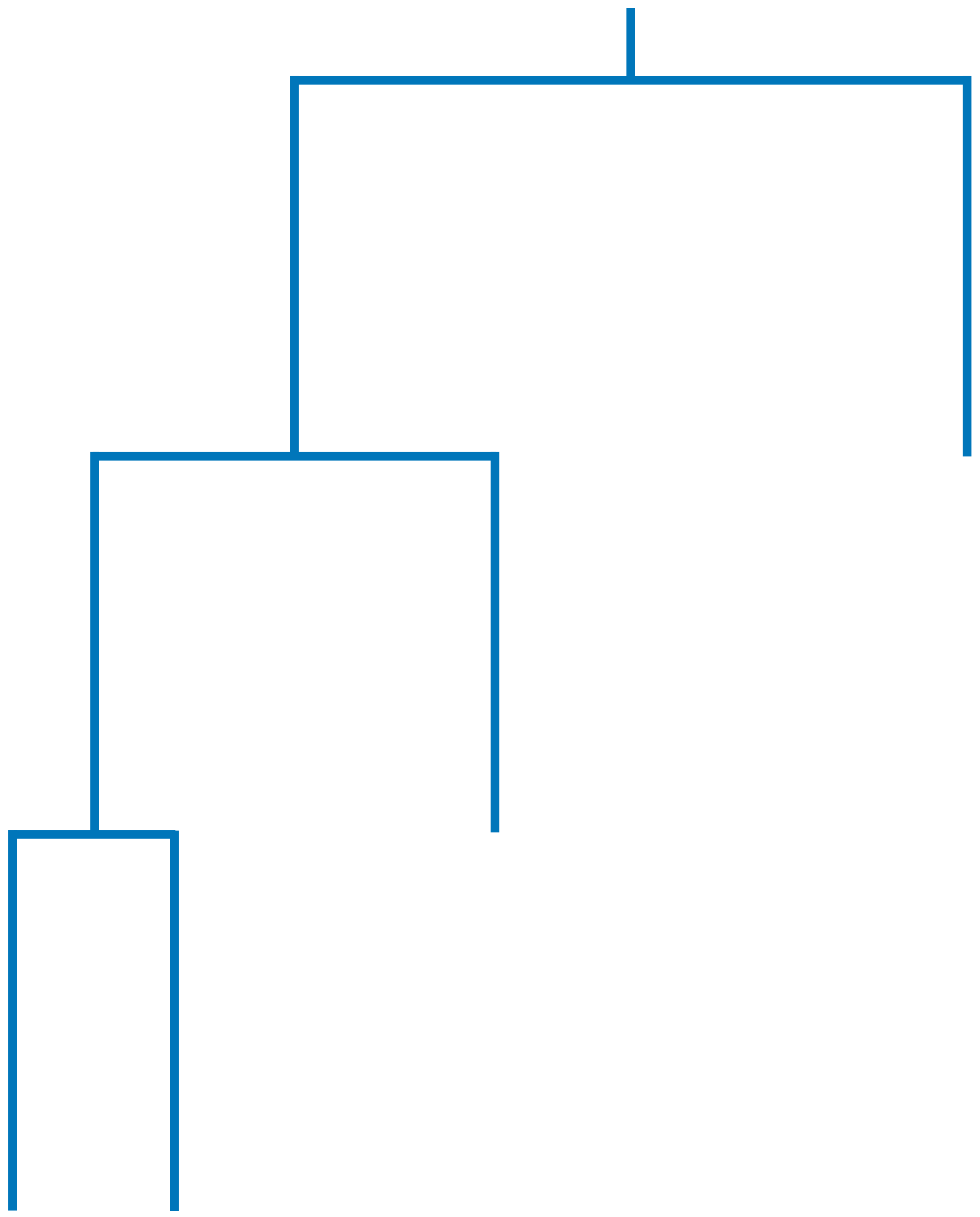
Step 6: Use one-standard-error rule to choose α.



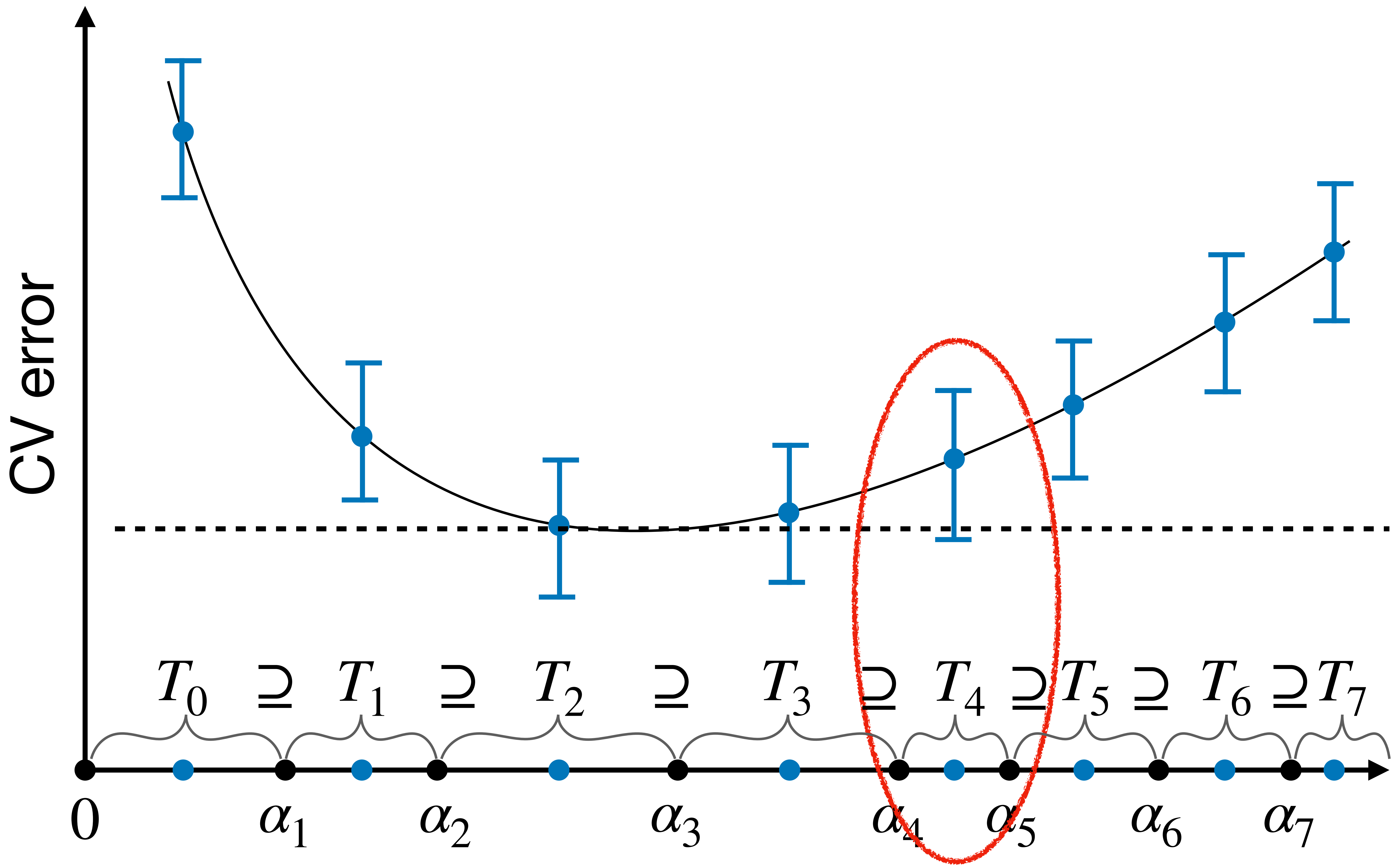
$$T_\alpha = \underset{T \subseteq T_0}{\arg\min} \left\{ \text{RSS}(T) + \alpha |T| \right\}$$

Illustration: Tree growing and pruning

Step 6: Use one-standard-error rule to choose α.



$$T_\alpha = \underset{T \subseteq T_0}{\arg\min} \left\{ \text{RSS}(T) + \alpha\,|\,T\,| \right\}$$



Tree growing versus tree pruning

Growing proceeds from smaller to larger trees; pruning from larger to smaller.

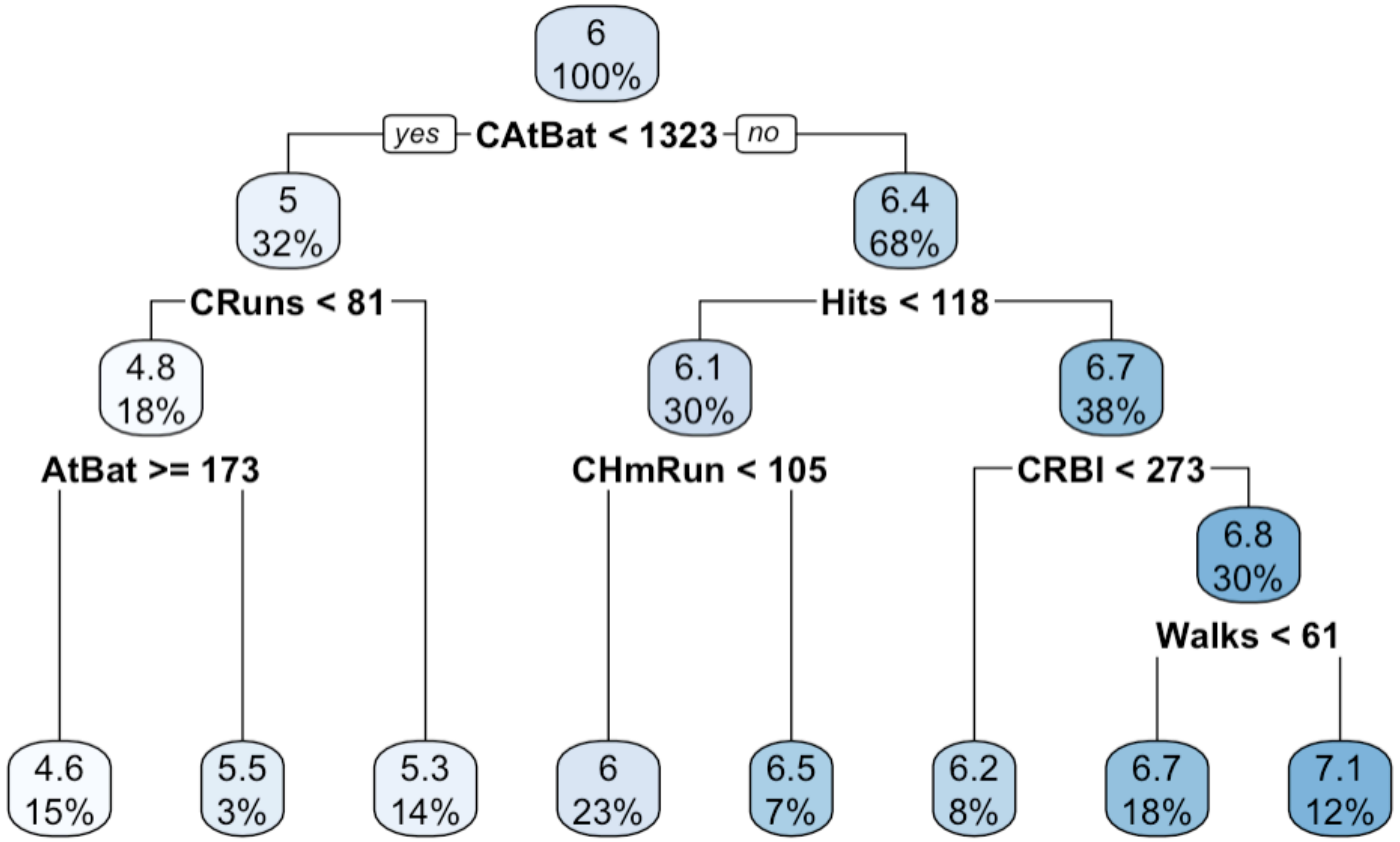
For regression trees, growing and pruning are both based on RSS.

For classification trees, growing based on Gini index but pruning based on misclassification error.

Growing and pruning both define a sequence of trees, but it may not be the same sequence. The sequence of trees for growing not used except to get the big tree T_0; the sequence of trees for pruning is the one used for cross-validation.

Summary: Decision Trees

- Nonlinear method for regression or classification based on recursive partitioning of feature space
- Trees grown in greedy top-down fashion, choosing feature and split point to maximally improve purity of terminal nodes.
- The complexity of a tree increases with the number of terminal nodes.
- A sequence of trees of varying complexities obtained from cost complexity pruning of a maximally-grown tree.
- Final tree chosen by cross-validation on penalty parameter α.



Pros	Cons
Easily interpretable	Tree structure very sensitive to training data
Captures non-linear relationships	High variance predictions; suboptimal predictive performance

How can we reduce the variance of trees?

When it comes to prediction accuracy, trees suffer because of their high variance.

Here's an idea for how we can obtain a prediction method with lower variance:

- "Shake up" the training data lots of times (bootstrap)
- For each version of the training data, fit a different tree
- Use the average of all these trees to make predictions (aggregation)

Bagging = Bootstrap Aggregation.

Intuition: By averaging a bunch of trees, we are reducing the variance while keeping the bias about the same. This should yield better predictive performance!

What does it mean to “shake up” the training data?

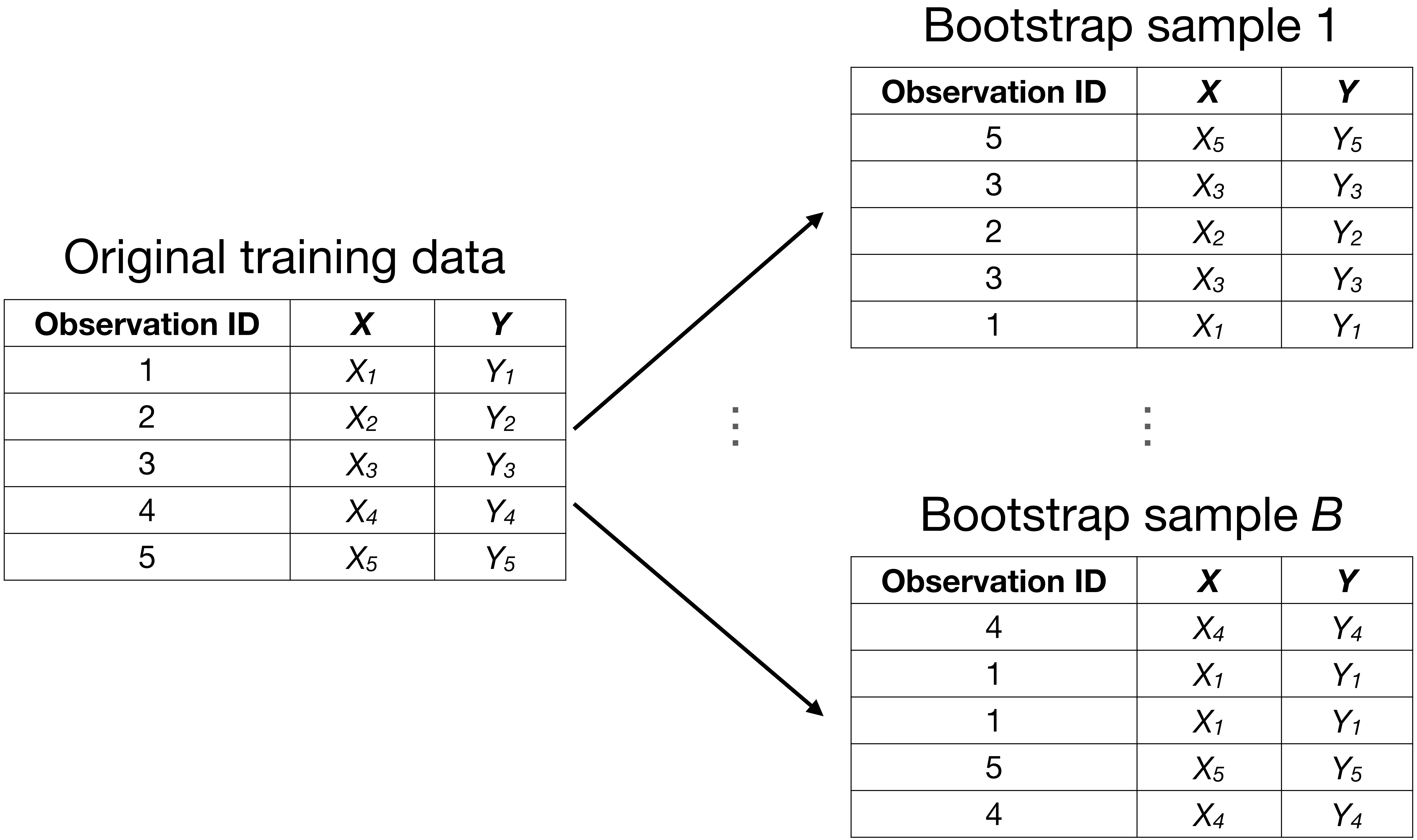
What we ideally would have wanted is to get many different random realizations of the training data, on which we could fit different trees.

We only get one realization of the training data, but we can still get different random versions of our data by bootstrapping:

A bootstrap sample is a new data set with the same number of observations, generating by sampling observations from the original data with replacement.

The idea is that your bootstrap samples are slightly different versions of your training data, allowing you to fit different trees to these different training sets.

The bootstrap: An example



Original training data

Observation ID	$\boldsymbol{X}$	$\boldsymbol{Y}$
1	X_1	Y_1
2	X_2	Y_2
3	X_3	Y_3
4	X_4	Y_4
5	X_5	Y_5

Bootstrap sample 1

Observation ID	$\boldsymbol{X}$	$\boldsymbol{Y}$
5	X_5	Y_5
3	X_3	Y_3
2	X_2	Y_2
3	X_3	Y_3
1	X_1	Y_1

⋮

⋮

Bootstrap sample *B*

Observation ID	$\boldsymbol{X}$	$\boldsymbol{Y}$
4	X_4	Y_4
1	X_1	Y_1
1	X_1	Y_1
5	X_5	Y_5
4	X_4	Y_4

Bagging

Original training data

Obs ID	*X*	*Y*
1	X_1	Y_1
2	X_2	Y_2
3	X_3	Y_3
4	X_4	Y_4
5	X_5	Y_5

Bootstrap sample 1

Obs ID	*X*	*Y*
5	X_5	Y_5
3	X_3	Y_3
2	X_2	Y_2
3	X_3	Y_3
1	X_1	Y_1

Bootstrap sample *B*

Obs ID	*X*	*Y*
4	X_4	Y_4
1	X_1	Y_1
1	X_1	Y_1
5	X_5	Y_5
4	X_4	Y_4

Regression:

$$\hat{f}(X) = \frac{1}{B}\sum_{b=1}^{B} T^{*b}(X)$$

Classification:

$$\hat{f}(X) = \text{mode}(\{T^{*b}(X)\}_{b=1}^{B})$$