Tree pruning and bagging

STAT 471

November 4, 2021

Where we are

- ✓ **Unit 1:** Intro to modern data mining
- ✓ **Unit 2:** Tuning predictive models
- ✓ **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

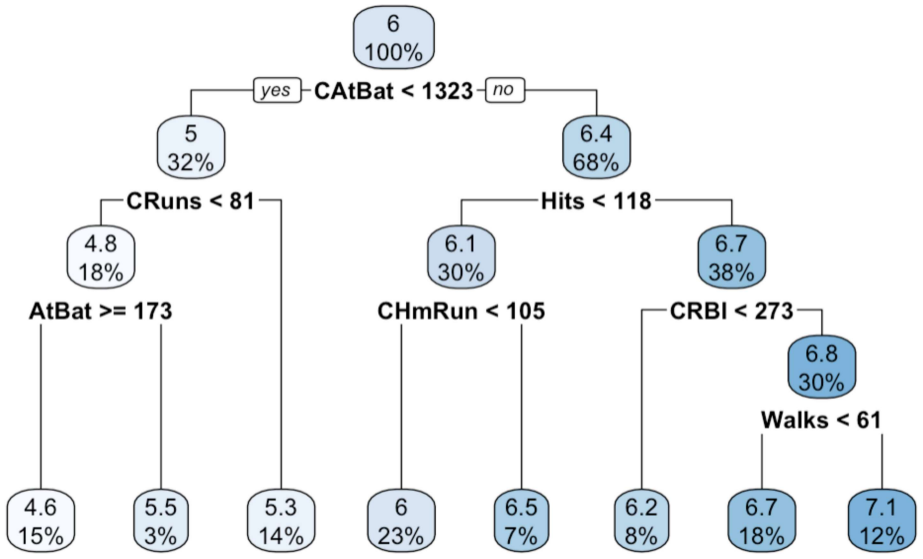
Lecture 4: Boosting

Lecture 5: Unit review and quiz in class

Homework 4 due the following Wednesday.

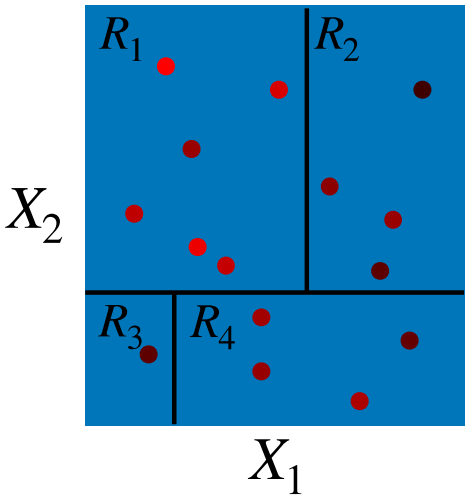
Recall: Decision trees

- 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

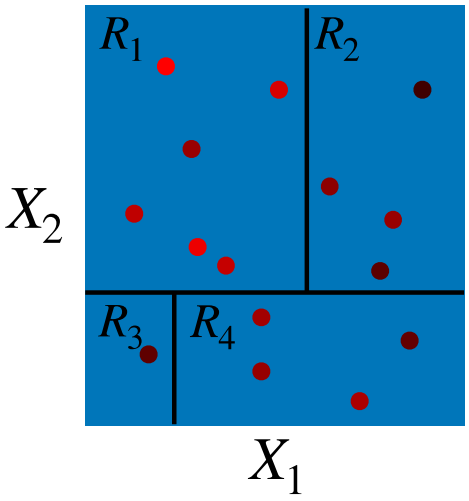
Regression tree



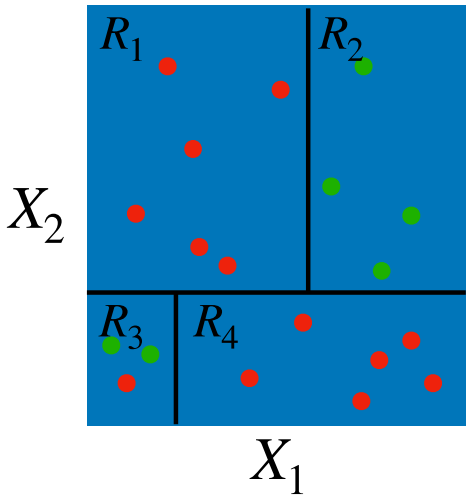
Regression: Total RSS $= \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i \in R_1} (Y_i - \hat{Y}_i)^2 + \sum_{i \in R_2} (Y_i - \hat{Y}_i)^2 + \sum_{i \in R_3} (Y_i - \hat{Y}_i)^2 + \sum_{i \in R_4} (Y_i - \hat{Y}_i)^2$

Clarifications from last time

Regression tree



Classification tree

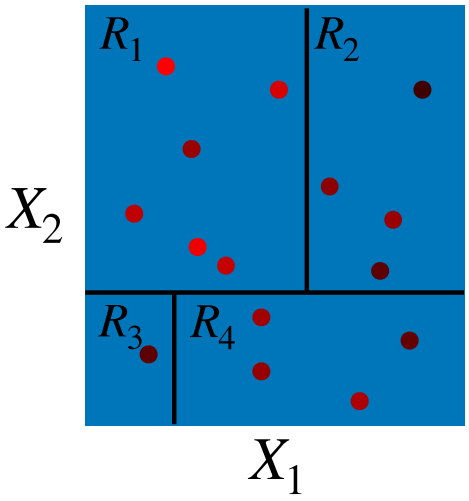


Regression: Total RSS $= \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i \in R_1} (Y_i - \hat{Y}_i)^2 + \sum_{i \in R_2} (Y_i - \hat{Y}_i)^2 + \sum_{i \in R_3} (Y_i - \hat{Y}_i)^2 + \sum_{i \in R_4} (Y_i - \hat{Y}_i)^2$

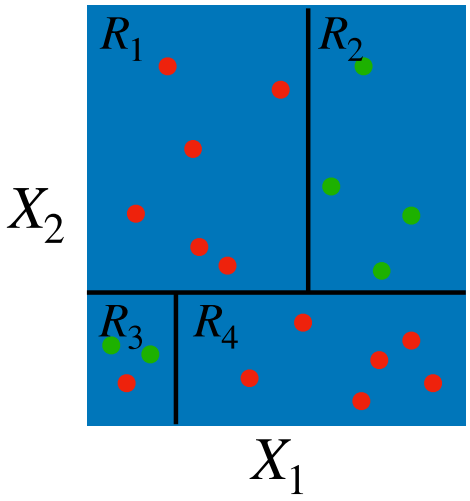
Classification: Total Gini $= n_1 \cdot 2\hat{p}_1(1 - \hat{p}_1) + n_2 \cdot 2\hat{p}_2(1 - \hat{p}_2) + n_3 \cdot 2\hat{p}_3(1 - \hat{p}_3) + n_4 \cdot 2\hat{p}_4(1 - \hat{p}_4)$

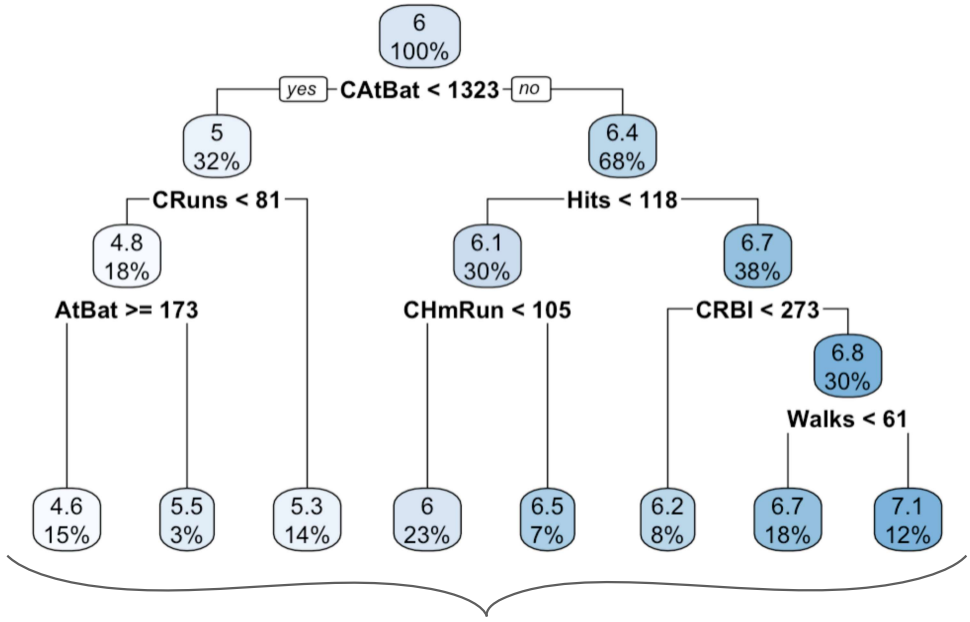
Clarifications from last time

Regression tree



Classification tree





Leaf nodes or terminal nodes (equivalent)

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

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

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.

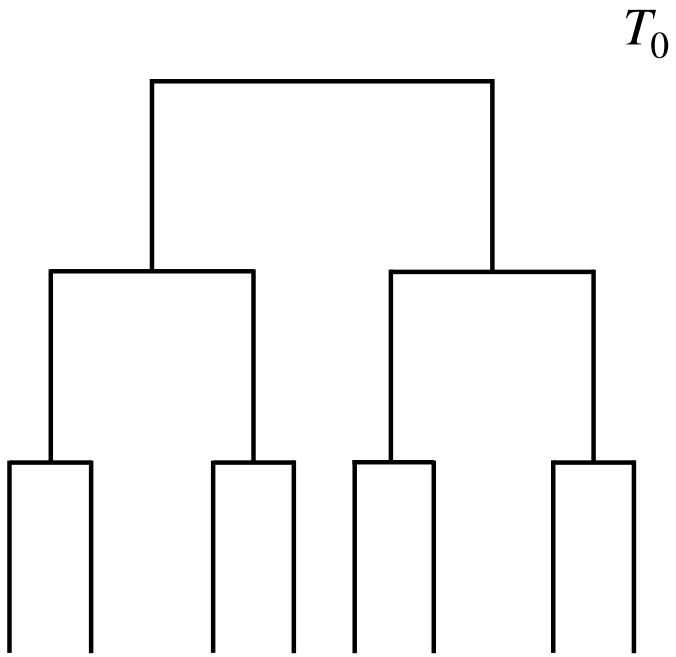
↑ terminal nodes

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.

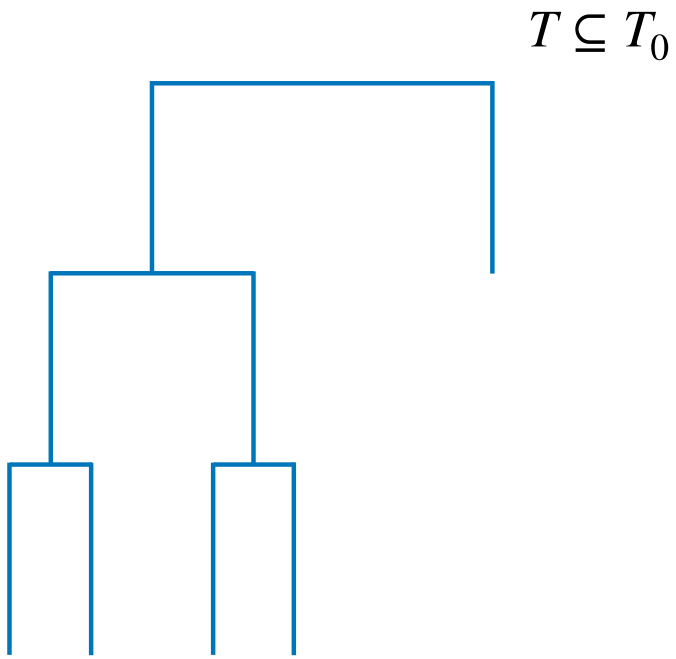


A family of decision trees of varying complexity

- 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.

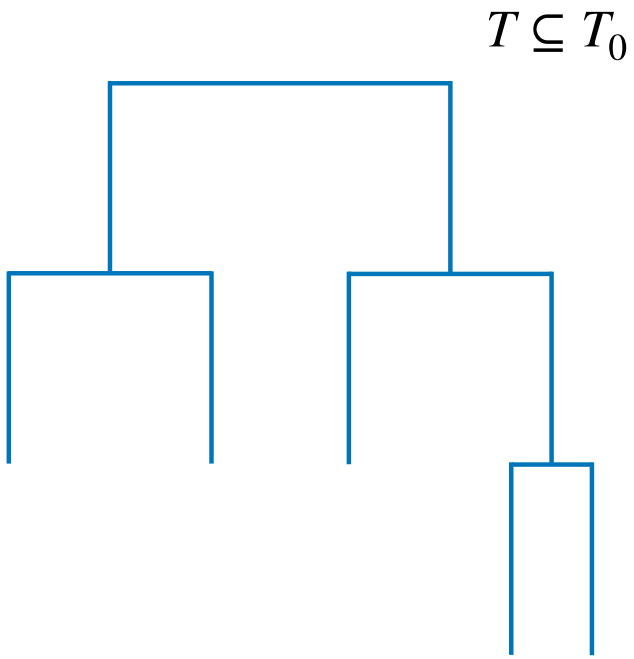


A family of decision trees of varying complexity

- 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.

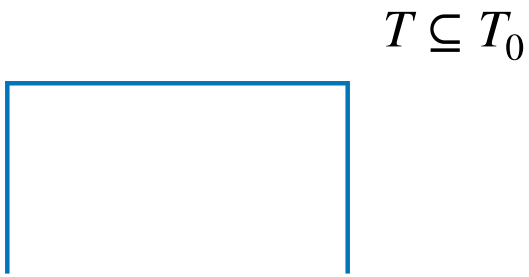


A family of decision trees of varying complexity

- 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.

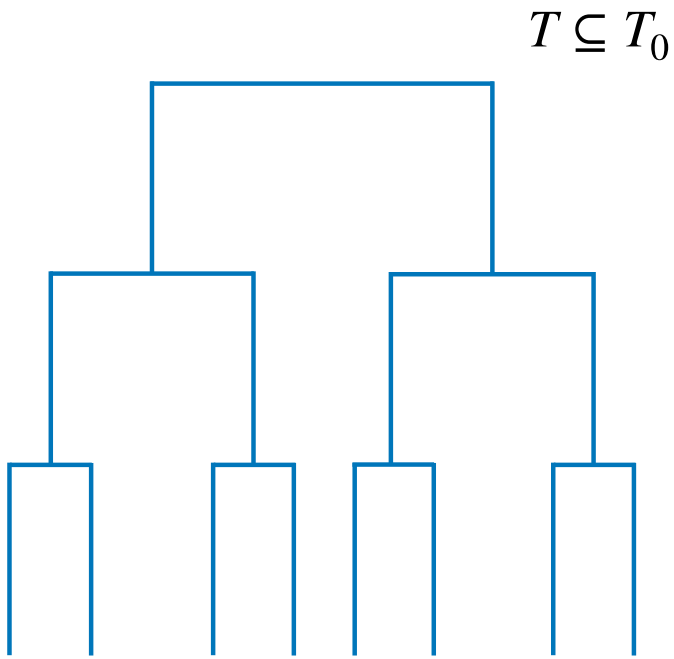


A family of decision trees of varying complexity

- 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

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

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

Interpretation of α: RSS should decrease by at least α to make a split worth it.

Unlike lasso, there is a discrete set of α values $\alpha_1 < \alpha_2 < \cdots < \alpha_M$ giving all possible solutions as α varies: T_{α_m} is the tree with m terminal nodes with lowest RSS.

Imposing a penalty for terminal nodes

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

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

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

~~Interpretation of α: RSS should decrease by at least α to make a split worth it.~~

Unlike lasso, there is a discrete set of α values $\alpha_1 < \alpha_2 < \cdots < \alpha_M$ giving all possible solutions as α varies: T_{α_m} is the tree with m terminal nodes with lowest RSS.

Imposing a penalty for terminal nodes

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

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

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

~~Interpretation of α: RSS should decrease by at least α to make a split worth it.~~

Unlike lasso, there is a discrete set of α values ~~$\alpha_1 < \alpha_2 < \cdots < \alpha_M$~~ giving all possible solutions as α varies: T_{α_m} is the tree with m terminal nodes with lowest RSS.

Imposing a penalty for terminal nodes

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

$$T_\alpha = \arg\min_{T \subseteq T_0} \text{RSS}(T) + \alpha |T|.$$

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

~~Interpretation of α: RSS should decrease by at least α to make a split worth it.~~

Unlike lasso, there is a discrete set of α values ~~$\alpha_1 < \alpha_2 < \cdots < \alpha_M$~~ giving all possible solutions as α varies: ~~T_{α_m} is the tree with m terminal nodes with lowest RSS.~~

Cost complexity pruning

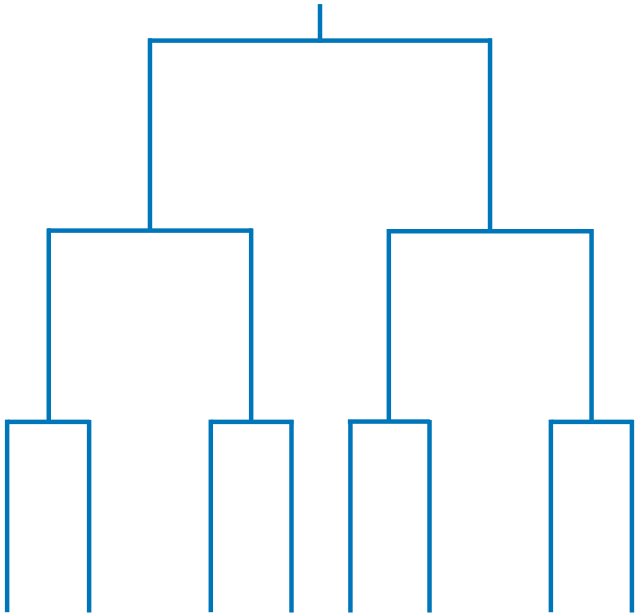
Finding the sequence of trees T_{α_m}

Given a fully grown tree T_0, cost complexity pruning is an algorithm that provably finds optimal sequence

$$T_0 \supseteq T_{\alpha_1} \supseteq T_{\alpha_2} \supseteq \cdots \supseteq T_{\alpha_M},$$

which turns out to be nested:

- Set $T_{\alpha_1} = T_0$, the fully grown tree
- Find the weakest link: the split in T_{α_1} that reduces cost (i.e. RSS or misclassification error) the least.
- Set T_{α_2} equal to T_{α_1} but without the weakest link
- Repeatedly remove weakest link until arriving at T_{α_M}, the tree with one terminal node.



Cost complexity pruning

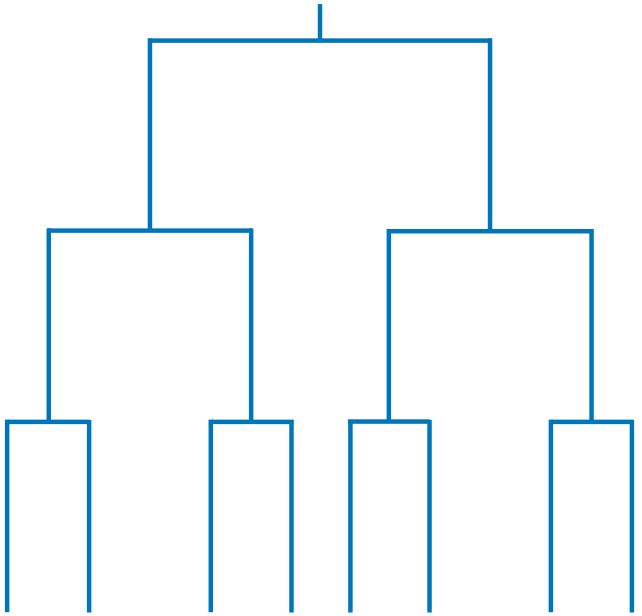
Finding the sequence of trees T_{α_m}

Given a fully grown tree T_0, cost complexity pruning is an algorithm that provably finds optimal sequence

$$T_0 \supseteq T_{\alpha_1} \supseteq T_{\alpha_2} \supseteq \cdots \supseteq T_{\alpha_M},$$

which turns out to be nested:

- Set $T_{\alpha_1} = T_0$, the fully grown tree
- Find the weakest link: the split in T_{α_1} that reduces cost (i.e. RSS or misclassification error) the least.
- Set T_{α_2} equal to T_{α_1} but without the weakest link
- Repeatedly remove weakest link until arriving at T_{α_M}, the tree with one terminal node.



Cost complexity pruning

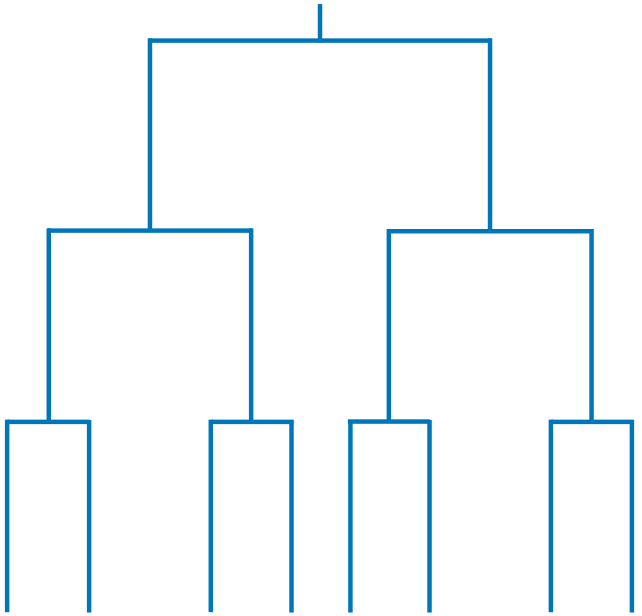
Finding the sequence of trees T_{α_m}

Given a fully grown tree T_0, cost complexity pruning is an algorithm that provably finds optimal sequence

$$T_0 \supseteq T_{\alpha_1} \supseteq T_{\alpha_2} \supseteq \cdots \supseteq T_{\alpha_M},$$

which turns out to be nested:

- Set $T_{\alpha_1} = T_0$, the fully grown tree
- Find the weakest link: the split in T_{α_1} that reduces cost (i.e. RSS or misclassification error) the least.
- Set T_{α_2} equal to T_{α_1} but without the weakest link
- Repeatedly remove weakest link until arriving at T_{α_M}, the tree with one terminal node.



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 sequence $T_0 \supseteq T_{\alpha_1} \supseteq T_{\alpha_2} \supseteq \cdots \supseteq T_{\alpha_M}$
- 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 $T_0^{-k} \supseteq T_{\alpha_1}^{-k} \supseteq T_{\alpha_2}^{-k} \supseteq \cdots \supseteq T_{\alpha_M}^{-k}$ by pruning (but using same α_m)
  - using these trees, make predictions $\widehat{Y}_i^m$ for each in-fold observation i and each m
- Find CV estimates and standard errors as usual; choose $\widehat{\alpha}$ based on 1-standard-error rule
- Output the final decision tree $T_{\widehat{\alpha}}$

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 sequence $T_0 \supseteq T_{\alpha_1} \supseteq T_{\alpha_2} \supseteq \cdots \supseteq T_{\alpha_M}$
- 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 $T_0^{-k} \supseteq T_{\alpha_1}^{-k} \supseteq T_{\alpha_2}^{-k} \supseteq \cdots \supseteq T_{\alpha_M}^{-k}$ by pruning (but using same α_m)
  - using these trees, make predictions $\widehat{Y}_i^m$ for each in-fold observation i and each m
- Find CV estimates and standard errors as usual; choose $\widehat{\alpha}$ based on 1-standard-error rule
- Output the final decision tree $T_{\widehat{\alpha}}$

Illustration: Tree growing and pruning (new)

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

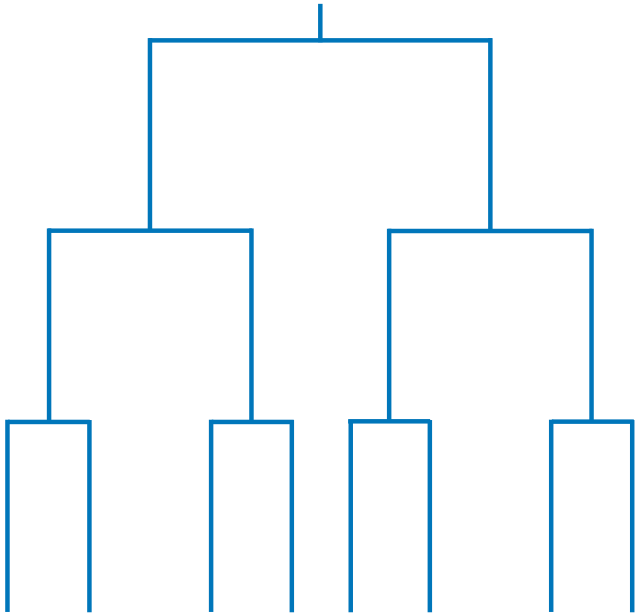
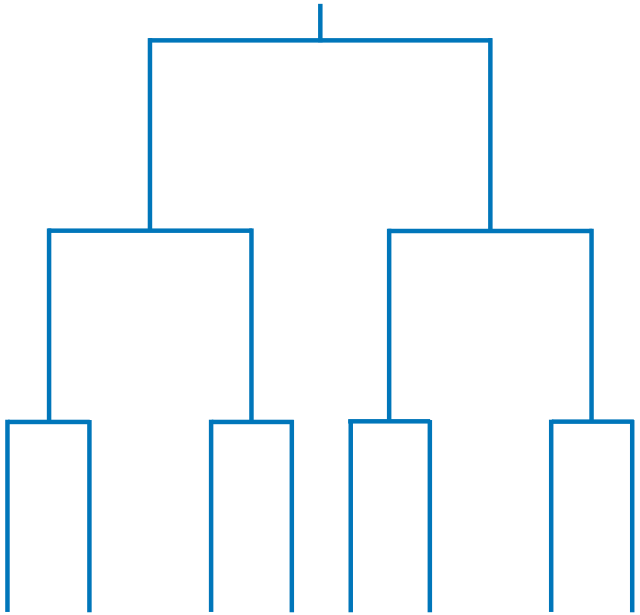


Illustration: Tree growing and pruning (new)

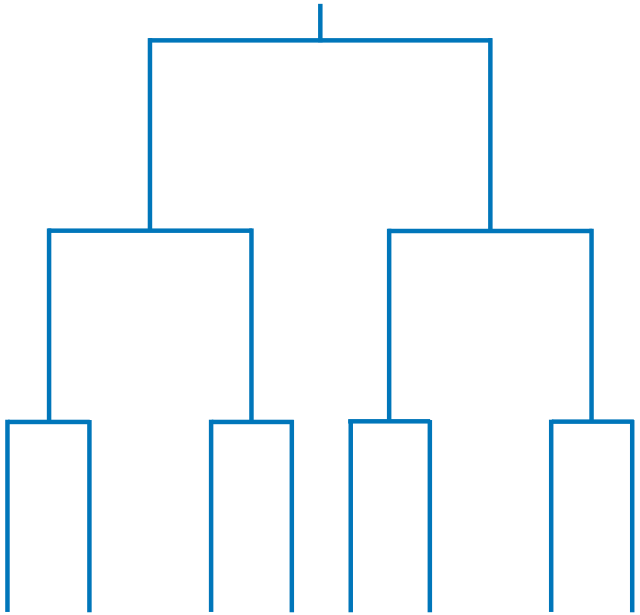
Step 2: Consider penalized objective function.



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

Illustration: Tree growing and pruning (new)

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



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

when alpha equals 0, you get the largest trees

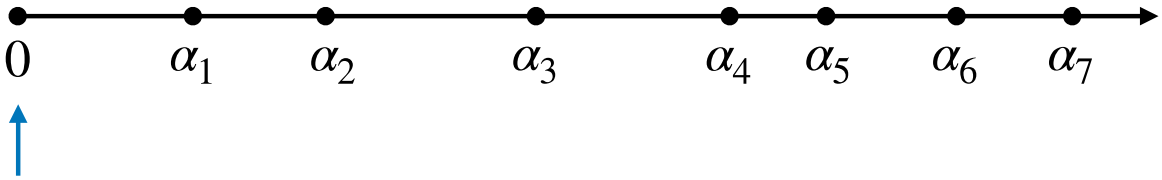


Illustration: Tree growing and pruning (new)

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

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

Most trimmed off all the trees

Lot of bias, and less variance

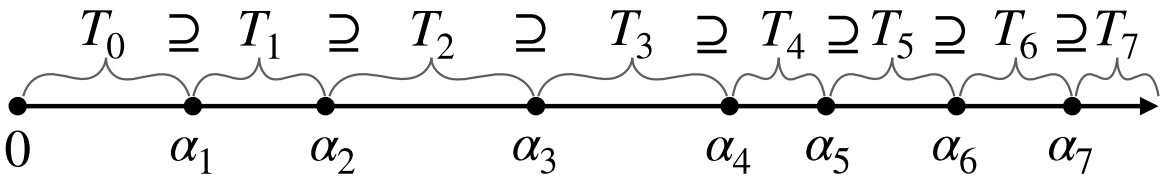


Illustration: Tree growing and pruning (new)

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

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

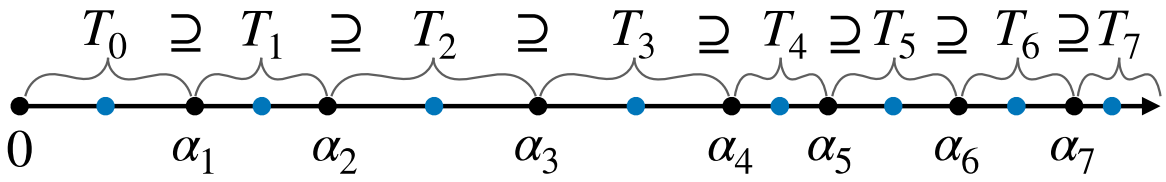


Illustration: Tree growing and pruning (new)

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

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

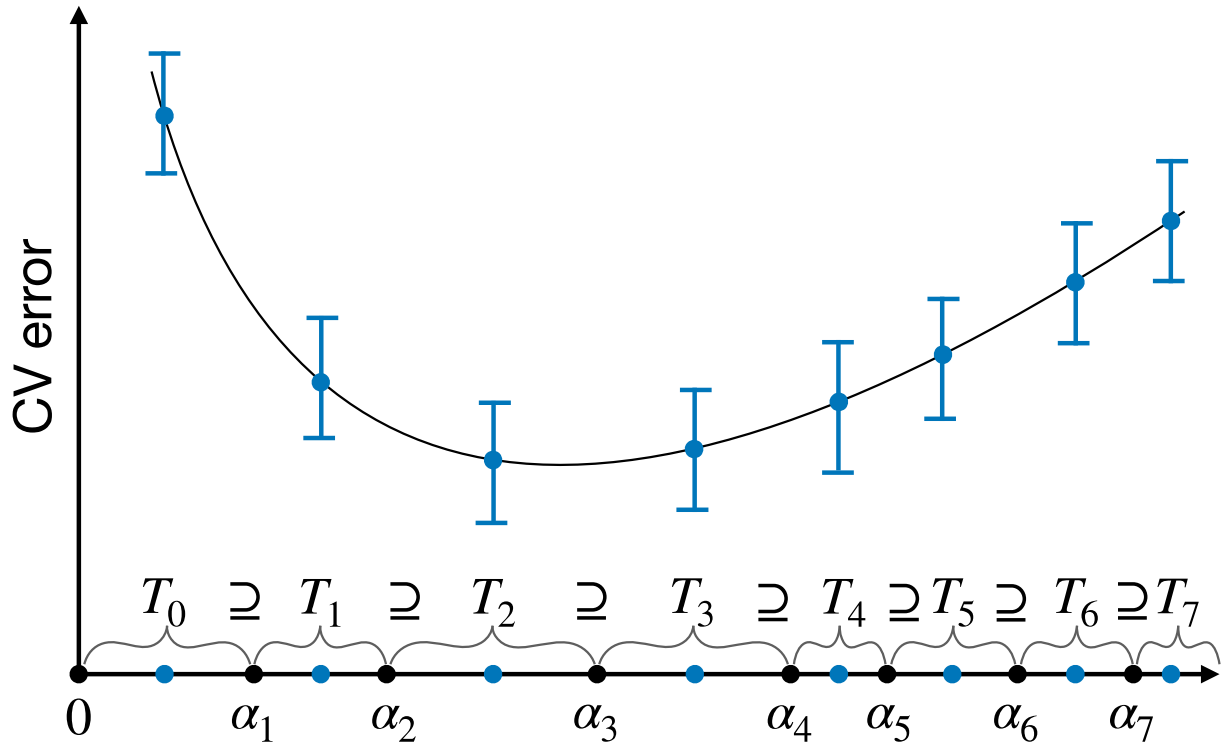
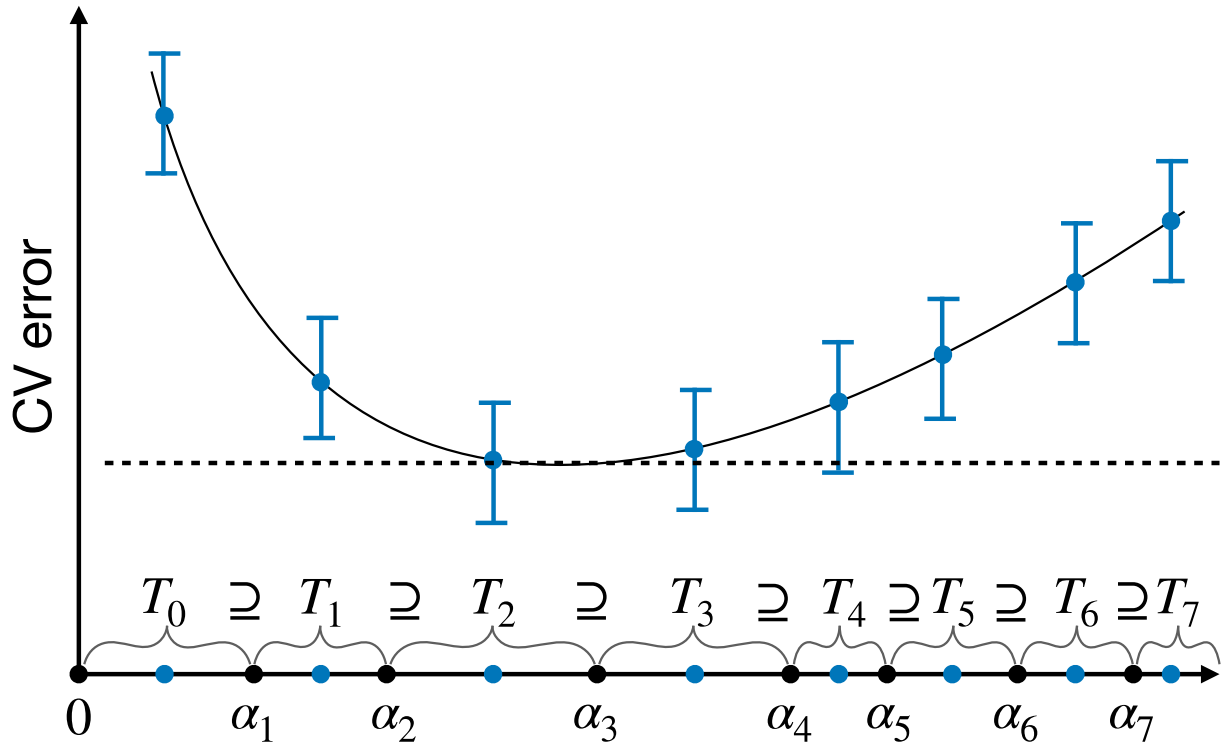


Illustration: Tree growing and pruning (new)

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

For every cross validation, have to regrow and prune

Based on one standard error model, we need to move to the right



$$T_\alpha = \arg\min_{T \subseteq T_0} \text{RSS}(T) + \alpha |T|$$

Illustration: Tree growing and pruning (new)

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

$$T_\alpha = \arg\min_{T \subseteq T_0} \text{RSS}(T) + \alpha |T|$$

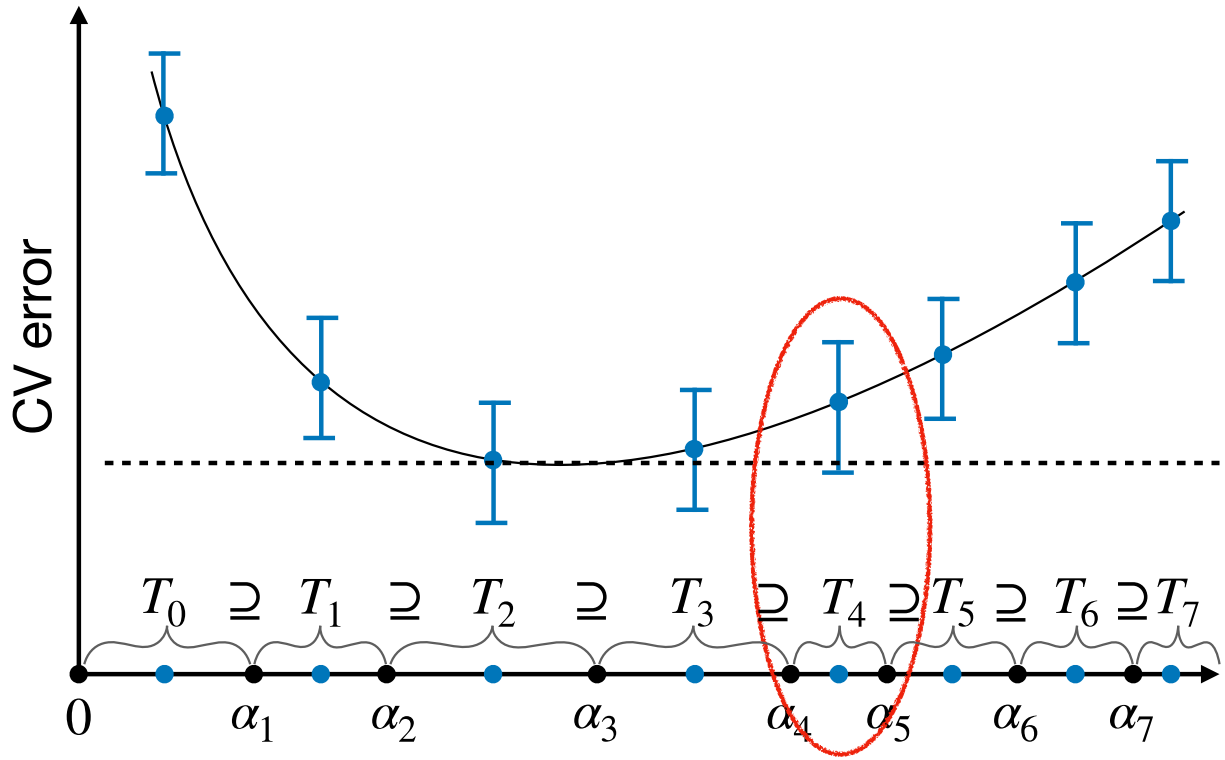
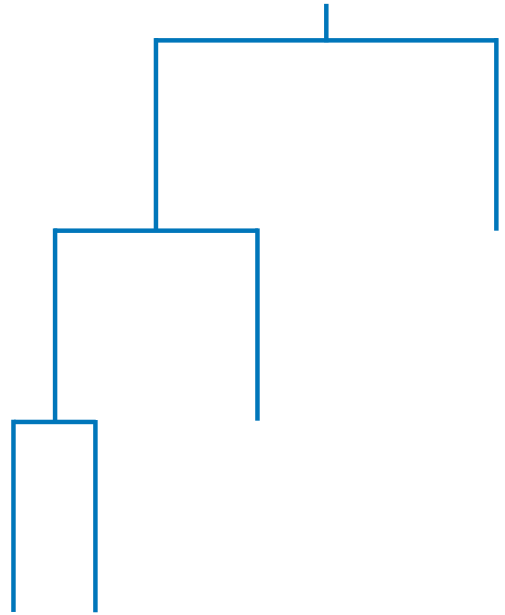
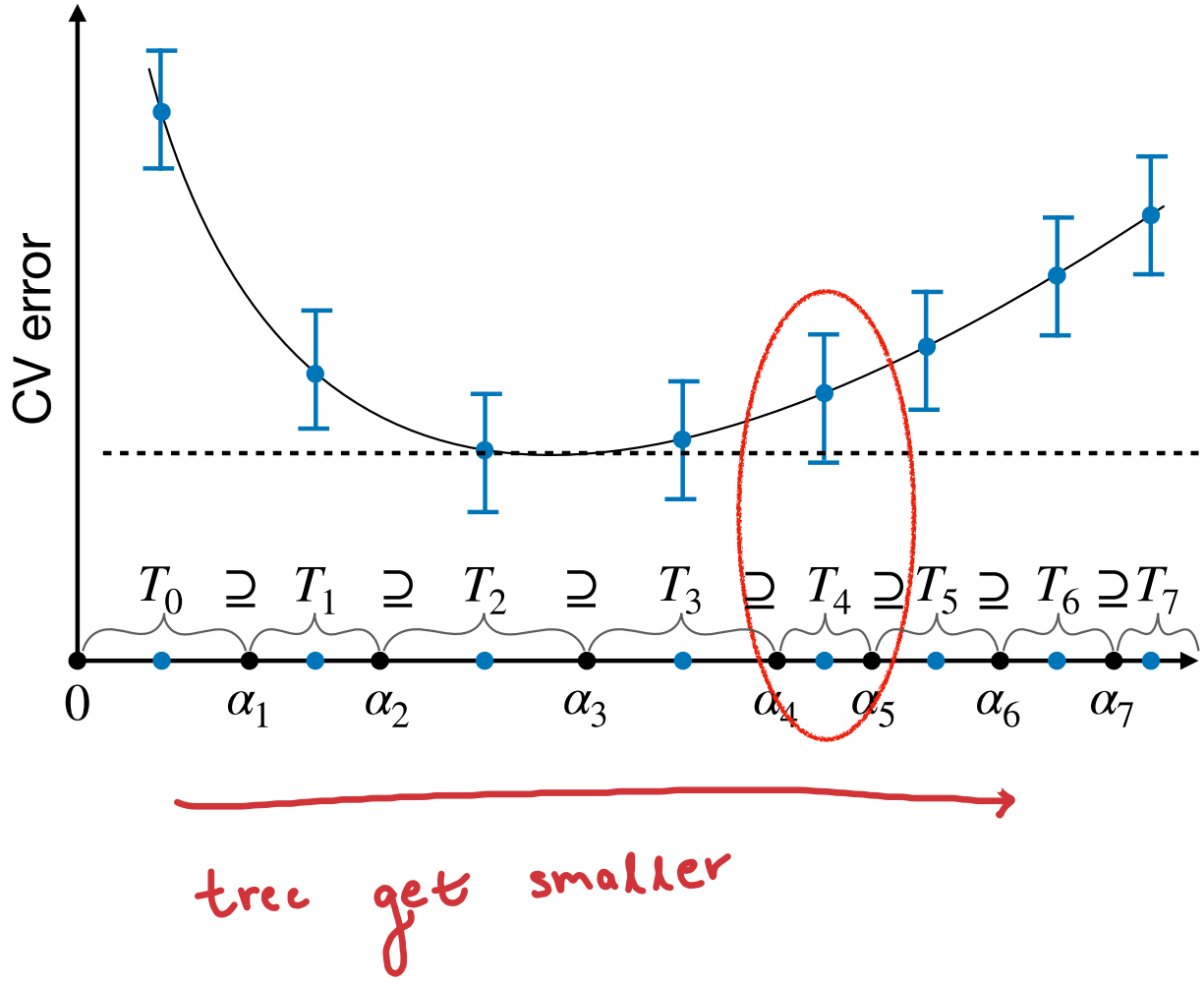


Illustration: Tree growing and pruning (new)

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



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



Relationship between α and cp (new)

- The cp parameter outputted by R is **not** the same as α due to rescaling. Its exact interpretation is beyond the scope of this class.
- We will think of cp as just another way to index the sequence of trees and we will not worry about what exactly this parameter means.
- Understanding the phenomenon that not every number of leaf nodes must be represented in the sequence of trees is also beyond the scope of this class.

CP	nsplit	rel error	xerror	xstd
0.526786	0	1.00000	1.00000	0.069256
0.053571	1	0.47321	0.47321	0.057444
0.035714	3	0.36607	0.38393	0.053093
0.010000	5	0.29464	0.41071	0.054498

This are going to be inputs to CV plot

Relationship between α and cp (new)

- The cp parameter outputted by R is **not** the same as α due to rescaling. Its exact interpretation is beyond the scope of this class.
- We will think of cp as just another way to index the sequence of trees and we will not worry about what exactly this parameter means.
- Understanding the phenomenon that not every number of leaf nodes must be represented in the sequence of trees is also beyond the scope of this class.

CP	nsplit	rel error	xerror	xstd
0.526786	0	1.00000	1.00000	0.069256
0.053571	1	0.47321	0.47321	0.057444
0.035714	3	0.36607	0.38393	0.053093
0.010000	5	0.29464	0.41071	0.054498

indexes sequence of trees

Relationship between α and cp (new)

- The cp parameter outputted by R is **not** the same as α due to rescaling. Its exact interpretation is beyond the scope of this class.
- We will think of cp as just another way to index the sequence of trees and we will not worry about what exactly this parameter means.
- Understanding the phenomenon that not every number of leaf nodes must be represented in the sequence of trees is also beyond the scope of this class.

CP	nsplit	rel error	xerror	xstd
0.526786	0	1.00000	1.00000	0.069256
0.053571	1	0.47321	0.47321	0.057444
0.035714	3	0.36607	0.38393	0.053093
0.010000	5	0.29464	0.41071	0.054498

indexes sequence of trees

number of splits

Relationship between α and cp (new)

- The cp parameter outputted by R is **not** the same as α due to rescaling. Its exact interpretation is beyond the scope of this class.
- We will think of cp as just another way to index the sequence of trees and we will not worry about what exactly this parameter means.
- Understanding the phenomenon that not every number of leaf nodes must be represented in the sequence of trees is also beyond the scope of this class.

CP	nsplit	rel error	xerror	xstd
0.526786	0	1.00000	1.00000	0.069256
0.053571	1	0.47321	0.47321	0.057444
0.035714	3	0.36607	0.38393	0.053093
0.010000	5	0.29464	0.41071	0.054498

indexes sequence of trees

number of splits

training error
(relative to zero-split tree)

Relationship between α and cp (new)

- The cp parameter outputted by R is **not** the same as α due to rescaling. Its exact interpretation is beyond the scope of this class.
- We will think of cp as just another way to index the sequence of trees and we will not worry about what exactly this parameter means.
- Understanding the phenomenon that not every number of leaf nodes must be represented in the sequence of trees is also beyond the scope of this class.

CP	nsplit	rel error	xerror	xstd
0.526786	0	1.00000	1.00000	0.069256
0.053571	1	0.47321	0.47321	0.057444
0.035714	3	0.36607	0.38393	0.053093
0.010000	5	0.29464	0.41071	0.054498

indexes sequence of trees

number of splits

training error
(relative to zero-split tree)

CV estimate

Relationship between α and cp (new)

- The cp parameter outputted by R is **not** the same as α due to rescaling. Its exact interpretation is beyond the scope of this class.
- We will think of cp as just another way to index the sequence of trees and we will not worry about what exactly this parameter means.
- Understanding the phenomenon that not every number of leaf nodes must be represented in the sequence of trees is also beyond the scope of this class.

CP	nsplit	rel error	xerror	xstd
0.526786	0	1.00000	1.00000	0.069256
0.053571	1	0.47321	0.47321	0.057444
0.035714	3	0.36607	0.38393	0.053093
0.010000	5	0.29464	0.41071	0.054498

indexes sequence of trees

number of splits

training error
(relative to zero-split tree)

CV estimate

CV standard error

Tree growing versus tree pruning

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

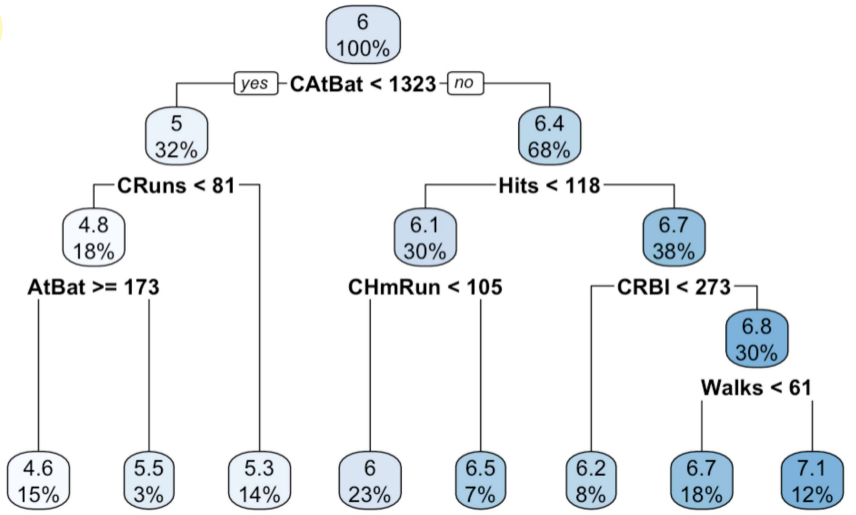
For regression trees, growing and pruning are both based on RSS.

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	*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

Observation 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*

Observation 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

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

T^{*1}

T^{*b}

T^{*B}

Regression:

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

Classification:

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