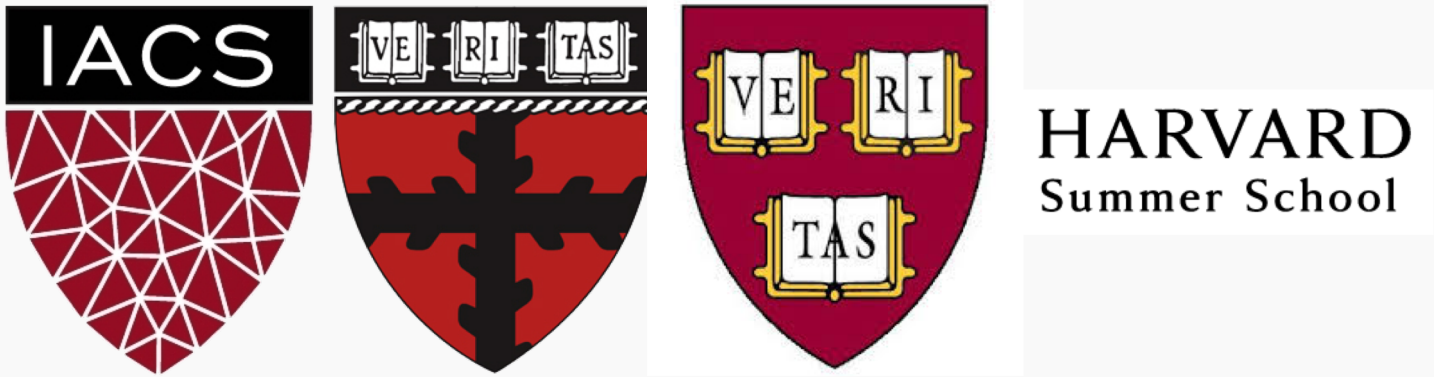
Lecture #8: Decision Trees and Random Forests

CS-S109A: Introduction to Data Science

Kevin Rader



Outline

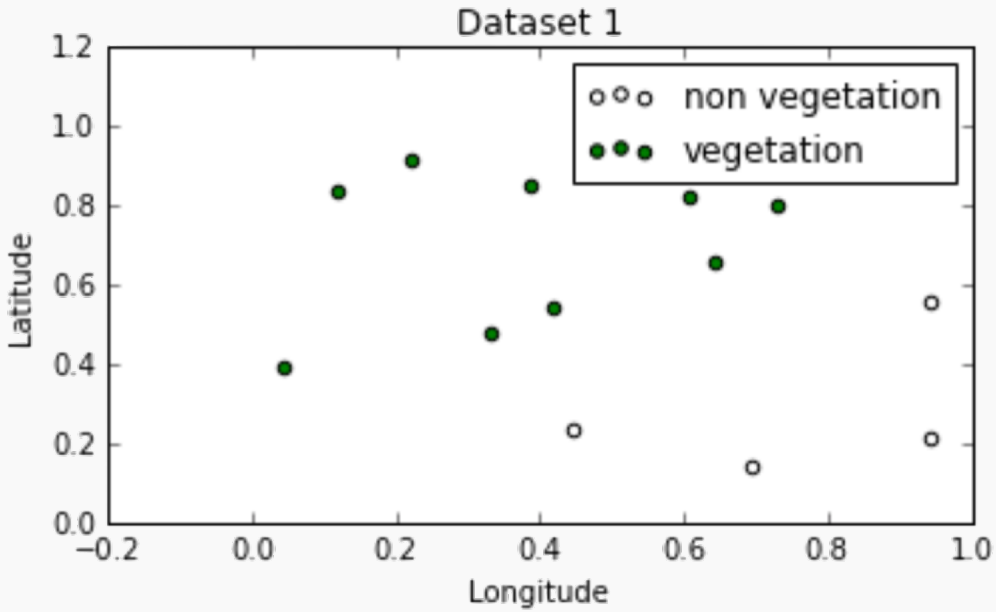
- Motivation
- Decision Trees
- Classification Trees
- Splitting Criteria
- Stopping Conditions & Pruning
- Regression Trees
- Ensemble Methods:
- Bootstrap-Aggregating (Bagging) Trees
- Random Forests

Geometry of Data

Recall:

logistic regression for building classification boundaries works best when:

- the classes are well-separated in the feature space
- have a nice geometry to the classification boundary)



Geometry of Data

Recall:

the decision boundary is defined where the probability of being in class 1 and class 0 are equal, i.e.

$$P(Y = 1) = 1 - P(Y = 1) \quad \Rightarrow. \quad P(Y = 1) = 0.5,$$

Which is equivalent to when the `log-odds=0`:

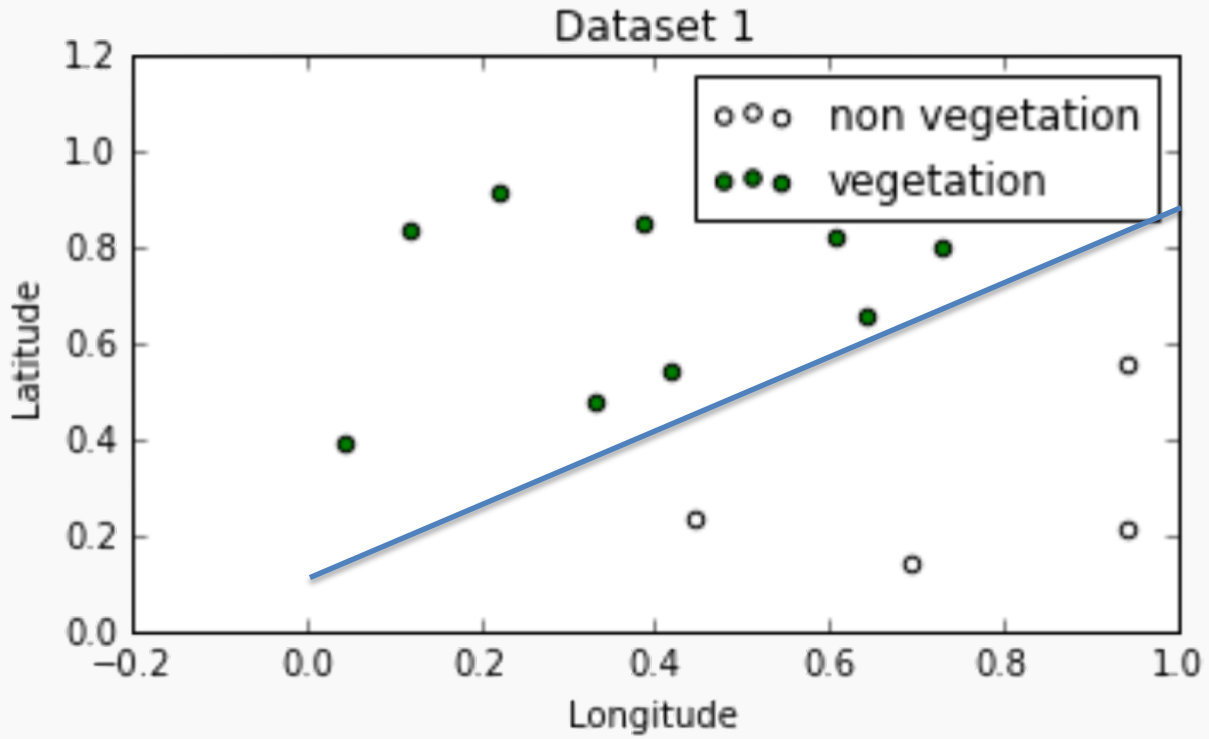
$$\boldsymbol{x}\beta = 0,$$

this equation defines a line or a hyperplane. It can be *generalized* with higher order polynomial terms.

Geometry of Data

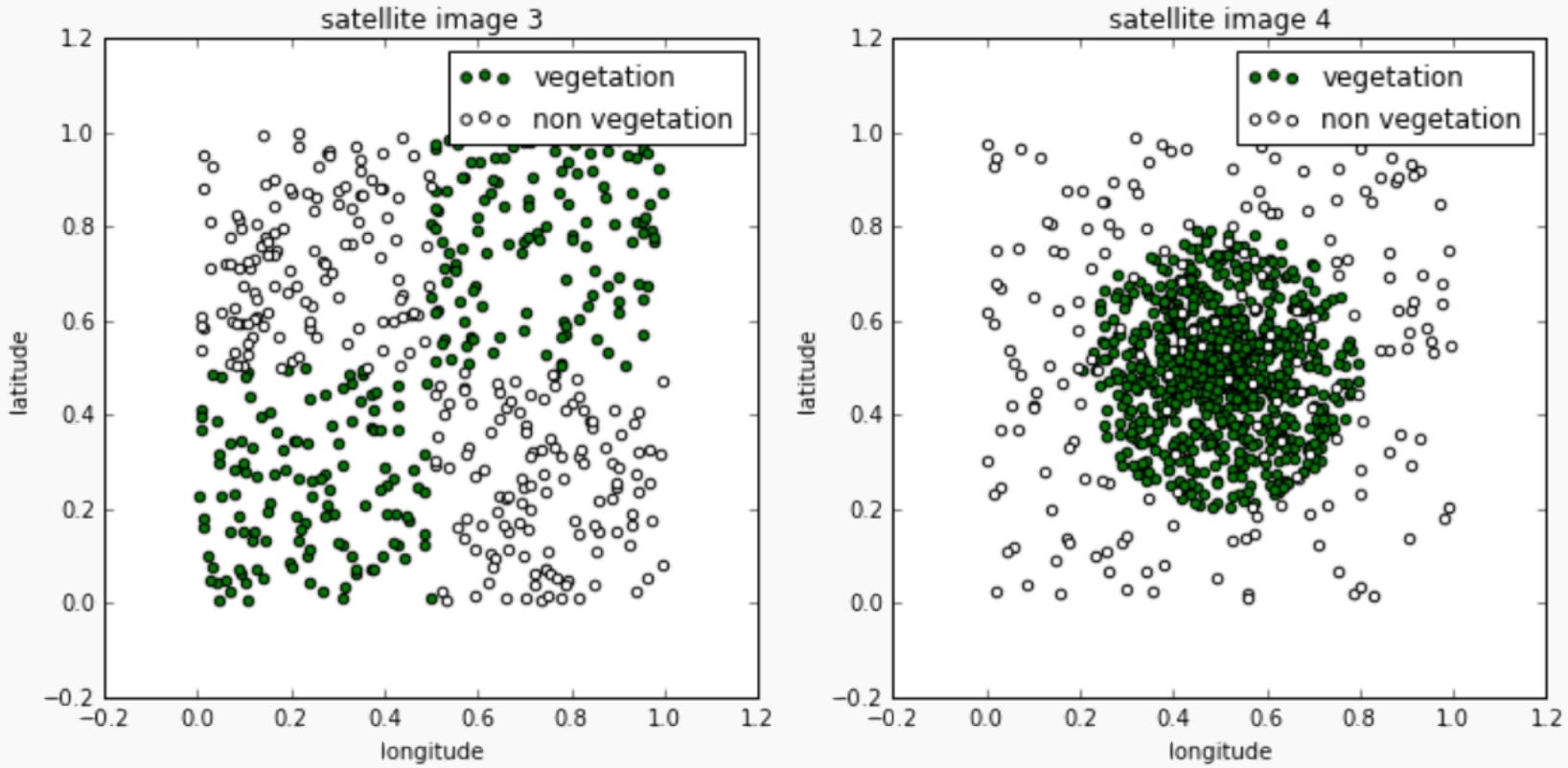
Question: Can you guess the equation that defines the decision boundary below?

$$-0.8x_1 + x_2 = 0 \implies x_2 = 0.8x_1 \Rightarrow \ Latitude = 0.8\ Lon$$



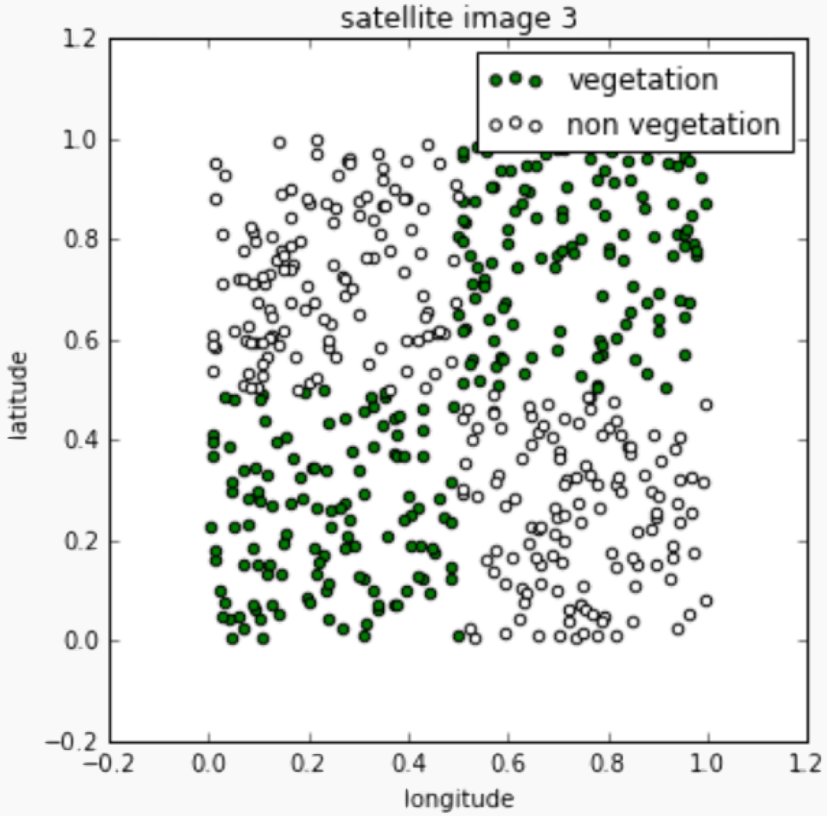
Geometry of Data

Question: How about these?



Geometry of Data

Notice that in these datasets the classes are still well-separated in the feature space, but ***the decision boundaries cannot easily be described by single equations***:



Geometry of Data

While logistic regression models with linear boundaries are intuitive to interpret by examining the impact of each predictor on the log-odds of a positive classification, it is less straightforward to interpret nonlinear decision boundaries in context:

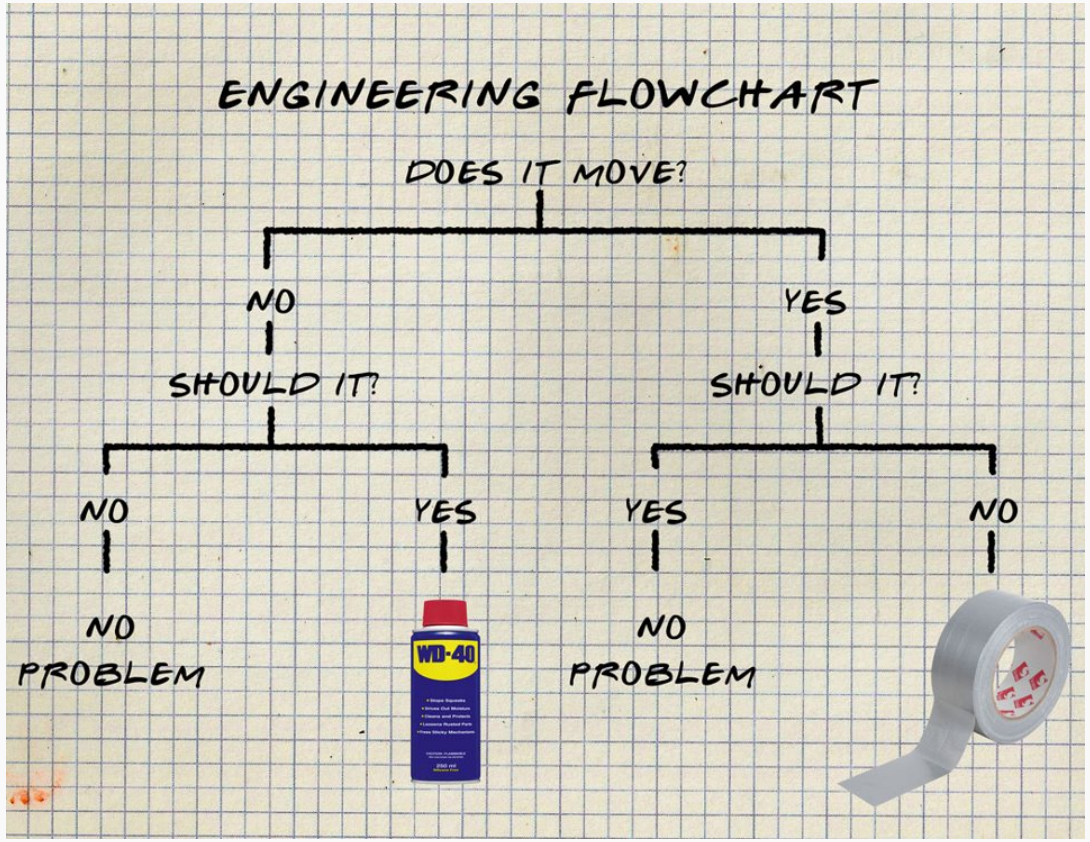
$$(x_3+2x_2) - x_1^2 + 10 = 0$$

It would be desirable to build models that:

1. allow for ***complex decision boundaries***.
2. are also ***easy to interpret***.

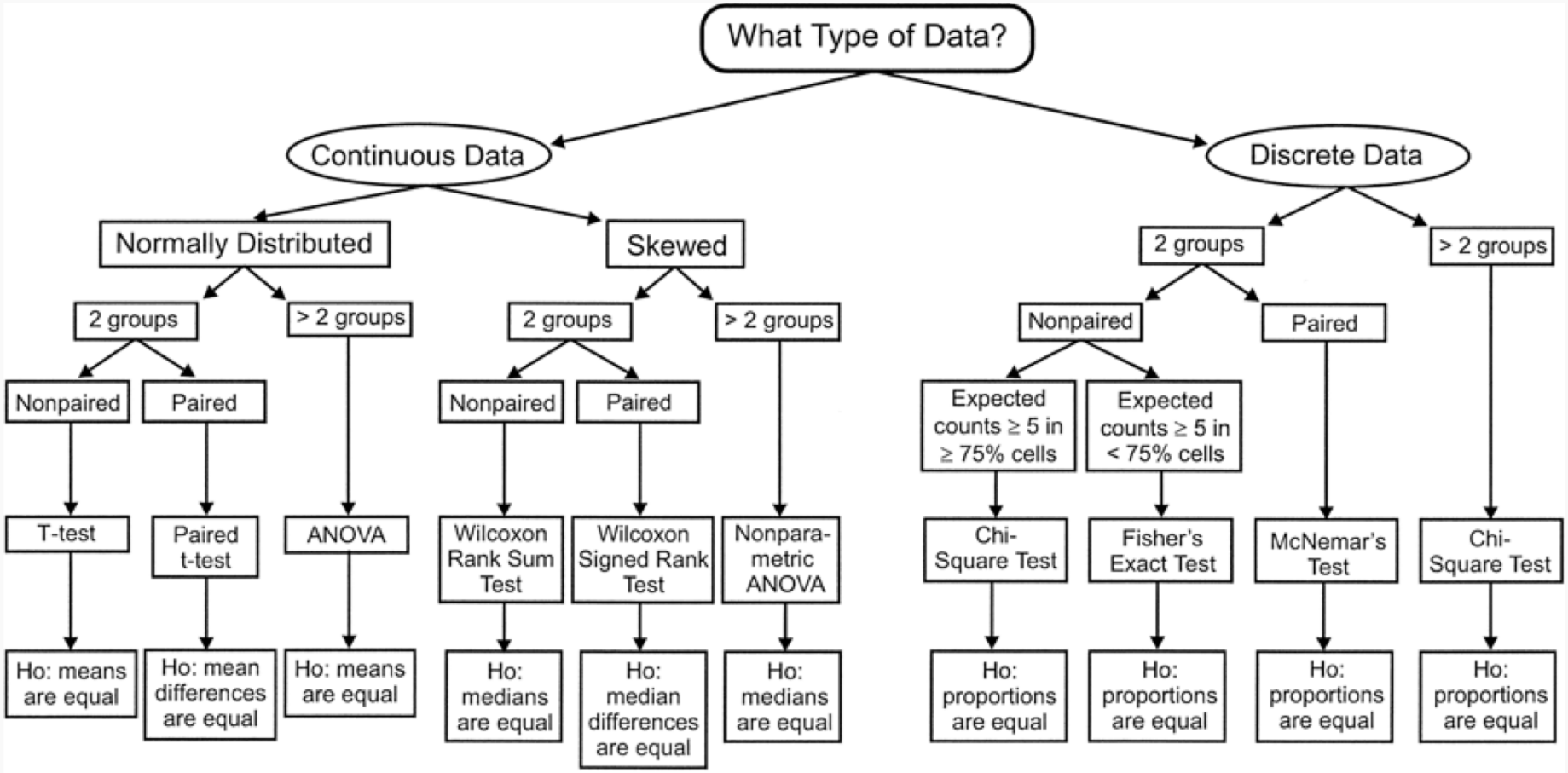
Interpretable Models

People in every walk of life have long been using interpretable models for differentiating between classes of objects and phenomena:



Interpretable Models (cont.)

Or in the [inferential] data analysis world:



Source: Waning B, Montagne M: *Pharmacoepidemiology: Principles and Practice*: http://www.accesspharmacy.com

Decision Trees

It turns out that the simple flow charts in our examples can be formulated as mathematical models for classification and these models have the properties we desire; they are:

1. interpretable by humans
2. have sufficiently complex decision boundaries
3. the decision boundaries are locally linear, each component of the decision boundary is simple to describe mathematically.

Decision Trees

The Geometry of Flow Charts

Flow charts whose graph is a tree (connected and no cycles) represents a model called a ***decision tree***.

Formally, a ***decision tree model*** is one in which the final outcome of the model is based on a series of comparisons of the values of predictors against threshold values.

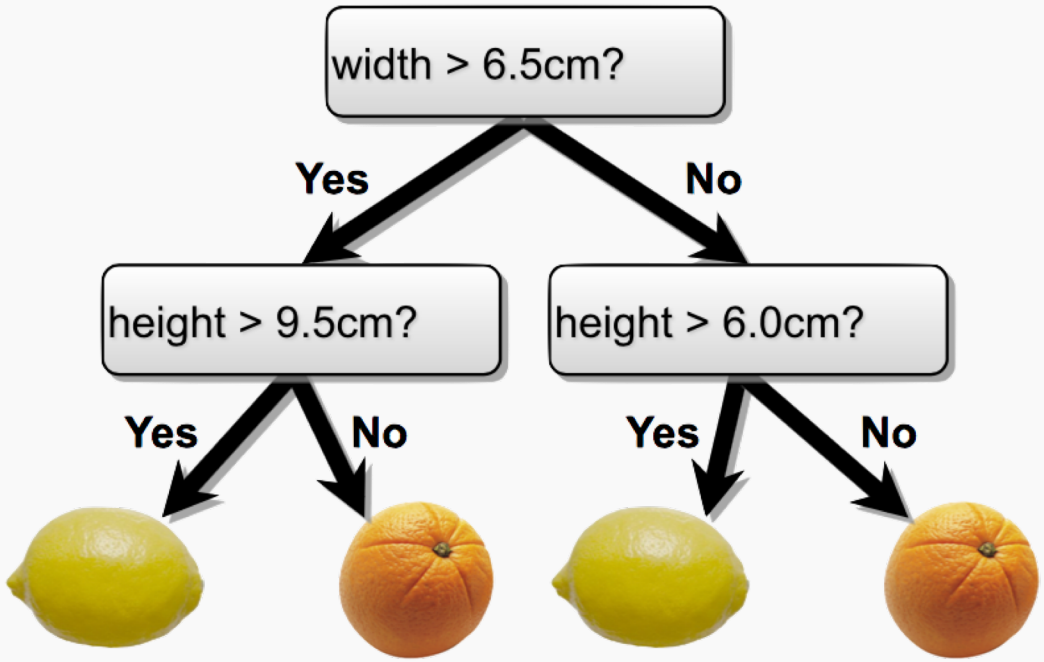
In a graphical representation (flow chart),

- the internal nodes of the tree represent attribute testing.
- branching in the next level is determined by attribute value (yes/no).
- terminal leaf nodes represent class assignments.

The Geometry of Flow Charts

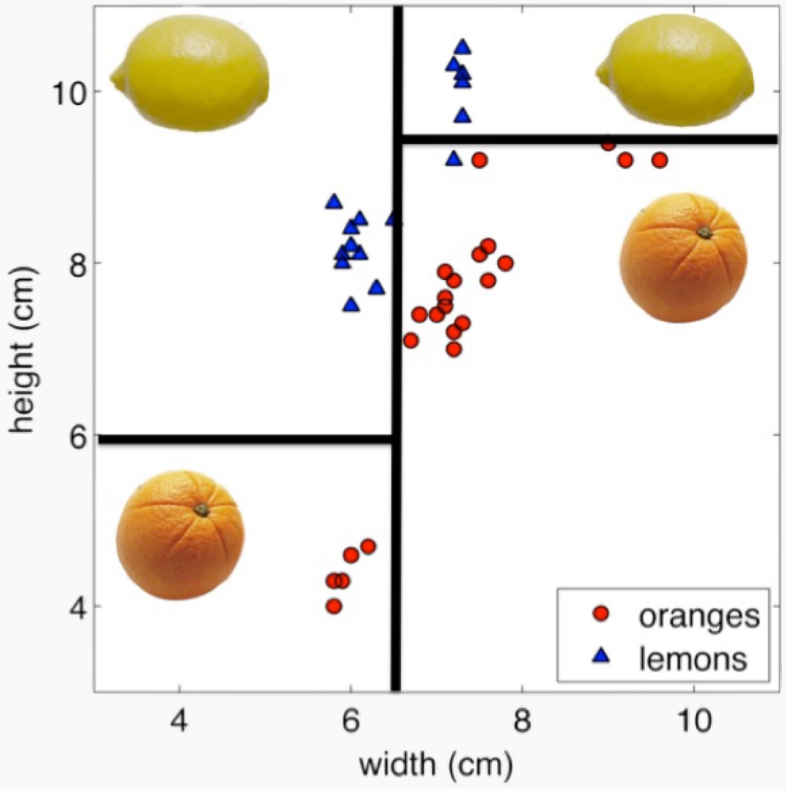
Flow charts whose graph is a tree (connected and no cycles) represents a model called a ***decision tree***.

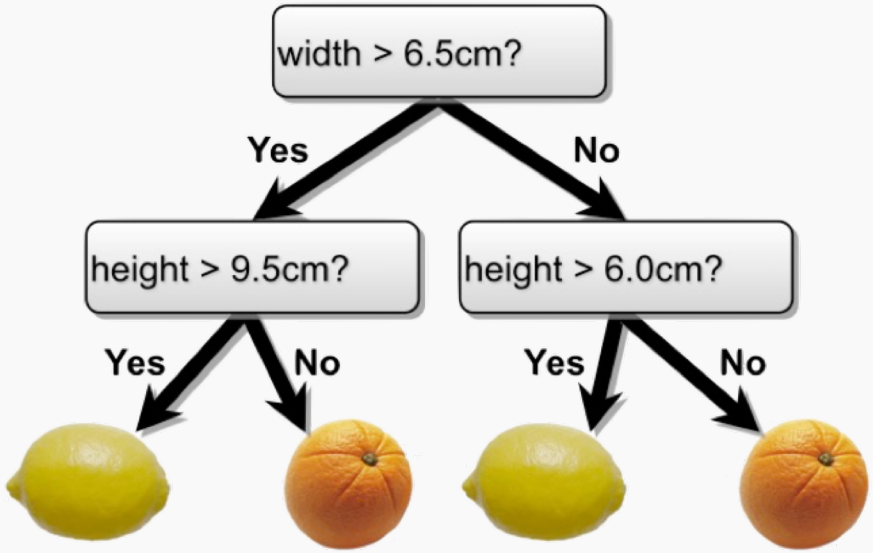
Formally, a ***decision tree model*** is one in which the final outcome of the model is based on a series of comparisons of the values of predictors against threshold values.



The Geometry of Flow Charts

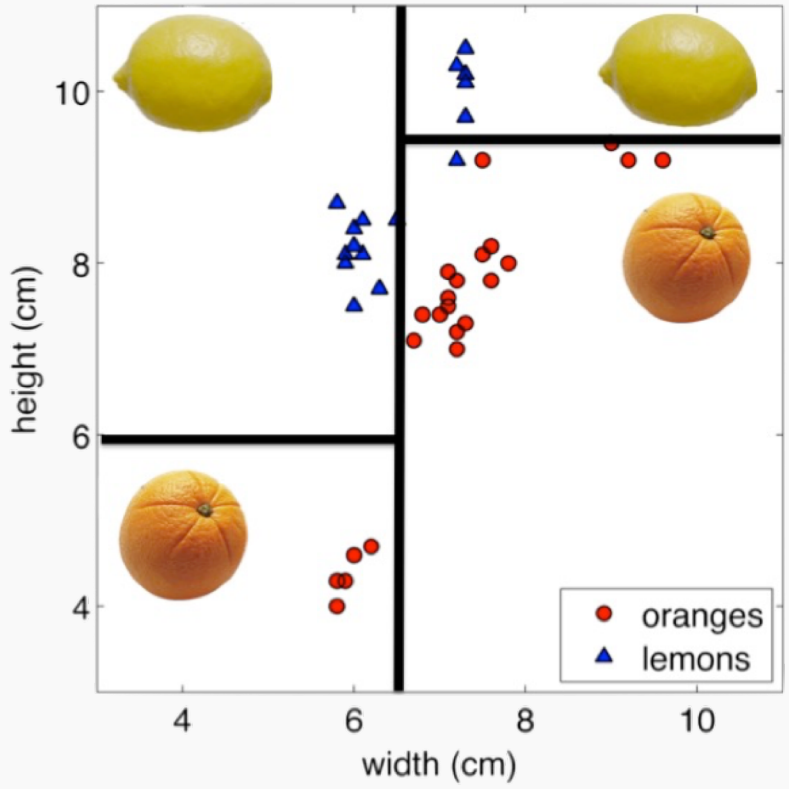
Every flow chart tree corresponds to a partition of the feature space by **axis aligned lines or (hyper) planes**. Conversely, every such partition can be written as a flow chart tree.

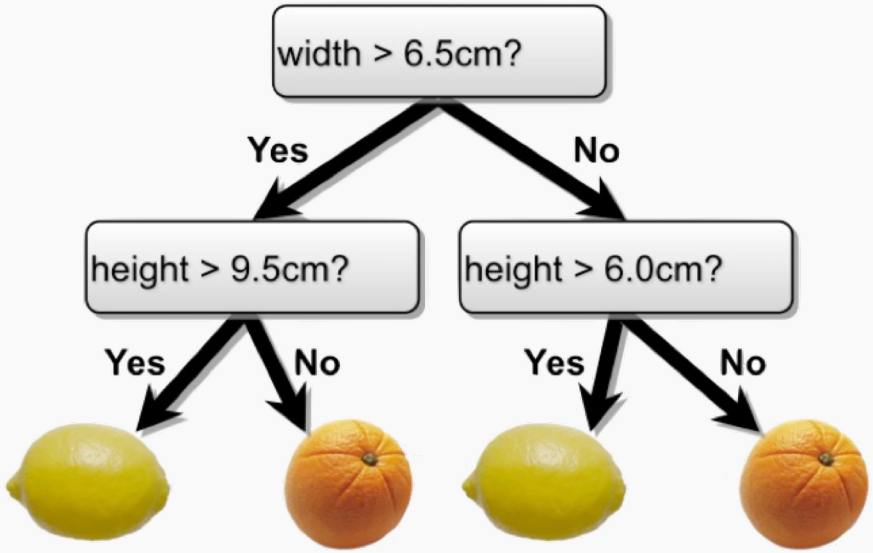




The Geometry of Flow Charts

Each comparison and branching represents splitting a region in the feature space on a single feature. Typically, at each iteration, we split once along one dimension (one predictor). Why?





Learning the Model

Given a training set, *learning* a decision tree model for binary classification means:

- producing an ***optimal*** partition of the feature space with axis-aligned linear boundaries (very interpretable!),
- each region is predicted to have a class label based on the **largest class** of the training points in that region (Bayes' classifier) when performing prediction.

Learning the Model

Learning the smallest 'optimal' decision tree for any given set of data is NP complete for numerous simple definitions of 'optimal'. Instead, we will seek a reasonably model using a greedy algorithm.

1. Start with an empty decision tree (undivided feature space)
2. Choose the 'optimal' predictor on which to split and choose the 'optimal' threshold value for splitting.
3. Recurse on each new node until ***stopping condition*** is met

Now, we need only define our splitting criterion and stopping condition.

Numerical vs Categorical Attributes

Note that the 'compare and branch' method by which we defined classification tree works well for numerical features.

However, if a feature is categorical (with more than two possible values), comparisons like *`feature < threshold`* does not make sense.

How can we handle this?

A simple solution is to encode the values of a categorical feature using numbers and treat this feature like a numerical variable. This is indeed what some computational libraries (e.g. `sklearn`) do, however, this method has drawbacks.

Numerical vs Categorical Attributes

Example

Supposed the feature we want to split on is **color**, and the values are: Red, Blue and Yellow. If we encode the categories numerically as:

Red = 0, Blue = 1, Yellow = 2

Then the possible non-trivial splits on **color** are

{{Red}, {Blue, Yellow}} {{Red, Blue},{Yellow}}

But if we encode the categories numerically as:

Red = 2, Blue = 0, Yellow = 1

The possible splits are

{{Blue}, {Yellow, Red}} {{Blue,Yellow}, {Red}}

Depending on the encoding, the splits we can optimize over can be different!

Numerical vs Categorical Attributes

In practice, the effect of our choice of naive encoding of categorical variables are often negligible - models resulting from different choices of encoding will perform comparably.

In cases where you might worry about encoding, there is a more sophisticated way to numerically encode the values of categorical variables so that one can optimize over all possible partitions of the values of the variable.

This more principled encoding scheme is computationally more expensive but is implemented in a number of computational libraries (e.g. `R's randomForest`).

Splitting Criteria (we'll skim this section)

Optimality of Splitting

While there is no 'correct' way to define an optimal split, there are some common sensical guidelines for every splitting criterion:

- the regions in the feature space should grow progressively more pure with the number of splits. That is, we should see each region 'specialize' towards a single class.
- the fitness metric of a split should take a differentiable form (making optimization possible).
- we shouldn't end up with empty regions - regions containing no training points.

Classification Error

Suppose we have J number of predictors and K classes.

Suppose we select the j^{th} predictor and split a region containing N number of training points along the threshold $t_j \in \mathbb{R}$.

We can assess the quality of this split by measuring the **classification error** made by each newly created region, R_1, R_2:

$$\text{Error}(i|j, t_j) = 1 - \max_k p(k|R_i)$$

where $p(k|R_i)$ is the proportion of training points in R_i that are labeled class k.

Classification Error

Example

	Class 1	Class 2	Error$(i\|j, t_j)$
R_1	0	6	$1 - \max\{6/6, 0/6\} = 0$
R_2	5	8	$1 - \max\{5/13, 8/13\} = 5/13$

We can now try to find the predictor j and the threshold t_j that minimizes the average classification error over the two regions, weighted by the population of the regions:

$$\min_{j,t_j} \left\{ \frac{N_1}{N} \text{Error}(1|j, t_j) + \frac{N_2}{N} \text{Error}(2|j, t_j) \right\}$$

where N_i is the number of training points inside region R_i.

Gini Index

Suppose we have J number of predictors, N number of training points and K classes.

Suppose we select the j^{th} predictor and split a region containing N number of training points along the threshold $t_j \in \mathbb{R}$.

We can assess the quality of this split by measuring the purity of each newly created region, R_1, R_2. This metric is called the **Gini Index**:

$$\text{Gini}(i|j, t_j) = 1 - \sum_k p(k|R_i)^2$$

Question: What is the effect of squaring the proportions of each class? What is the effect of summing the squared proportions of classes within each region?

Gini Index

Example

	Class 1	Class 2	$\text{Gini}(i\|j, t_j)$
R_1	0	6	$1 - (6/6^2 + 0/6^2) = 0$
R_2	5	8	$1 - [(5/13)^2 + (8/13)^2] = 80/169$

We can now try to find the predictor j and the threshold t_j that minimizes the average Gini Index over the two regions, weighted by the population of the regions:

$$\min_{j,t_j} \left\{ \frac{N_1}{N} \text{Gini}(1|j, t_j) + \frac{N_2}{N} \text{Gini}(2|j, t_j) \right\}$$

where N_i is the number of training points inside region R_i.

Information Theory

The last metric for evaluating the quality of a split is motivated by metrics of uncertainty in information theory.

Ideally, our decision tree should split the feature space into regions such that each region represents a single class. In practice, the training points in each region is distributed over multiple classes, e.g.:

	Class 1	Class 2
R_1	1	6
R_2	5	6

However, though both imperfect, R_1 is clearly sending a stronger 'signal' for a single class (Class 2) than R_2.

Information Theory

One way to quantify the strength of a signal in a particular region is to analyze the distribution of classes within the region. We compute the **entropy** of this distribution.

For a random variable with a discrete distribution, the entropy is computed by:

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Higher entropy means the distribution is uniform-like (flat histogram) and thus values sampled from it are 'less predictable' (all possible values are equally probable).

Lower entropy means the distribution has more defined peaks and valleys and thus values sampled from it are 'more predictable' (values around the peaks are more probable).

Entropy

Suppose we have J number of predictors, N number of training points and K classes.

Suppose we select the j^{th} predictor and split a region containing N number of training points along the threshold $t_j \in \mathbb{R}$.

We can assess the quality of this split by measuring the entropy of the class distribution in each newly created region, R_1, R_2:

$$\min_{j,t_j} \left\{ \frac{N_1}{N} \text{Entropy}(1|j,t_j) + \frac{N_2}{N} \text{Entropy}(2|j,t_j) \right\}$$

Note: we are actually computing the conditional entropy of the distribution of training points amongst the K classes given that the point is in region i.

Entropy

Example

	Class 1	Class 2	$\text{Entropy}(i\|j,t_j)$
R_1	0	6	$-(\frac{6}{6}\log_2\frac{6}{6}+\frac{0}{6}\log_2\frac{0}{6})=0$
R_2	5	8	$-(\frac{5}{13}\log_2\frac{5}{13}+\frac{8}{13}\log_2\frac{8}{13})\approx 1.38$

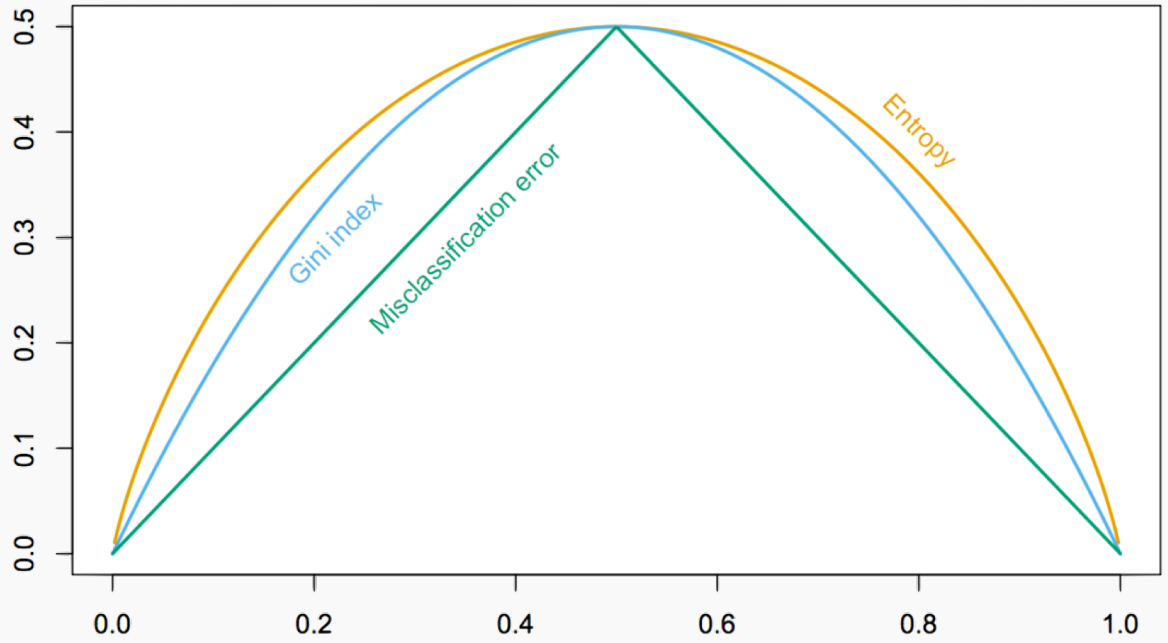
We can now try to find the predictor j and the threshold t_j that minimizes the average entropy over the two regions, weighted by the population of the regions:

$$\min_{j,t_j}\left\{\frac{N_1}{N}\text{Entropy}(1|j,t_j)+\frac{N_2}{N}\text{Entropy}(2|j,t_j)\right\}$$

Comparison of Criteria

Recall our intuitive guidelines for splitting criteria, which of the three criteria fits our guideline the best?

We have the following comparison of the value of the three criteria at different levels of purity (from 0 to 1) in a single region (for binary outcomes).



Comparison of Criteria

Recall our intuitive guidelines for splitting criteria, which of the three criteria fits our guideline the best?

To note that entropy penalizes impurity the most is not to say that it is the best splitting criteria. For one, a model with purer leaf nodes on a training set may not perform better on the testing test.

Another factor to consider is the size of the tree (i.e. model complexity) each criteria tends to promote.

To compare different decision tree models, we need to first discuss ***stopping conditions***.

Stopping Conditions & Pruning

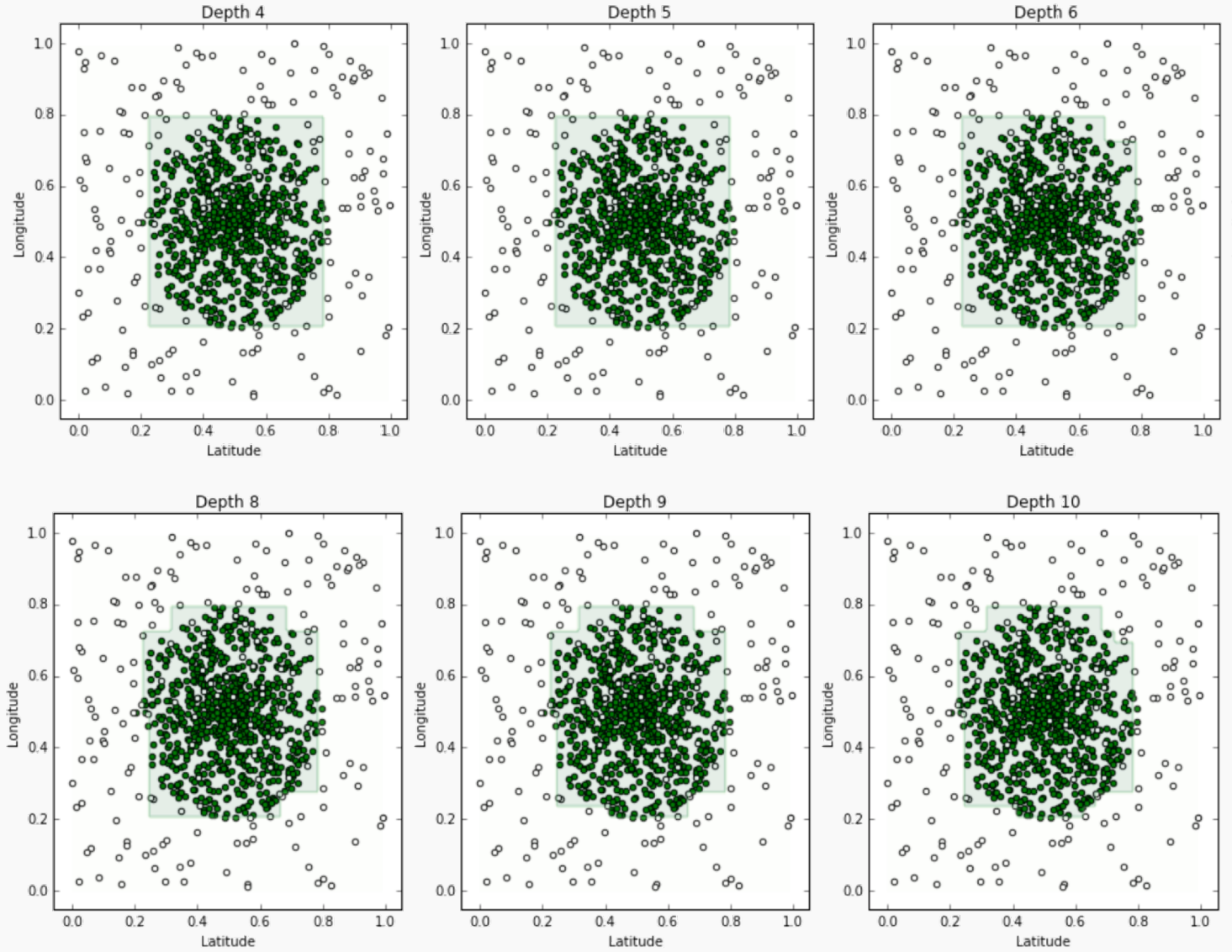
Variance vs Bias

If we don't terminate the decision tree learning algorithm manually, the tree will continue to grow until each region defined by the model possibly contains exactly one training point (and the model attains 100% training accuracy).

To prevent this from happening, we can simply stop the algorithm at a particular depth.

But how do we determine the appropriate depth?

Variance vs Bias



Variance vs Bias

We make some observations about our models:

- **(High Bias)** A tree of depth 4 is not a good fit for the training data - it's unable to capture the nonlinear boundary separating the two classes.
- **(Low Bias)** With an extremely high depth, we can obtain a model that correctly classifies all points on the boundary (by zig-zagging around each point).
- **(Low Variance)** The tree of depth 4 is robust to slight perturbations in the training data - the square carved out by the model is stable if you move the boundary points a bit.
- **(High Variance)** Trees of high depth are sensitive to perturbations in the training data, especially to changes in the boundary points.

Not surprisingly, complex trees have low bias (able to capture more complex geometry in the data) but high variance (can overfit). Complex trees are also harder to interpret and more computationally expensive to train.

Stopping Conditions

Common simple stopping conditions:

- Don’t split a region if all instances in the region belong to the same class.
- Don’t split a region if the number of instances in the sub-region will fall below pre-defined threshold (**min_samples_leaf**).
- Don’t split a region if the total number of leaves in the tree will exceed pre-defined threshold.

The appropriate thresholds can be determined by evaluating the model on a held-out data set or, better yet, via cross-validation.

Stopping Conditions

More restrictive stopping conditions:

- Compute the gain in purity, gain in information, or reduction in entropy of splitting a region R into R_1 and R_2:

$$Gain(R) = \Delta(R) = m(R) - \frac{N_1}{N} m(R_1) - \frac{N_2}{N} m(R_2)$$

where m is a metric like the Gini Index or entropy. Don't split if the gain is less than some pre-defined threshold (**min_impurity_decrease**).

Alternative to Using Stopping Conditions

What is the major issue with pre-specifying a stopping condition?

- you may stop too early or stop too late.

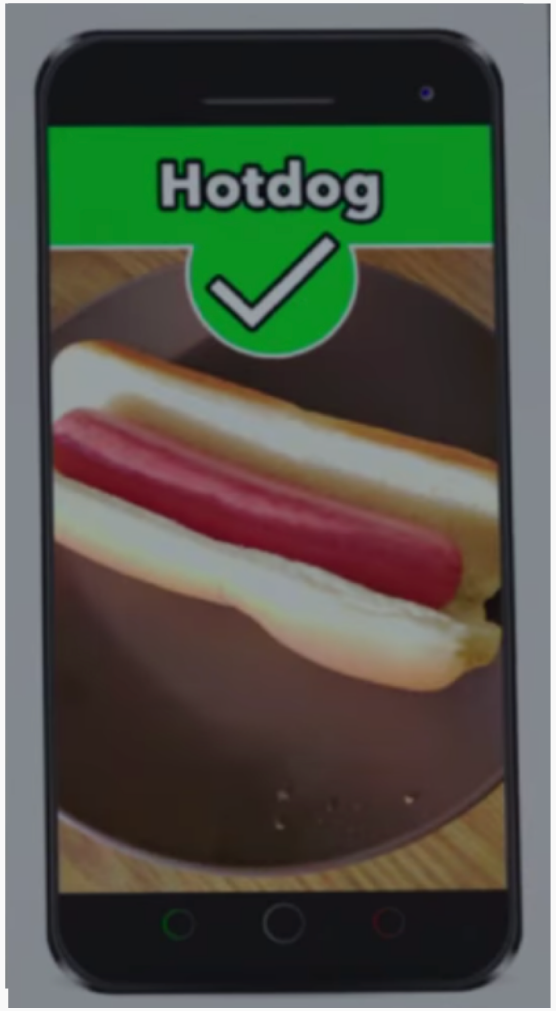
How can we fix this issue?

- choose several stopping criterion (set minimal Gain(R) at various levels) and cross-validate which is the best.

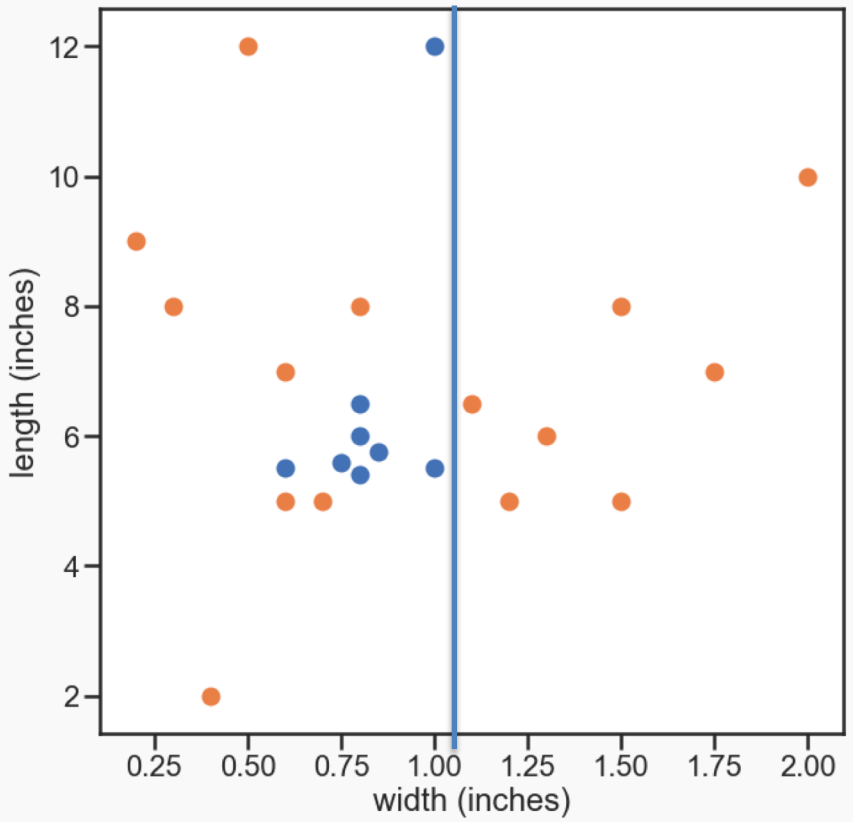
What is an alternative approach to this issue?

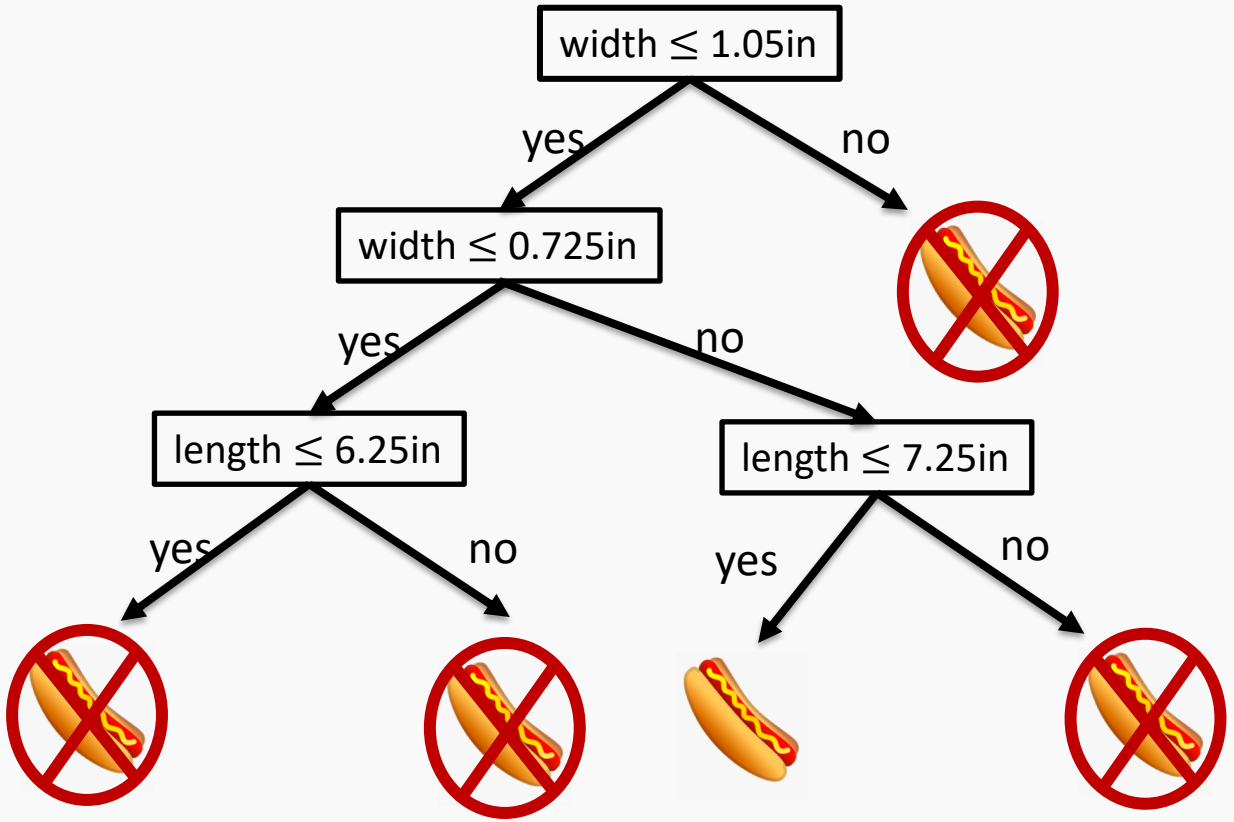
- Don't stop. Instead prune back!

To Hot Dog or Not Hot Dog…

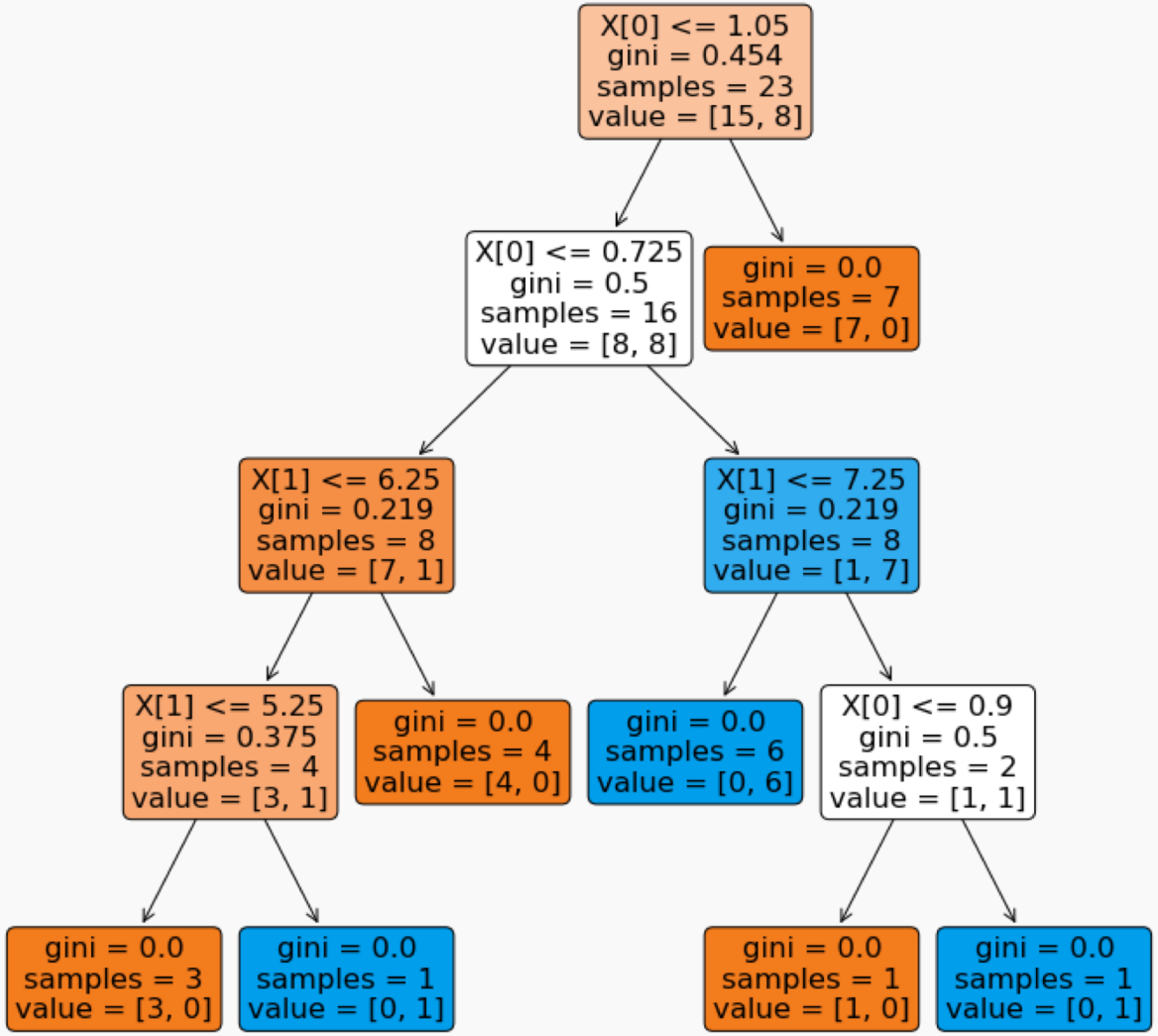


Hot Dog or Not

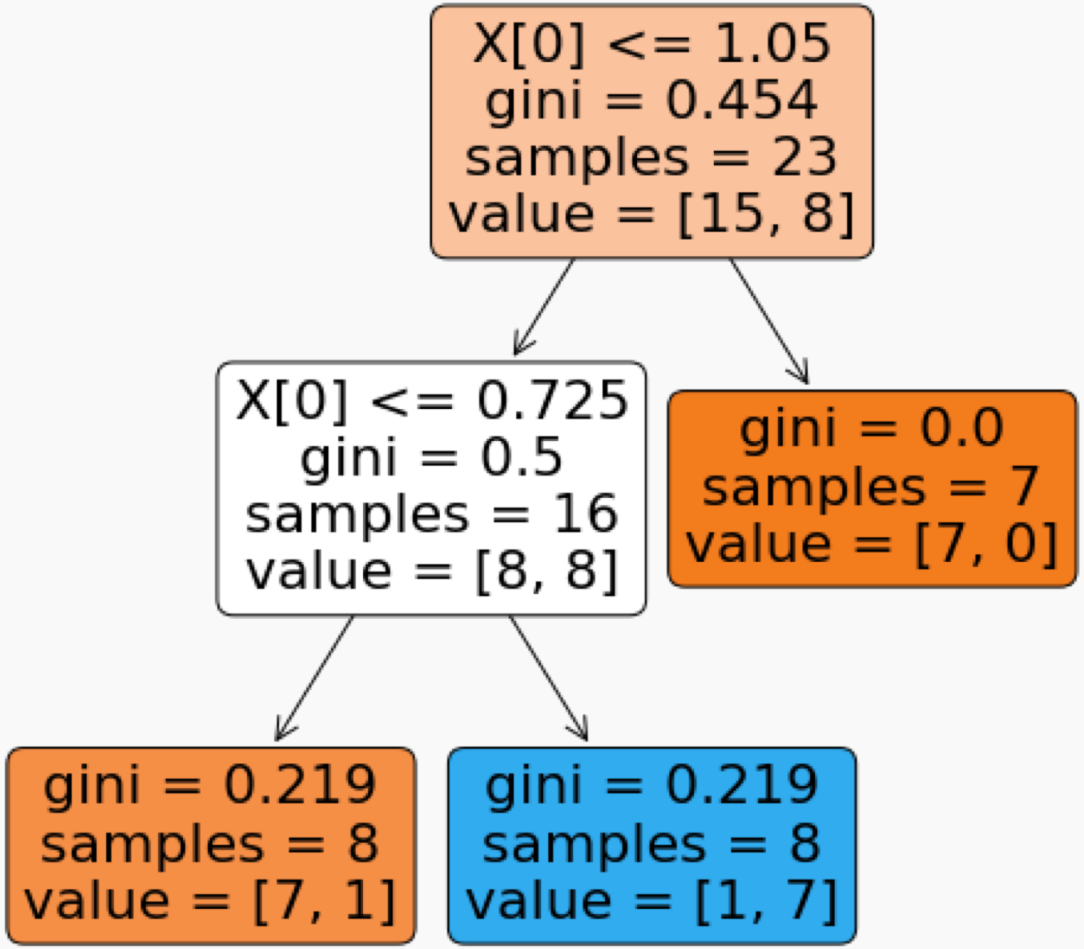




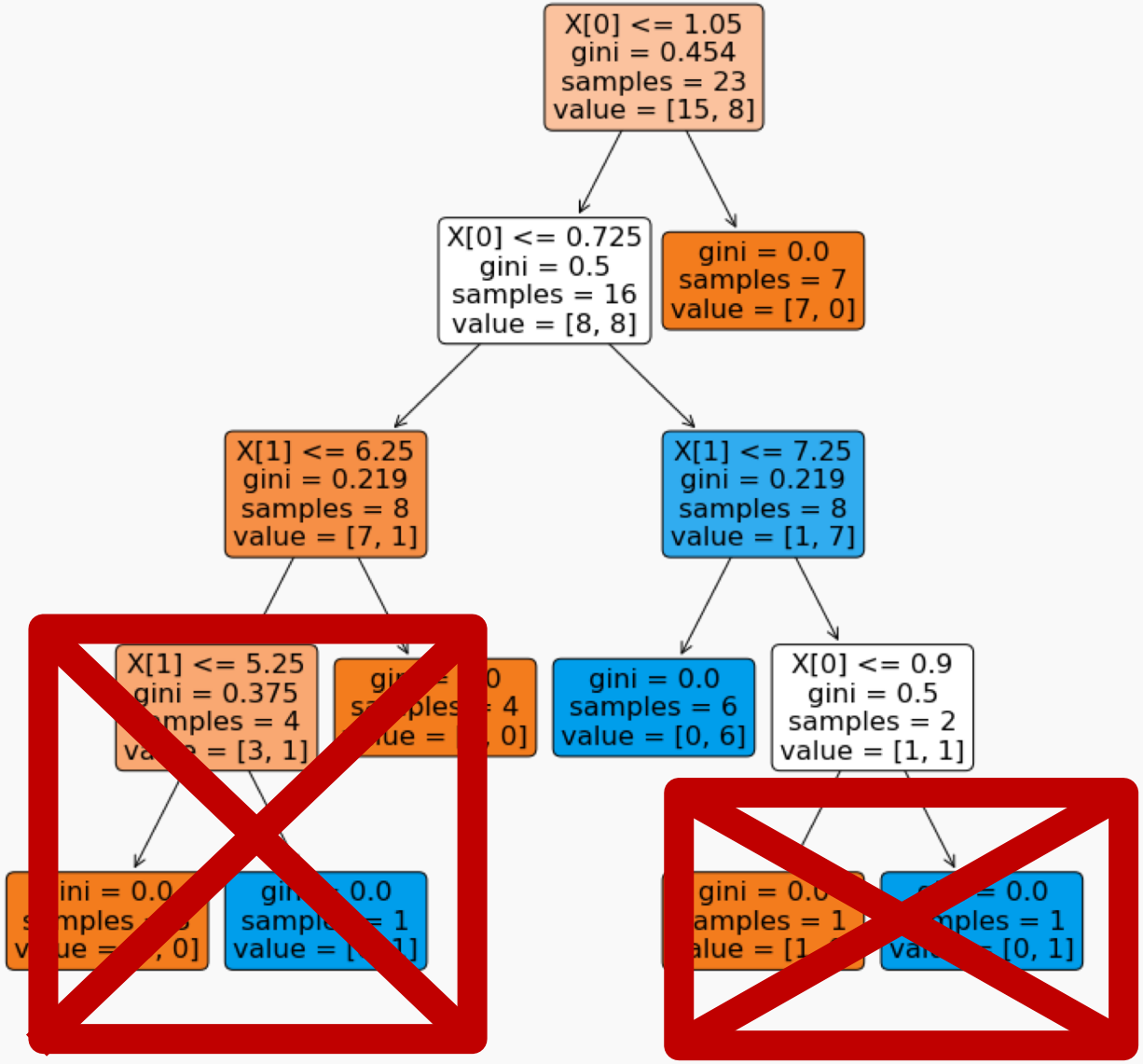
Motivation for Pruning



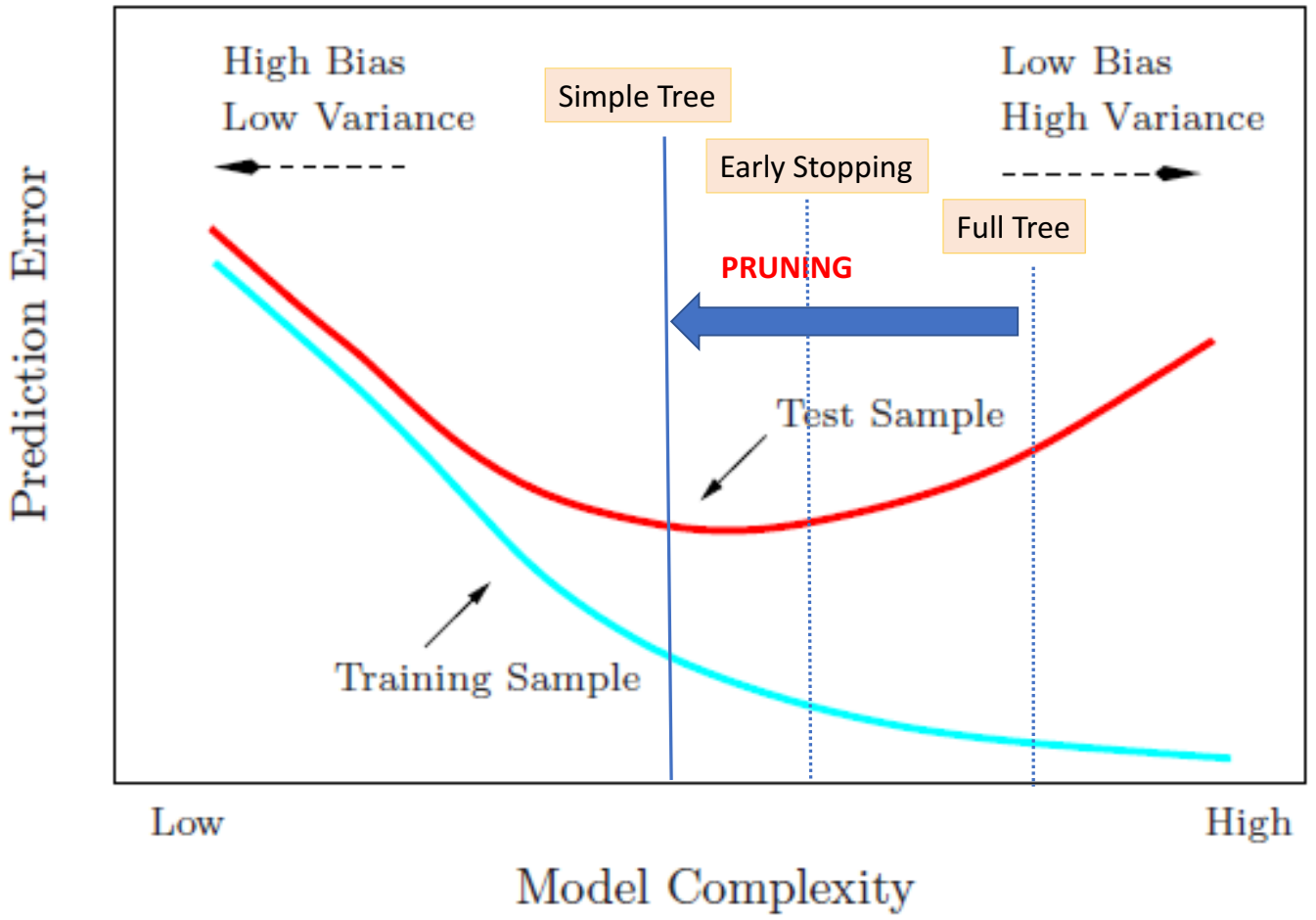
Motivation for Pruning: if we were to stop early



Motivation for Pruning



Motivation for Pruning



Pruning

Rather than preventing a complex tree from growing, we can obtain a simpler tree by 'pruning' a complex one.

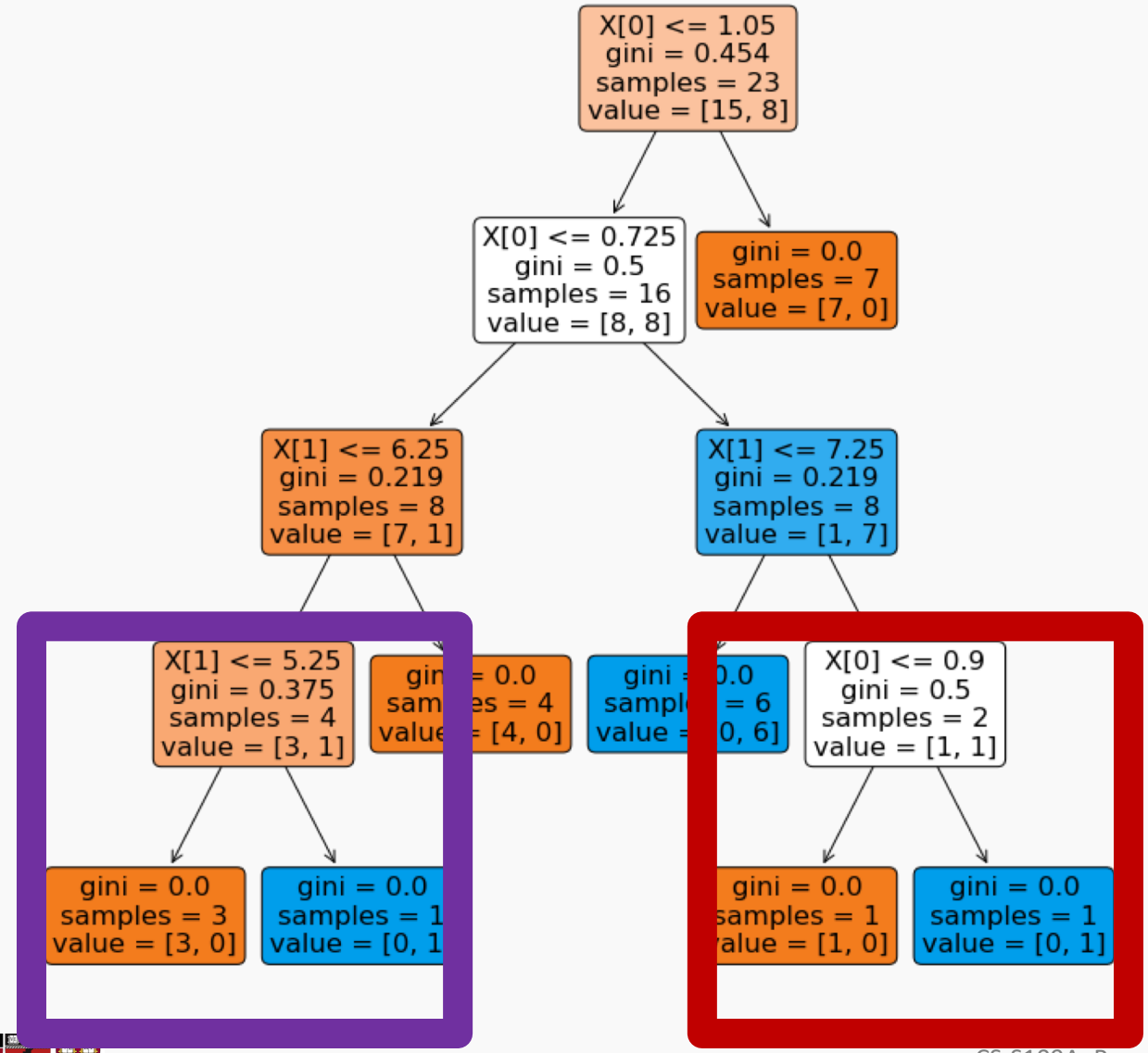
There are many method of pruning, a common one is ***cost complexity pruning***, where by we select from a array of smaller subtrees of the full model that optimizes a balance of performance and efficiency.

That is, we measure

$$C(T) = Error(T) + \alpha|T|$$

where T is a decision (sub) tree, $|T|$ is the number of leaves in the tree and α is the parameter for penalizing model complexity.

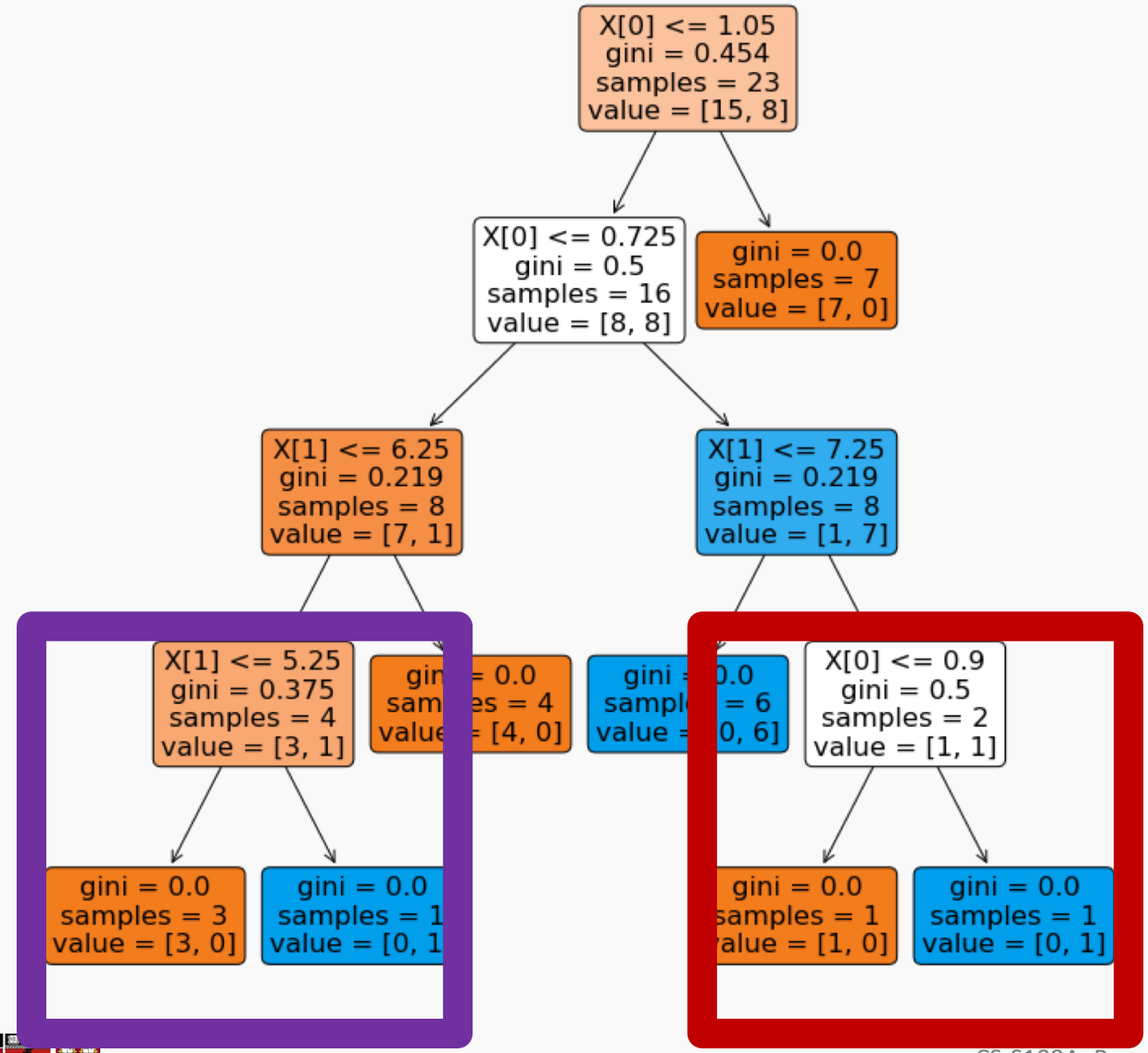
Pruning our Tree of Hot Dogs



Let $\alpha = 0.01$:

Tree	Error	# Leaves	Total Cost
Full (T)	0	6	0.06
T_1	0.0434	5	0.0934
T_2	0.0434	5	0.0934

Pruning our Tree of Hot Dogs



Let $\alpha = 0.2$:

Tree	Error	# Leaves	Total Cost
Full (T)	0	6	1.2
T_1	0.0434	5	1.0434
T_2	0.0434	5	1.0434

Pruning

$$C(T) = Error(T) + \alpha|T|$$

1. Fix α.

2. Find best tree for a given α and based on cost complexity *C*.

3. Tune for the best α using CV (what should be the error measure?)

Note: in `sklearn`, use the **`cost_complexity_pruning_path`** method in the `DecisionTree` classes

Pruning

The pruning algorithm:

1. Start with a full tree T_0 (each leaf node is pure)
2. Replace a subtree in T_0 with a leaf node to obtain a pruned tree T_1. This subtree should be selected to minimize

$$\frac{Error(T_0) - Error(T_1)}{|T_0| - |T_1|}$$

3. Iterate this pruning process to obtain $T_0, T_1, \ldots, T_L$ where T_L is the tree containing just the root of T_0
4. Select the optimal tree T_i by cross validation.

Note: you might wonder where we are computing the cost-complexity $C(T_l)$. One can prove that this process is equivalent to explicitly optimizing *C* at each step.

Next

How can this decision tree approach apply to a ***regression problem*** (quantitative outcome)?

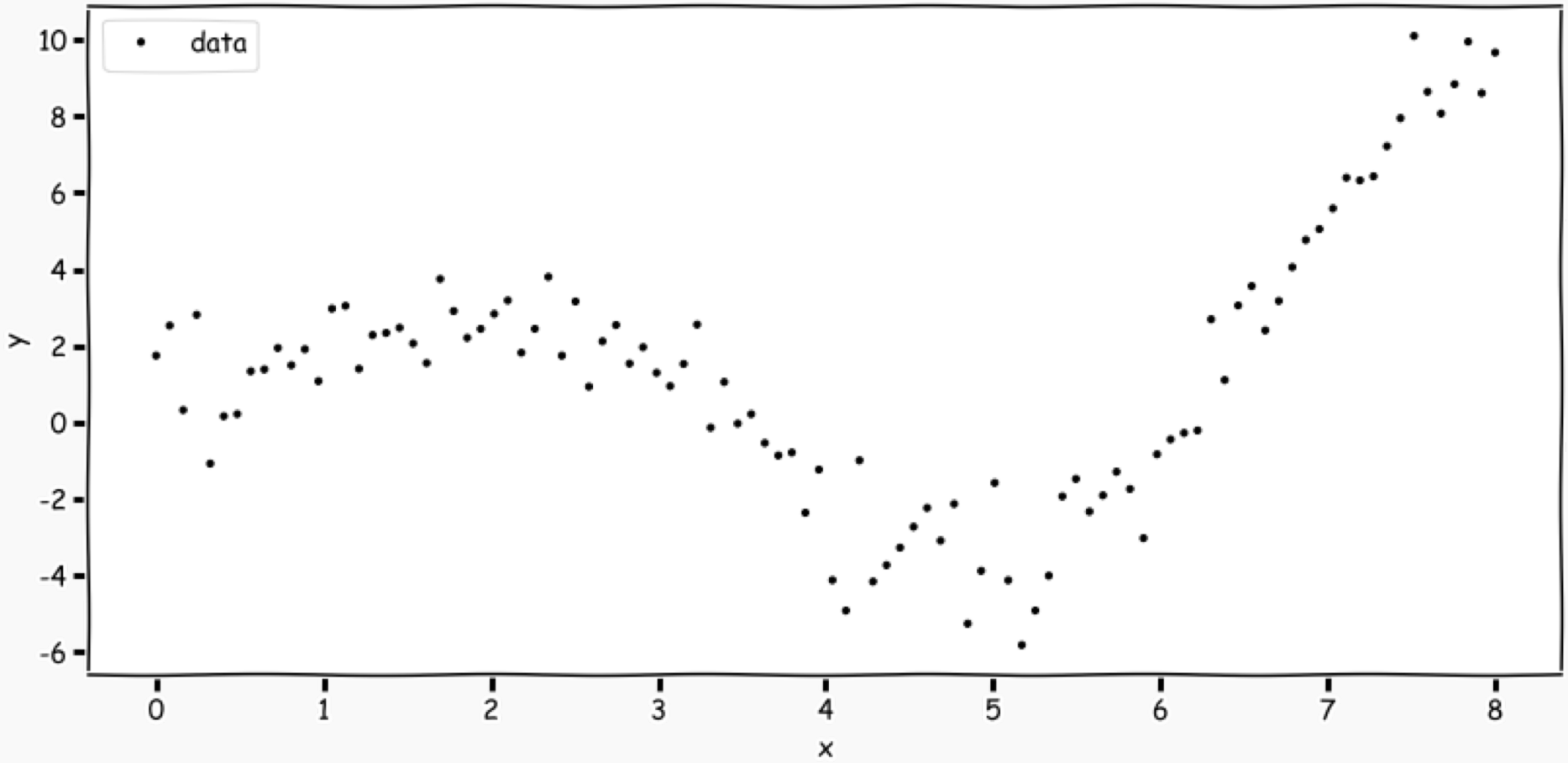
Questions to consider:

- What would be a reasonable loss function?
- How would you determine any splitting criteria?
- How would you perform prediction in each leaf?

A picture is worth a thousand words…

Regression Tree Example

How do we decide a split here?



Decision Trees for Regression

Adaptations for Regression

With just two modifications, we can use a decision tree model for regression:

1. The three splitting criteria we've examined each promoted splits that were pure - new regions increasingly specialized in a single class.

    A. **For classification**, purity of the regions is a good indicator the performance of the model.

    B. **For regression**, we want to select a splitting criterion that promotes splits that improves the predictive accuracy of the model as measured by, say, overall MSE.

2. For regression with output in $\mathbb{R}$, we want to label each region in the model with a real number - typically the average of the output values of the training points contained in the region.

Learning Regression Trees

The learning algorithms for decision trees in regression tasks is:

1. Start with an empty decision tree (undivided features pace)
2. Choose a predictor j on which to split and choose a threshold value t_j for splitting such that the weighted average MSE of the new regions as smallest possible:

$$\underset{j,t_j}{\operatorname{argmin}} \left\{ \frac{N_1}{N} \operatorname{MSE}(R_1) + \frac{N_2}{N} \operatorname{MSE}(R_2) \right\}$$

or equivalently,

$$\underset{j,t_j}{\operatorname{argmin}} \left\{ \frac{N_1}{N} \operatorname{Var}(y|x \in R_1) + \frac{N_2}{N} \operatorname{Var}(y|x \in R_2) \right\}$$

where N_i is the number of training points in R_i and N is the number of points in R.

3. Recurse on each new node until ***stopping condition*** is met.

Regression Trees Prediction

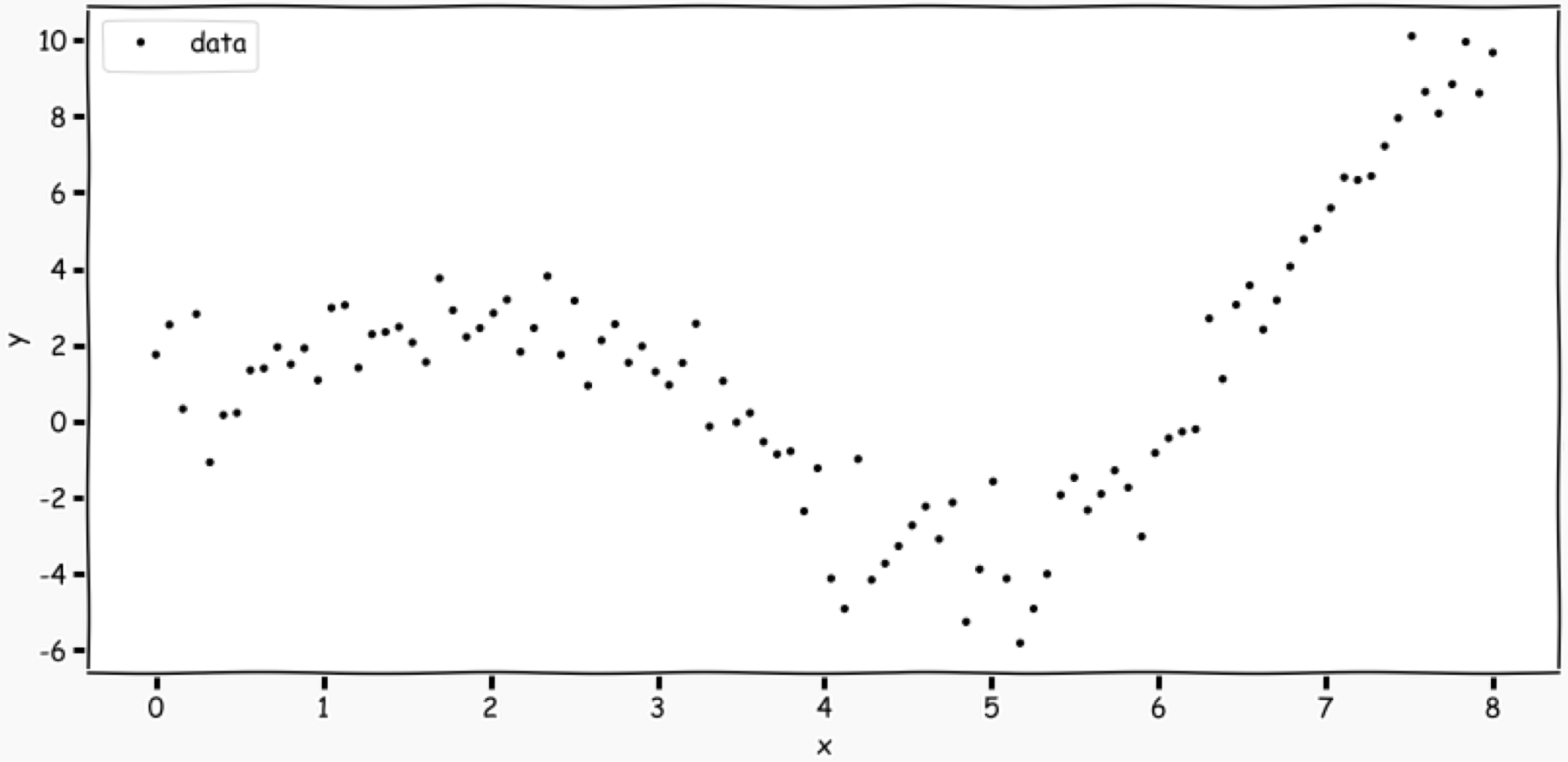
For any data point x_i

1. Traverse the tree until we reach a leaf node.

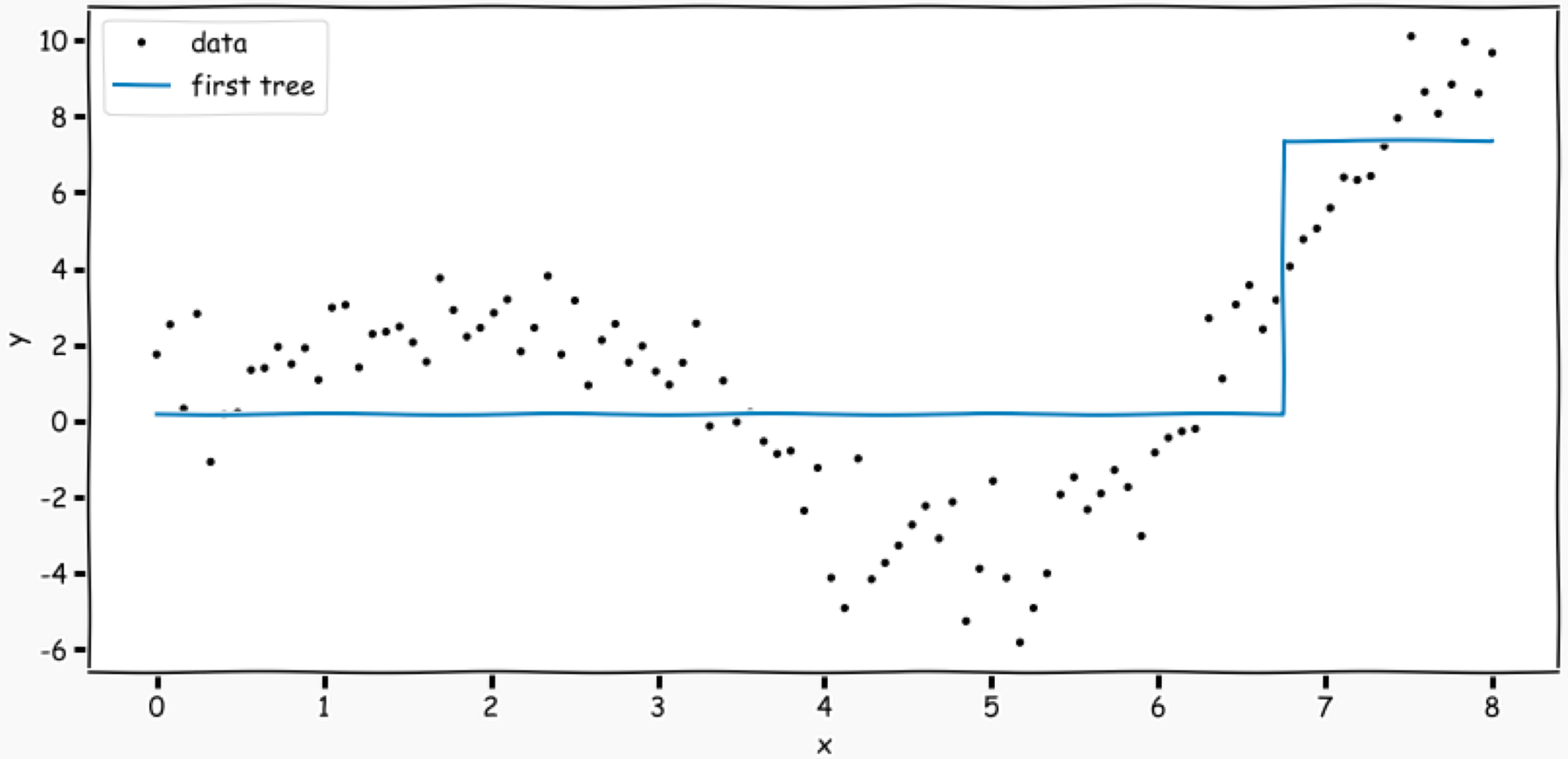
2. Averaged value of the response variable y's in the leaf (this is from the training set) is the $\widehat{y_i}$.

Regression Tree Example

How do we decide a split here? Where should we split?

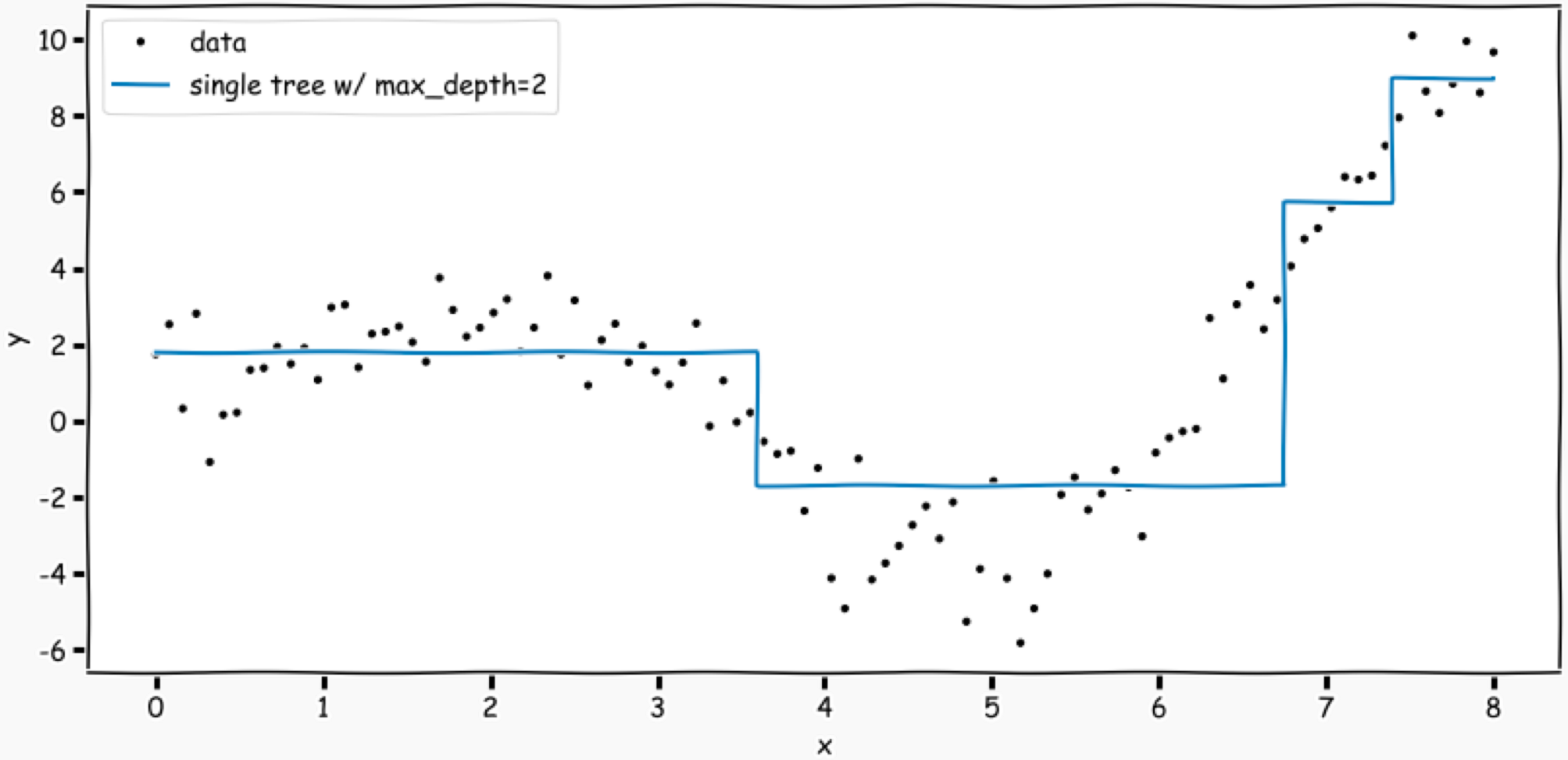


Regression Tree (max_depth = 1)

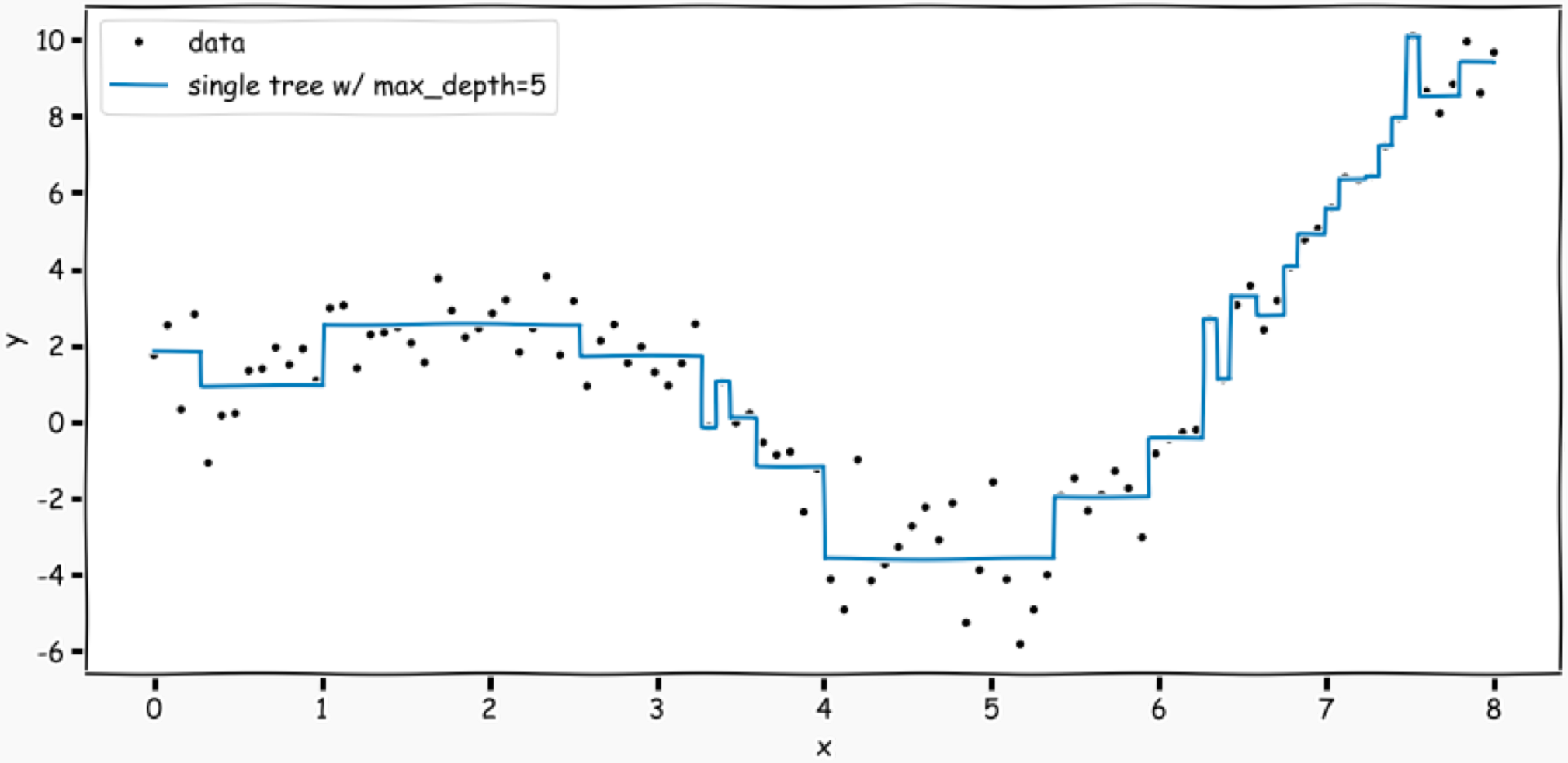


If we go to a depth of 2, where should we split next?

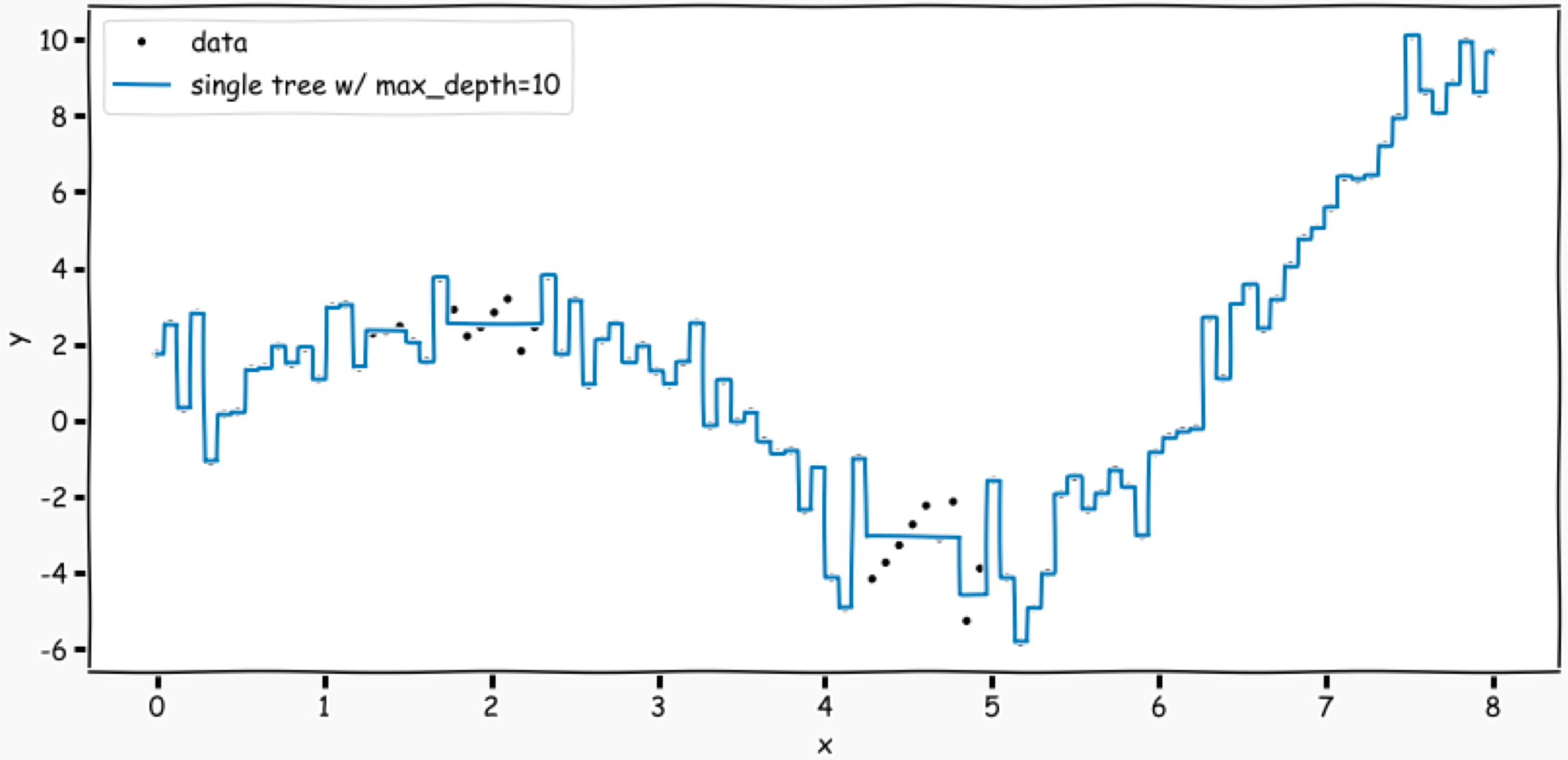
Regression Tree (max_depth = 2)



Regression Tree (max_depth = 5)



Regression Tree (max_depth = 10)



Stopping Conditions

Most of the stopping conditions, like maximum depth or minimum number of points in region, we saw for classification trees can still be applied.

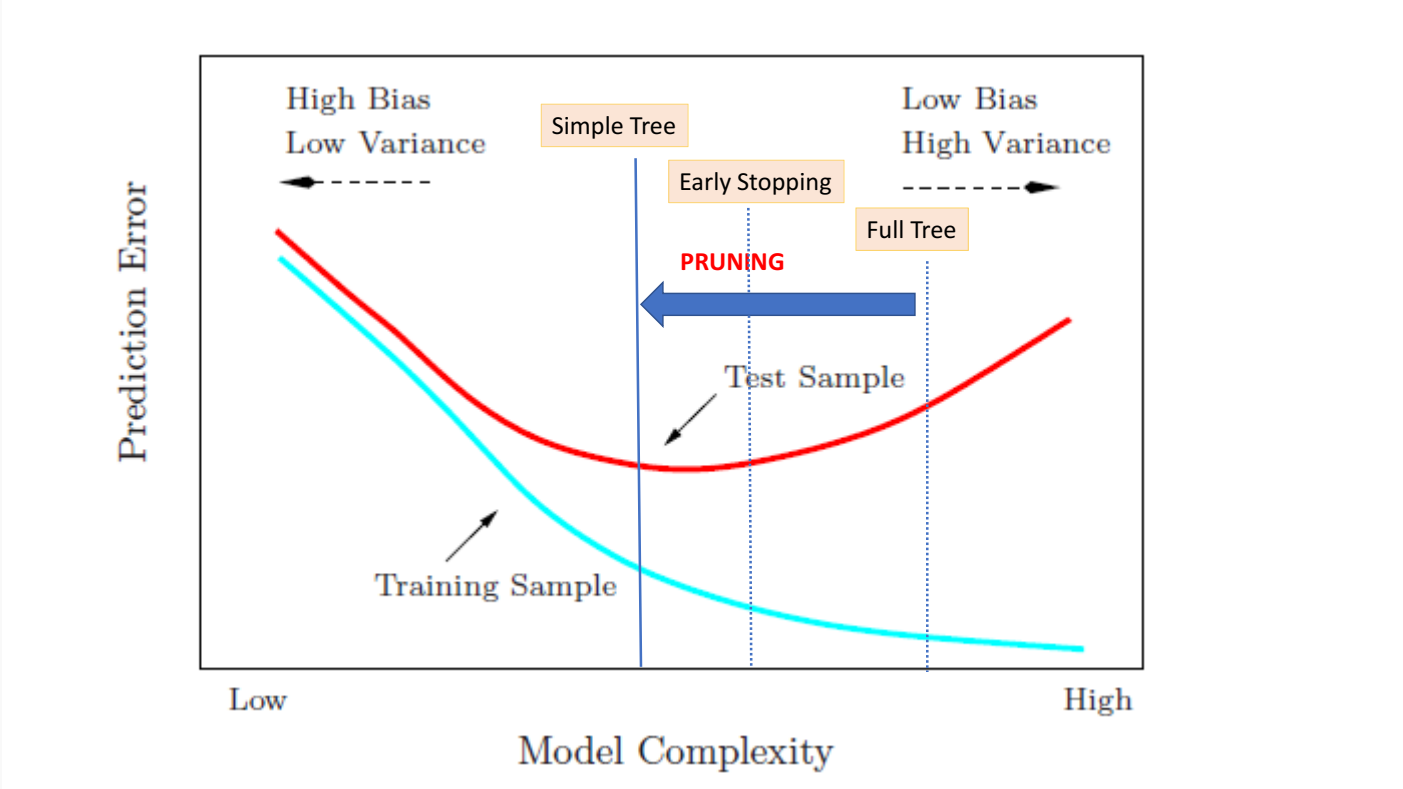
In the place of purity gain, we can instead compute accuracy gain for splitting a region R

$$\text{Gain}(R) = \Delta(R) = MSE(R) - \frac{N_1}{N} MSE(R_1) - \frac{N_2}{N} MSE(R_2)$$

and stop the tree when the gain is less than some pre-defined threshold.

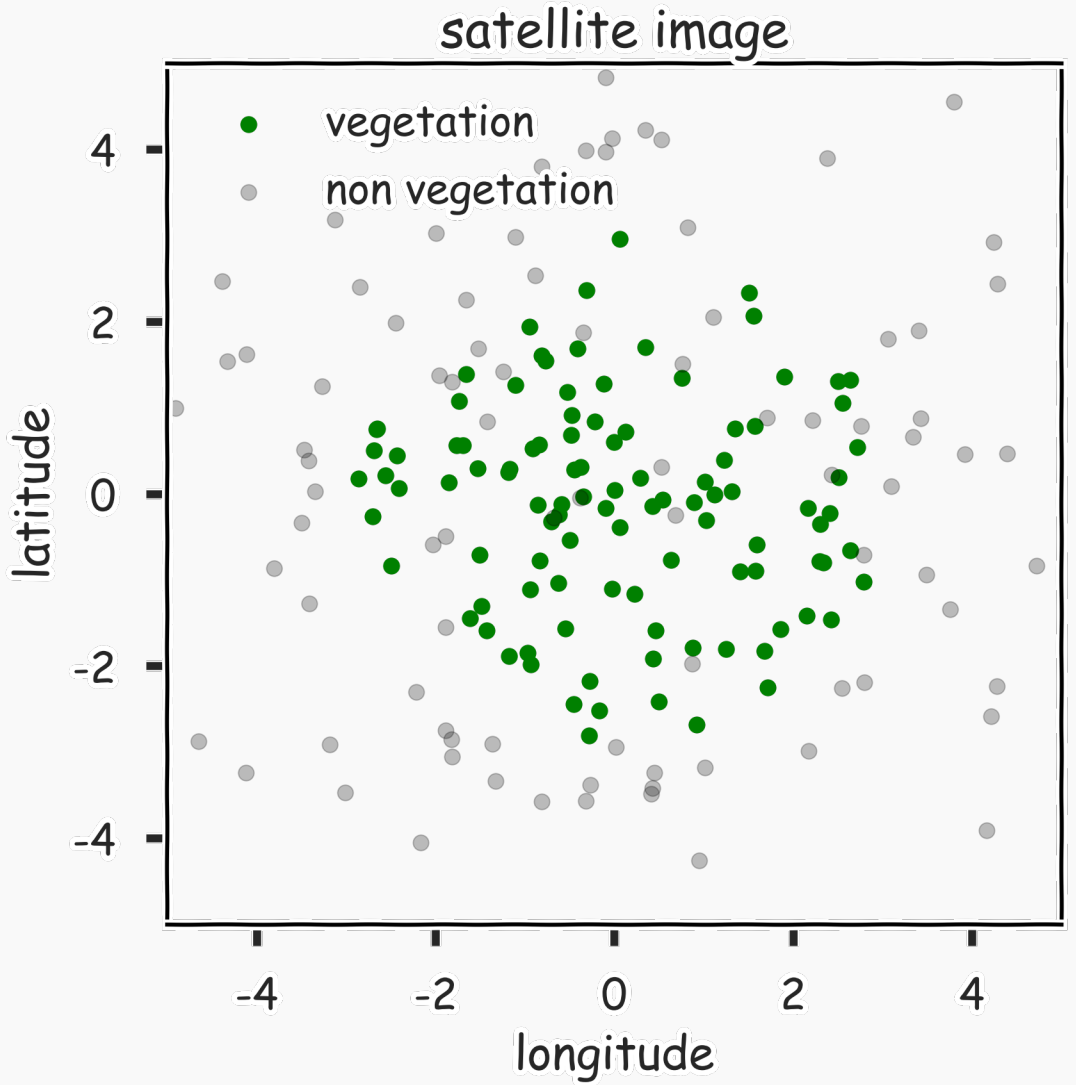
Overfitting

Same issues as with classification trees. Avoid overfitting by pruning or limiting the depth of the tree and using CV.

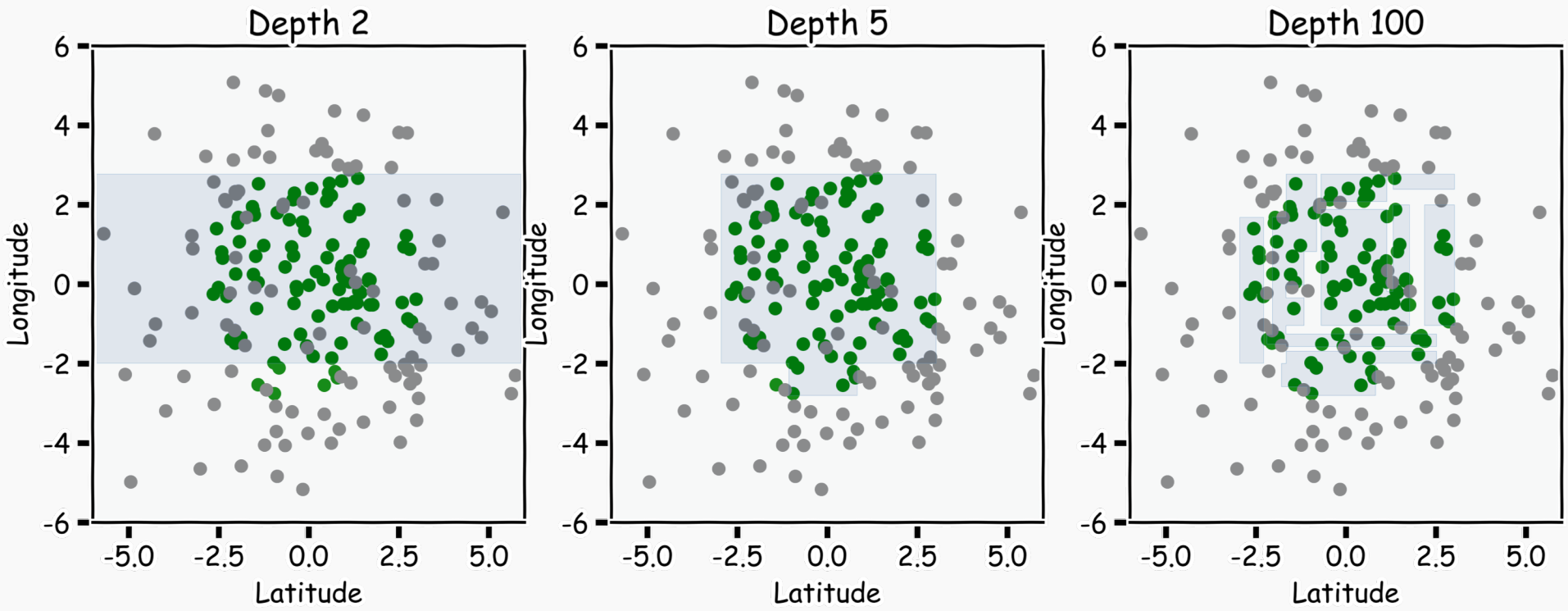


Bootstrap-Aggregating (Bagging)

Reduce the variance

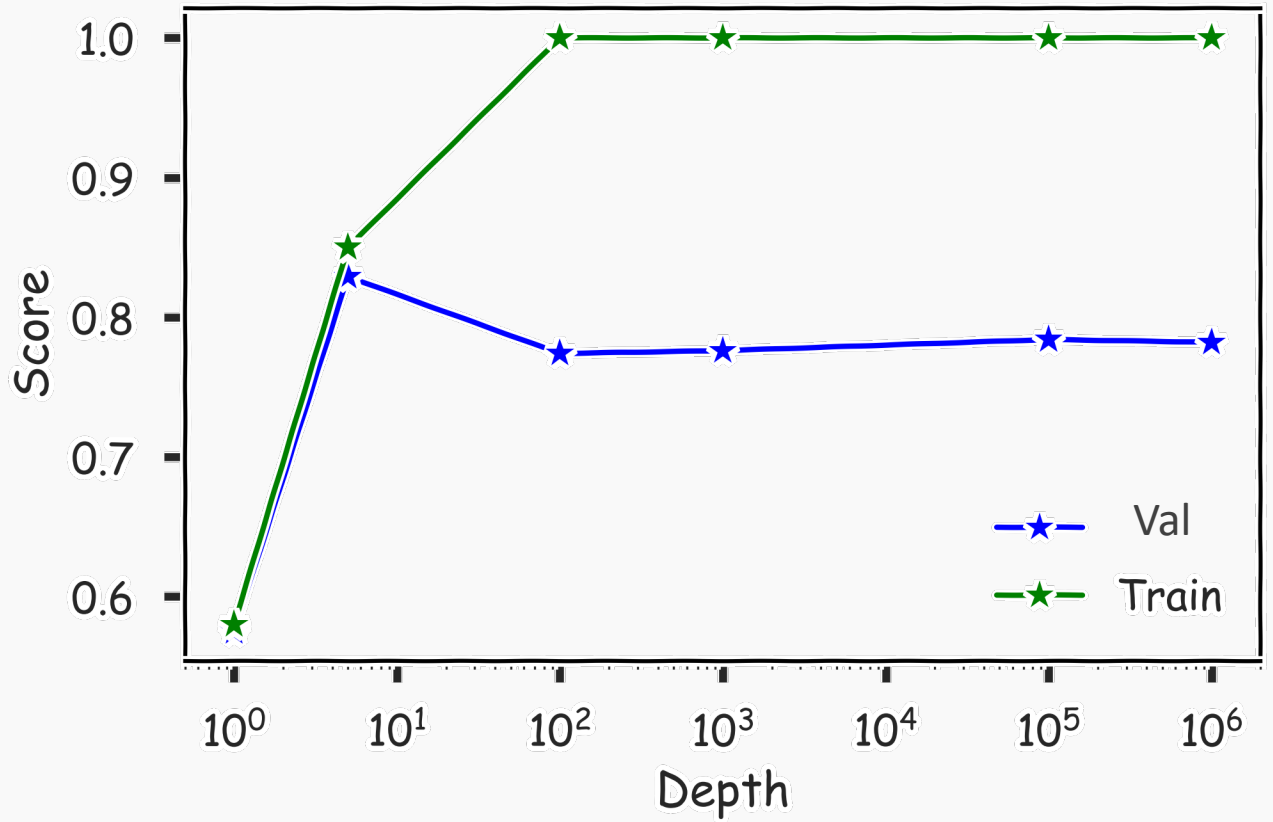


Hyper-parameters: Depth

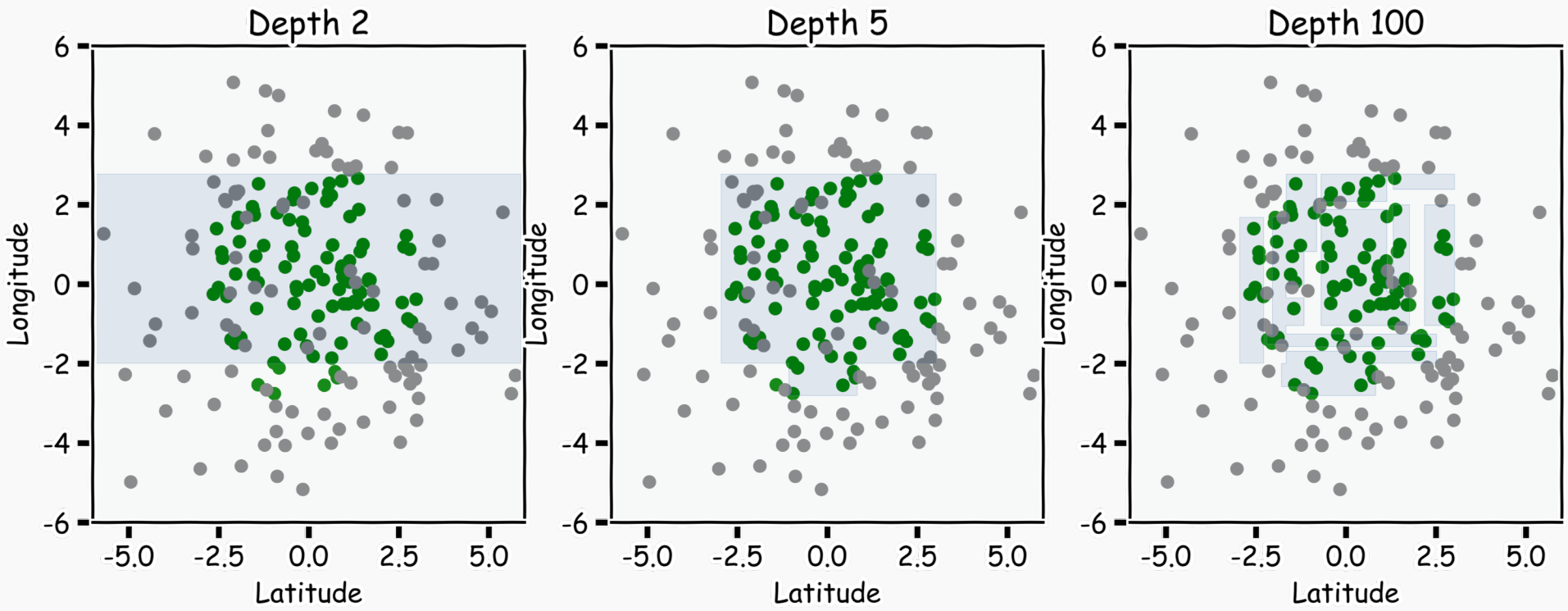


Hyper-parameters: Depth

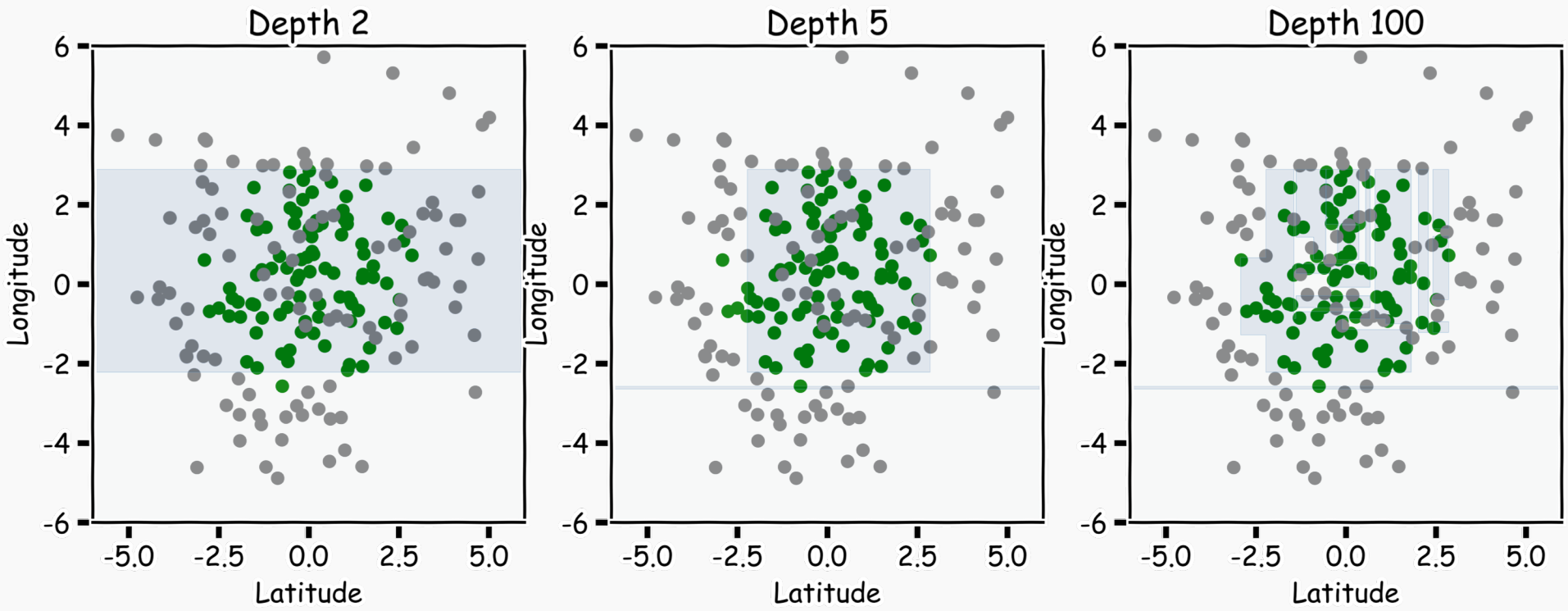
Use train/validation or cross validation to estimate the best depth.



Magic realism: Bootstrap



Magic realism: Bootstrap



Limitations of Decision Tree Models

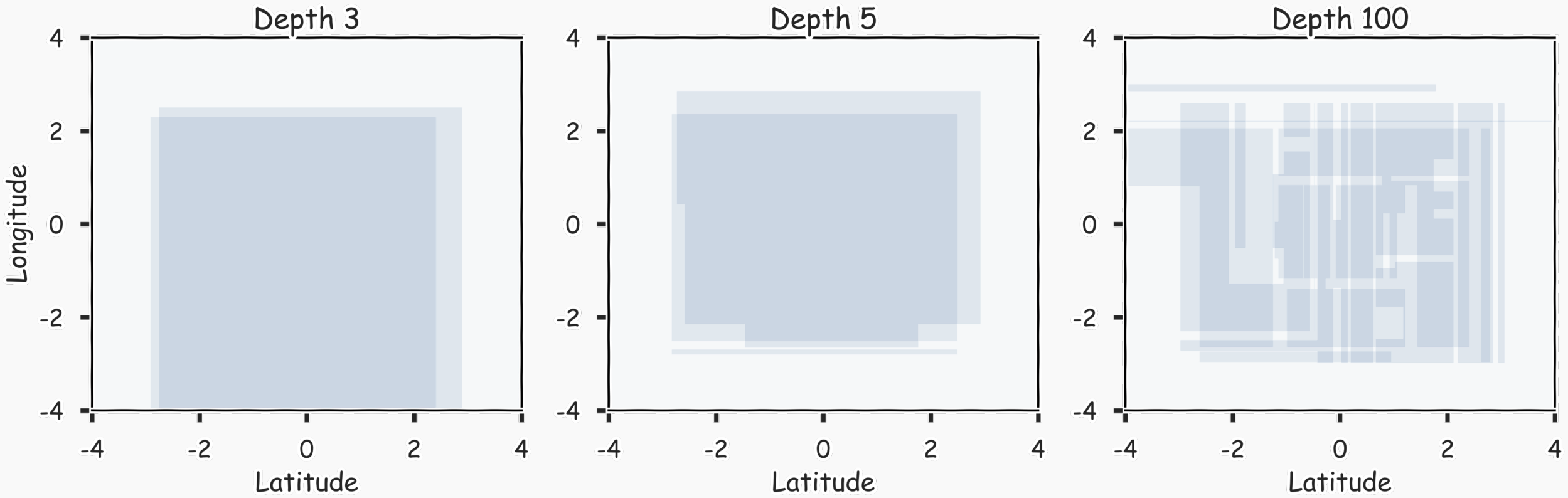
Decision trees models are highly interpretable and fast to train, using our greedy learning algorithm.

However, in order to **capture a complex decision boundary** (or to approximate a complex function), we need to use a large tree (since each time we can only make axis aligned splits).

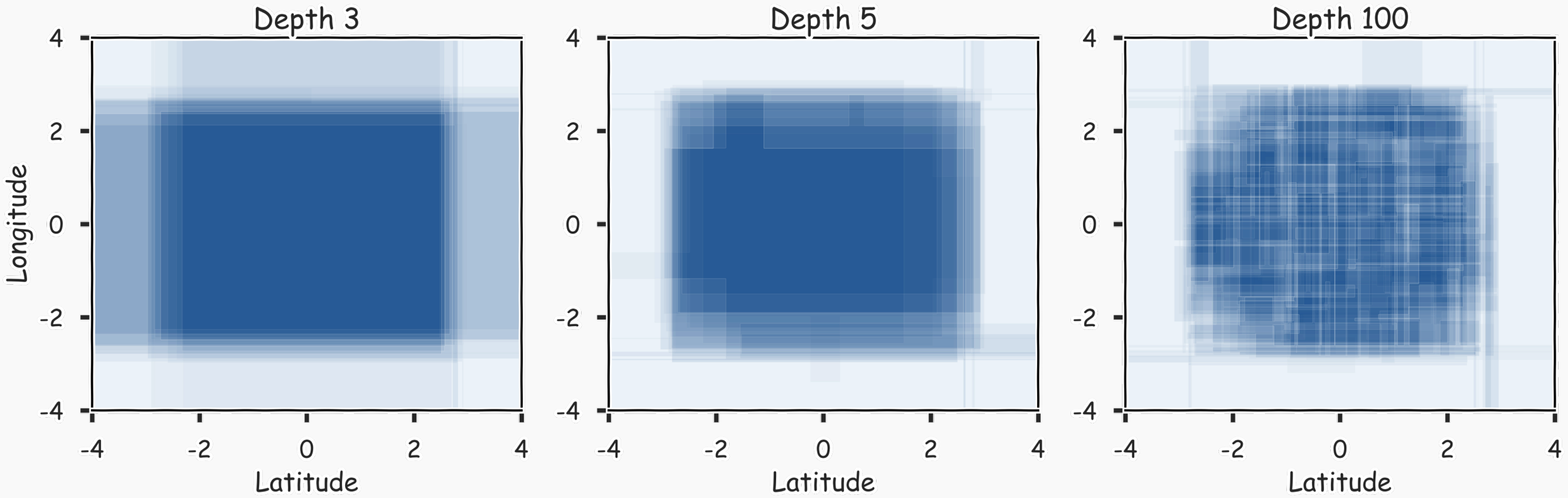
We've seen that large trees have high variance and are prone to overfitting.

For these reasons, in practice, decision tree models often underperforms when compared with other classification or regression methods.

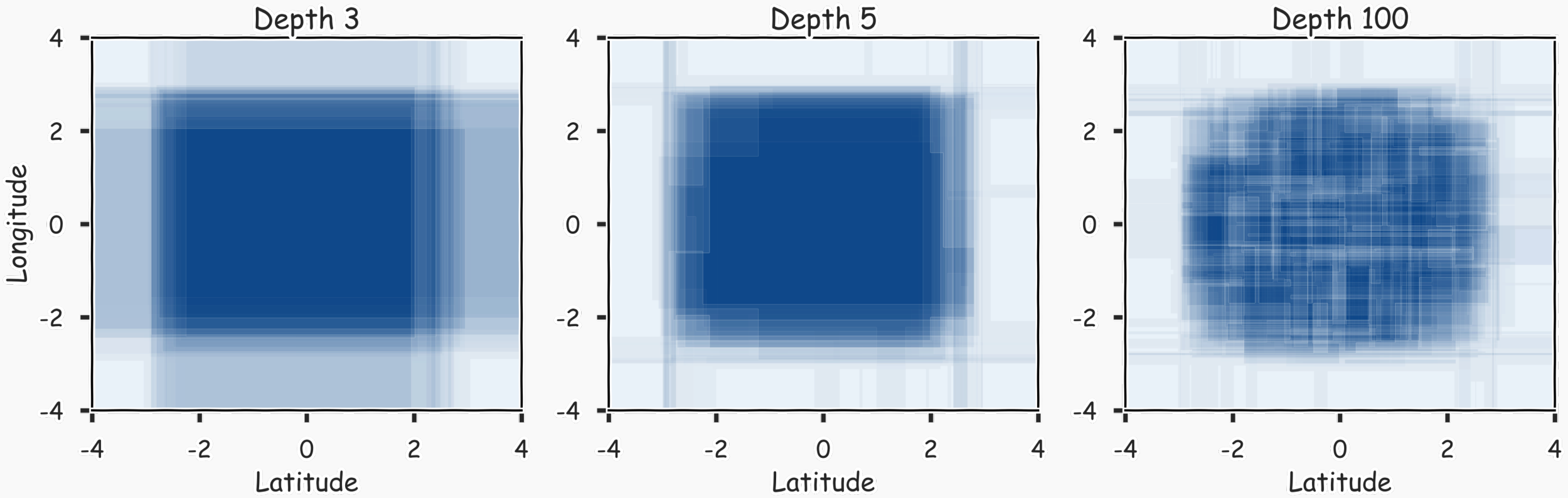
Combine them? 2 magic realisms



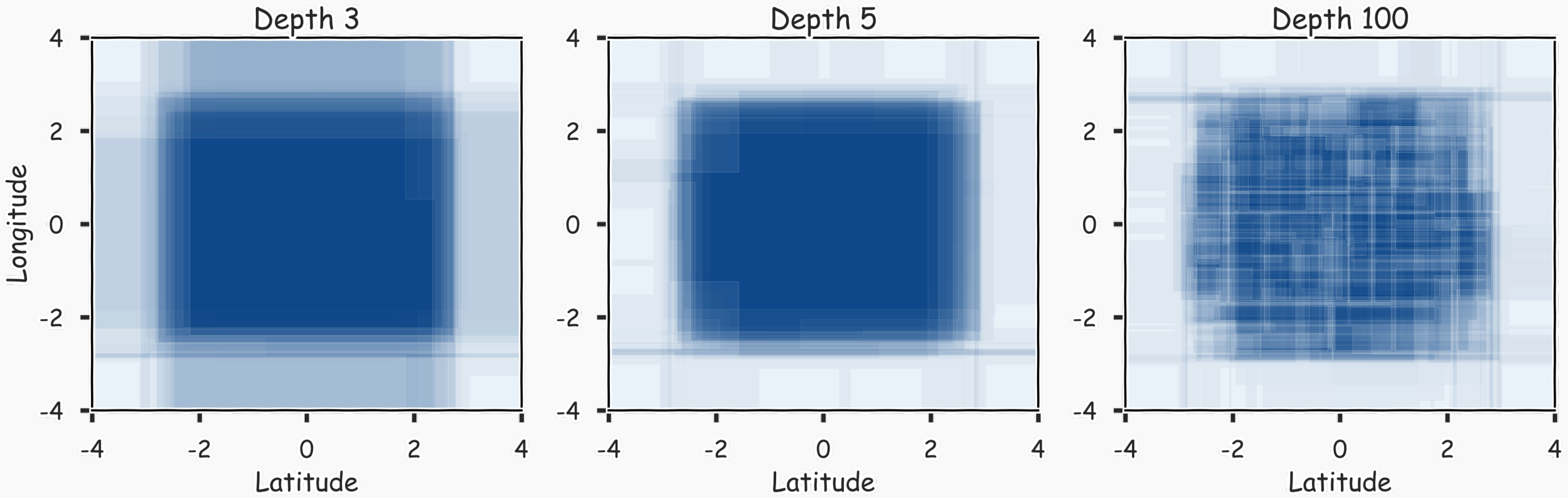
Combine them? 20 magic realisms



Combine them? 100 magic realisms



Combine them? 300 magic realisms



Bagging

One way to adjust for the high variance of the output of an experiment is to perform the experiment multiple times and then average the results.

The same idea can be applied to high variance models:

1. **(Bootstrap)**we generate multiple samples of training data, via bootstrapping. We train a full decision tree on each sample of data.
2. **(Aggregate)**for a given input, we output the averaged outputs of all the models for that input.

For classification, we return the class that is outputted by the plurality of the models. For regression we return the average of the outputs for each tree.

This method is called ***Bagging*** (Breiman, 1996), short for, of course, Bootstrap Aggregating.

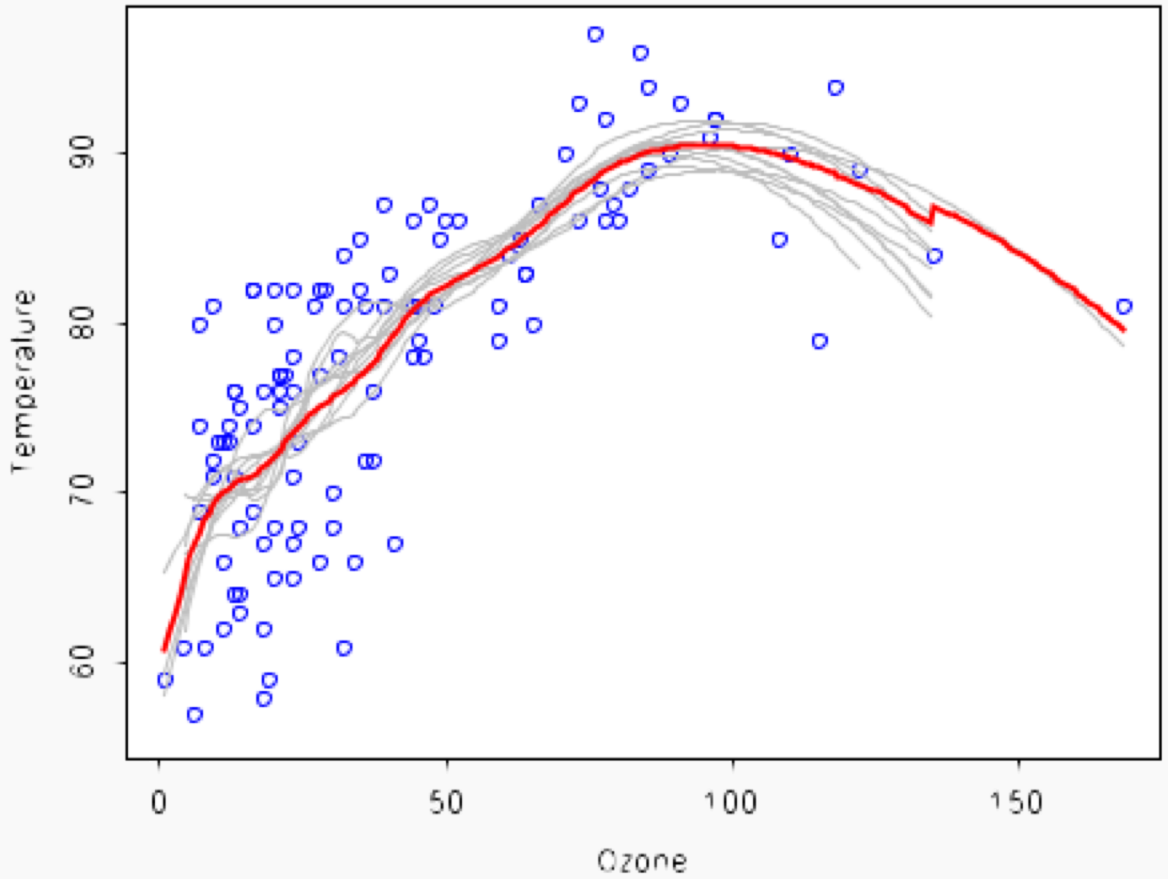
Bagging

Note that bagging enjoys the benefits of:

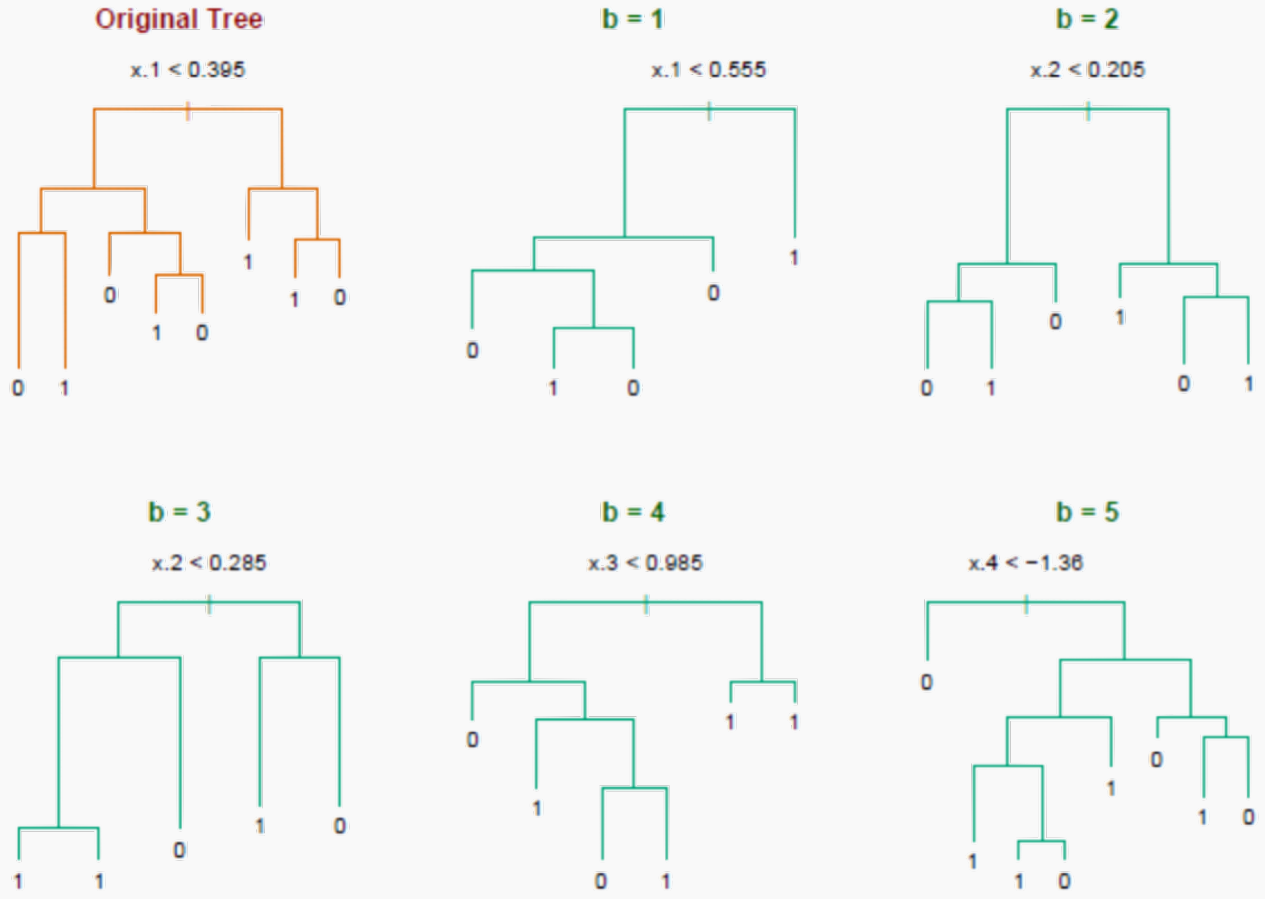
1. High expressiveness - by using full trees each model is able to approximate complex functions and decision boundaries.

2. Low variance - averaging the prediction of all the models reduces the variance in the final prediction, assuming that we choose a sufficiently large number of trees.

Bagging (regression)



Bagging (classification)



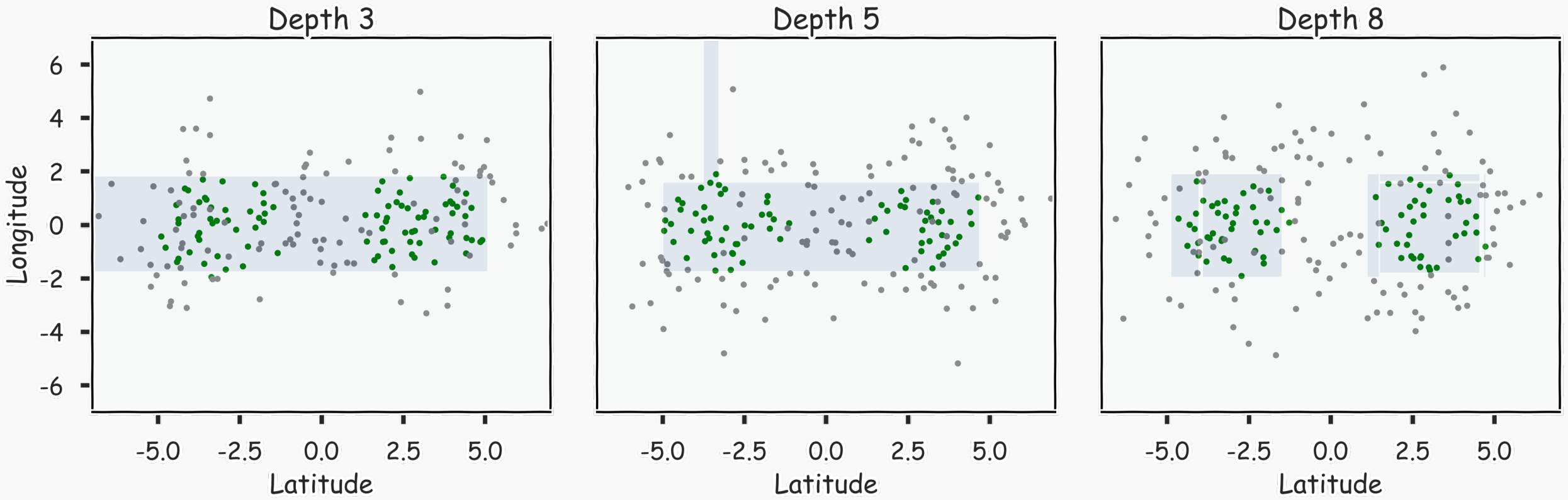
Bagging

Question: Do you see any problems?

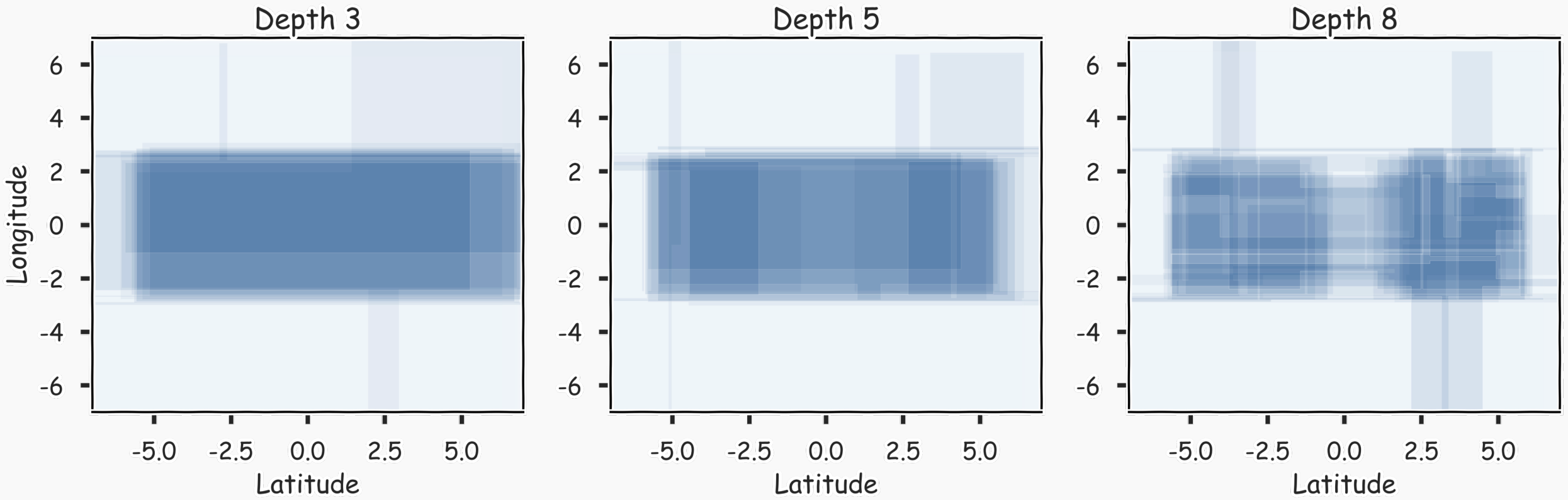
- Still some overfitting if the trees are too large.
- If trees are too shallow it can still underfits.
- Interpretability:

  The **major drawback** of bagging (and other ***ensemble methods*** that we will study) is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the 'logic' of an output through a series of decisions based on predictor values!

Case of underfitting



Case of underfitting



Bagging

Question: Do you see any problems?

- Still some overfitting if the trees are too large
- If trees are too shallow it can still underfits.

Cross Validations

Out-of-Bag Error

Bagging

Original Data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
⋮	⋮
X_n	y_n

Bootstrap Sample 1

X	Y
X_4	y_4
X_{14}	y_{14}
X_{11}	y_{11}
X_2	y_2
X_{35}	y_{35}
⋮	⋮
X_k	y_k

Decision Tree 1

Used and unused data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
⋮	⋮
X_n	y_n

Bagging

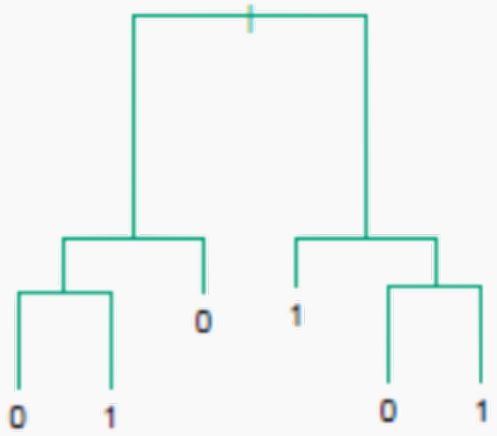
Original Data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
⋮	⋮
X_n	y_n

Bootstrap Sample 2

X	Y
X_5	y_5
X_3	y_3
X_{12}	y_{12}
X_{43}	y_{43}
X_1	y_1
⋮	⋮
X_k	y_k

Decision Tree 2



Used and unused data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
⋮	⋮
X_n	y_n

Bagging

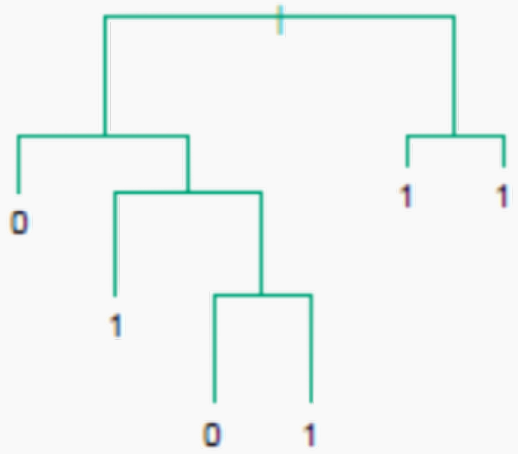
Original Data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
⋮	⋮
X_n	y_n

Bootstrap Sample 3

X	Y
X_9	y_9
X_4	y_4
X_1	y_1
X_1	y_1
X_{65}	y_{65}
⋮	⋮
X_k	y_k

Decision Tree 3



Used and unused data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
⋮	⋮
X_n	y_n

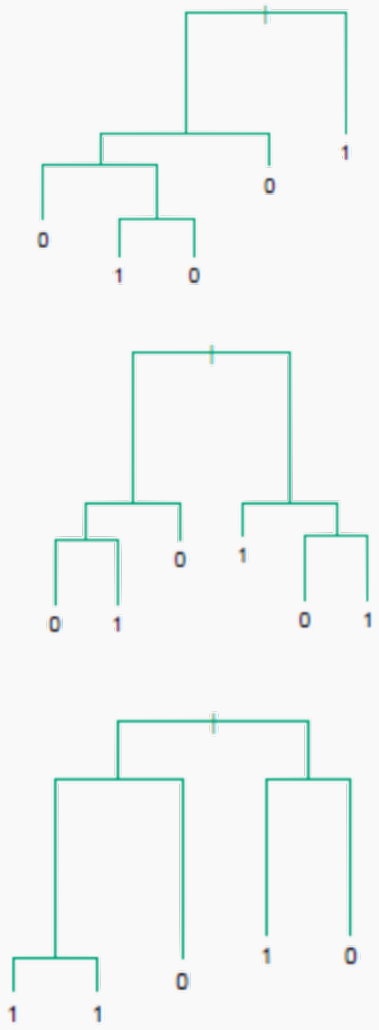
Point-wise out-of-bag error

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
⋮	⋮
X_i	y_i
⋮	⋮
X_n	y_n

Point-wise out-of-bag error

B Trees that did not see $\{X_i, y_i\}$

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
⋮	⋮
X_i	y_i
⋮	⋮
X_n	y_n



Classification

$$\hat{y}_{i,pw} = \text{majority}(\hat{y}_i)$$

$$e_i = \mathbb{I}(\hat{y}_{i,pw} \neq y_i)$$

Regression

$$\hat{y}_{i,pw} = \sum_{j \in B} \hat{y}_{i,j}$$

$$e_i = (y_i - \hat{y}_{i.pw})^2$$

OOB Error

We average the point-wise out-of-bag error over the full training set.

Classification

$$Error_{OOB} = \sum_{i}^{n} e_i = \sum_{i}^{n} \mathbb{I}(\hat{y}_{i,pw} \neq \mathrm{y_i})$$

Regression

$$Error_{OOB} = \sum_{i}^{n} e_i = \sum_{i}^{n} (y_i - \hat{y}_{i,pw})^2$$

Out-of-Bag Error

Bagging is an example of an ***ensemble method***, a method of building a single model by training and aggregating multiple models.

With ensemble methods, we get a new metric for assessing the predictive performance of the model, the ***out-of-bag error***.

Given a training set and an ensemble of models, each trained on a bootstrap sample, we compute the ***out-of-bag error*** of the averaged model by

1. For each point in the training set, we average the predicted output for this point over the models whose bootstrap training set excludes this point. We compute the error or squared error of this averaged prediction. Call this the point-wise out-of-bag error.

2. We average the point-wise out-of-bag error over the full training set.

Bagging

Question: Do you see any problems?

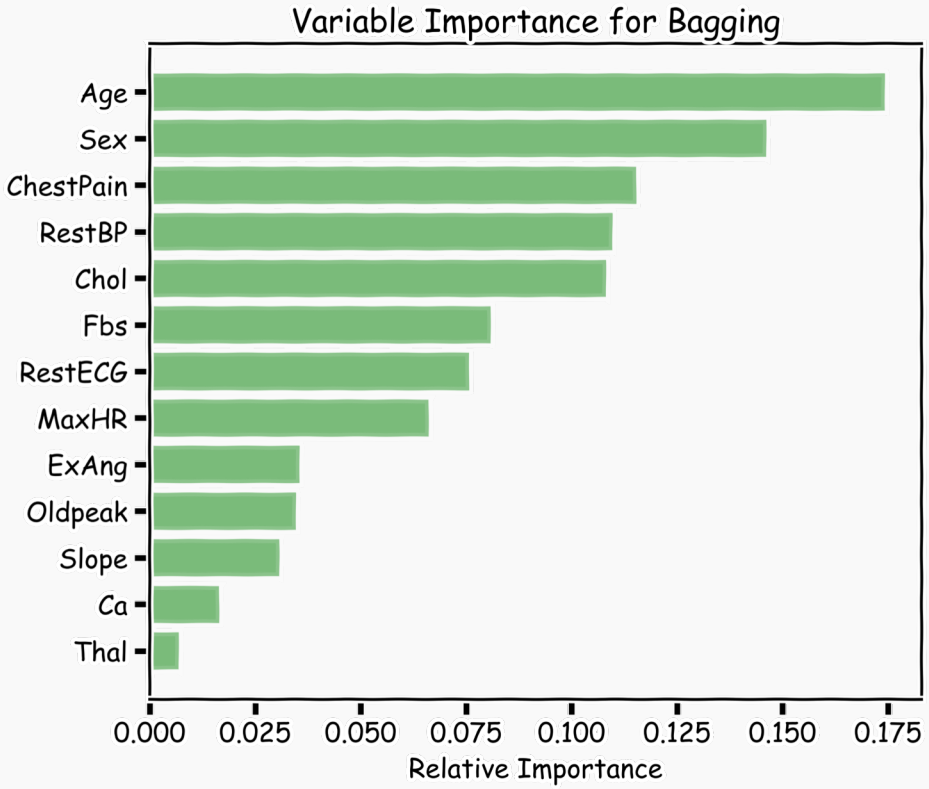
- Still some overfitting if the trees are too large.
- If trees are too shallow it can still underfits.
- Interpretability:

  The **major drawback** of bagging (and other ***ensemble methods*** that we will study) is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the 'logic' of an output through a series of decisions based on predictor values!

Variable Importance for Bagging

Bagging improves prediction accuracy at the expense of interpretability.

Calculate the total amount that the MSE (for regression) or Gini index (for classification) is decreased due to splits over a given predictor, averaged over all B trees.



100 trees, max_depth=10

Improving on Bagging

In practice, the ensembles of trees in Bagging tend to be highly correlated.

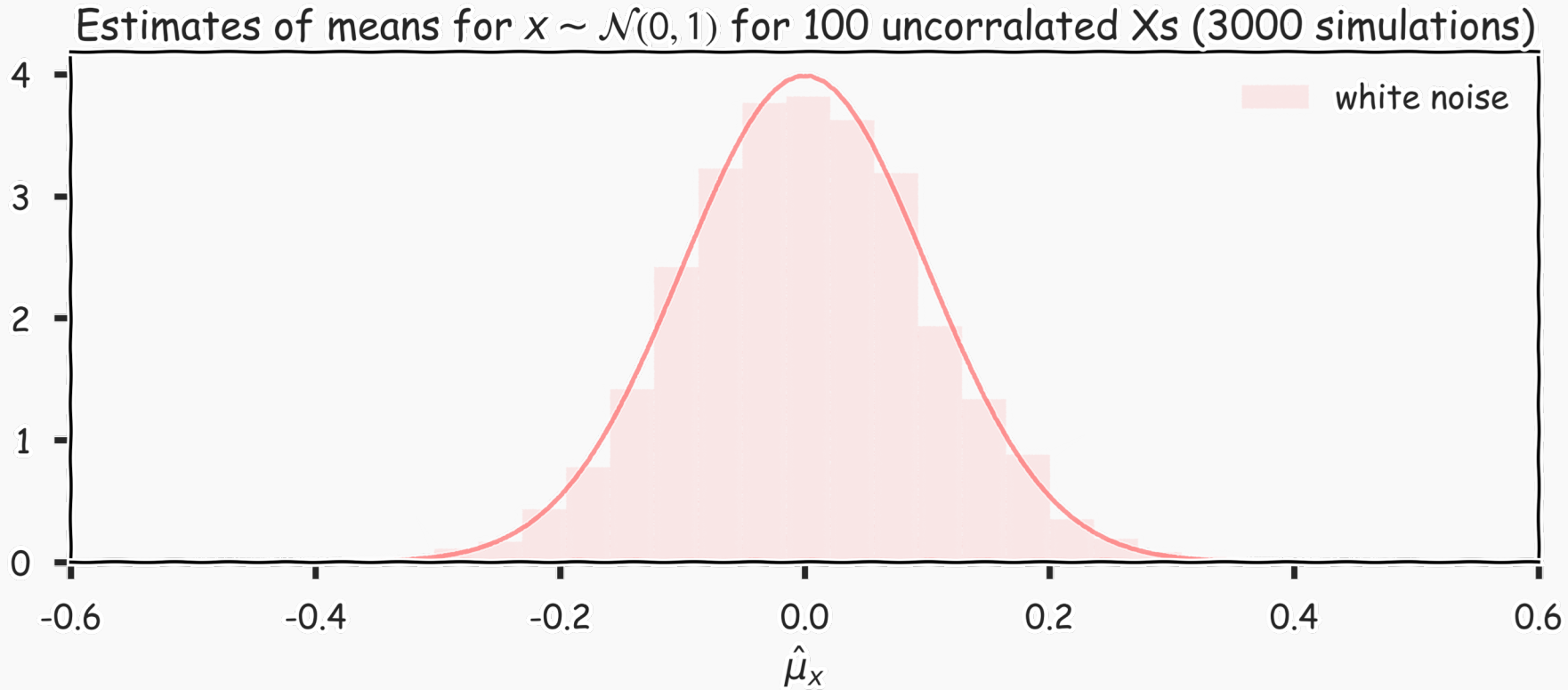
Suppose we have an extremely strong predictor, x_j , in the training set amongst moderate predictors. Then the greedy learning algorithm ensures that most of the models in the ensemble will choose to split on x_j in early iterations.

That is, each tree in the ensemble is identically distributed, with the expected output of the averaged model the same as the expected output of any one of the trees.

Improving on Bagging

Recall, for B number of identically and independently distributed variable, X, with variance σ^2, the variance of the estimate of the mean is :

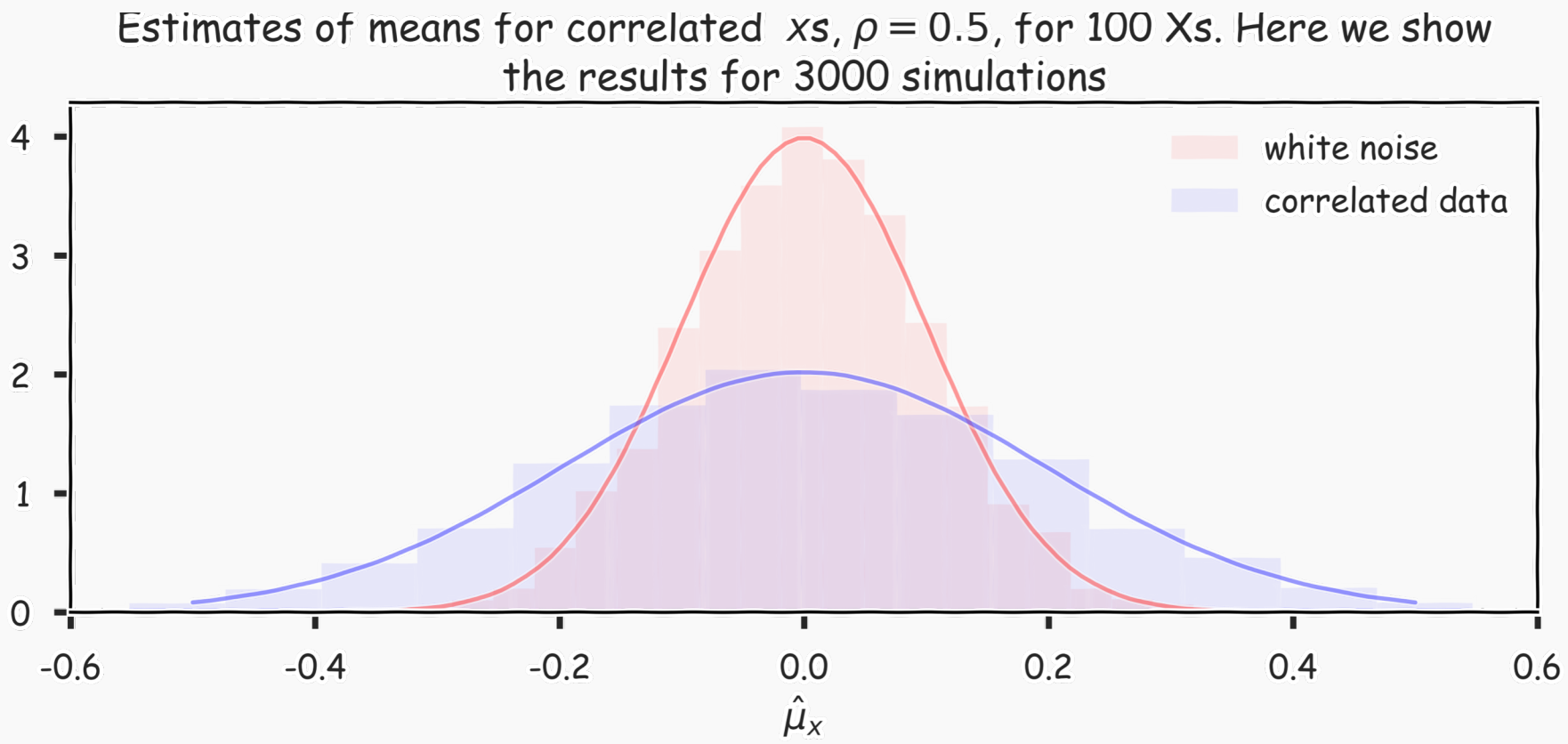
$$\text{var}(\hat{\mu}_x) = \frac{\sigma^2}{B}$$



Improving on Bagging

For B number of identically but not independently distributed variables with pairwise correlation ρ and variance σ^2, the variance of their mean is

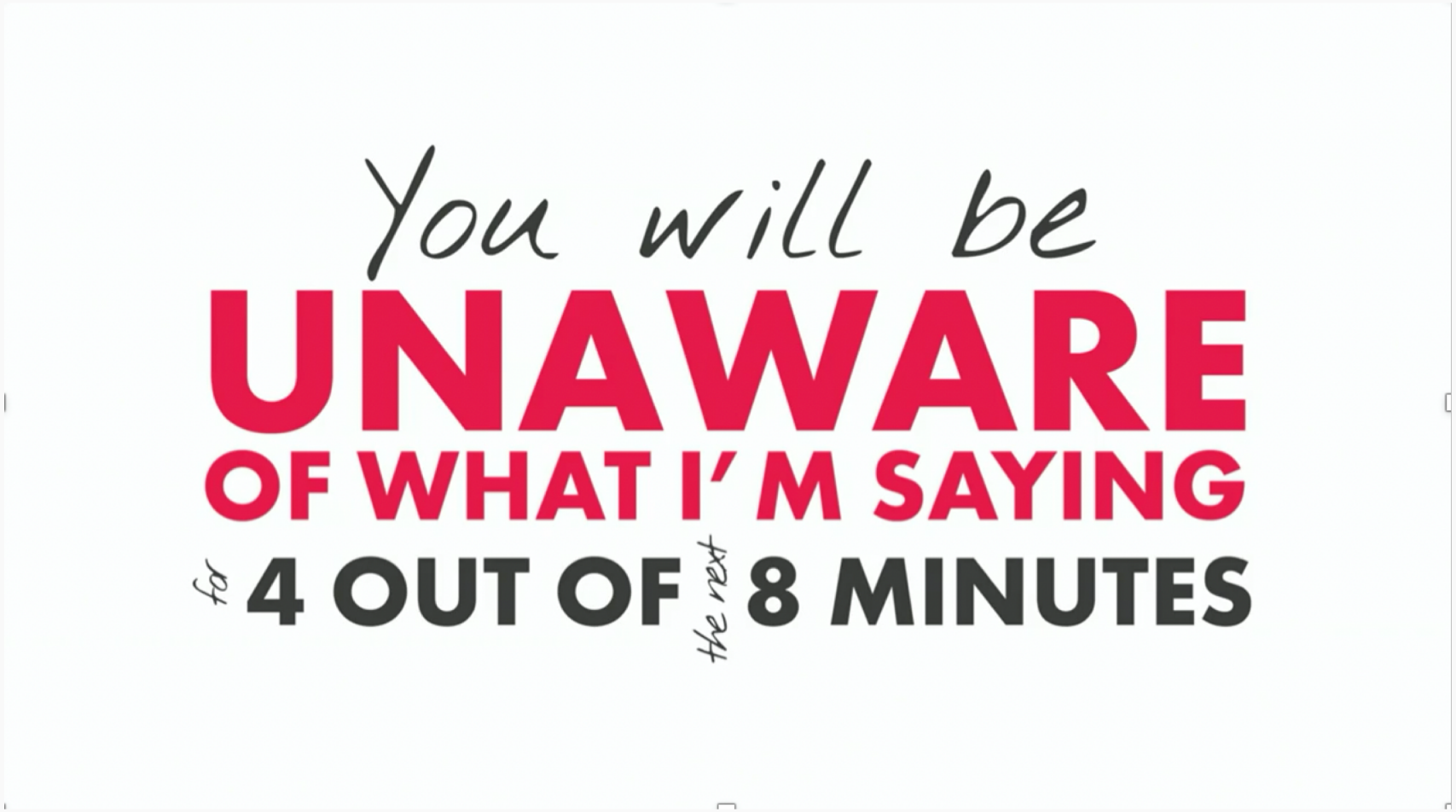
$$\text{var}(\hat{\mu}_x) \propto \sigma^2(1 + \rho^2)/B$$



Bagging

Question: Do you see any problems?

- Still some overfitting if the trees are too large
- If trees are too shallow it can still underfits.
- interpretability
- The **major drawback** of bagging (and other ***ensemble methods*** that we will study) is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the 'logic' of an output through a series of decisions based on predictor values!



Zeenat Potia

Random Forests

Random Forests

Random Forest is a modified form of bagging that creates ensembles of independent decision trees.

To de-correlate the trees, we:

1. train each tree on a separate bootstrap sample of the full training set (same as in bagging)

2. for each tree, at each split, we ***randomly*** select a set of J' predictors from the full set of predictors.

From amongst the J' predictors, we select the optimal predictor and the optimal corresponding threshold for the split.

Tuning Random Forests

Random forest models have multiple hyper-parameters to tune:

1. the number of predictors to randomly select at each split
2. the total number of trees in the ensemble
3. the minimum leaf node size

In theory, each tree in the random forest is full, but in practice this can be computationally expensive (and added redundancies in the model), thus, imposing a minimum node size is not unusual.

Tuning Random Forests

There are standard (default) values for each of random forest hyper-parameters recommended by long time practitioners, but generally these parameters should be tuned through **OOB** (making them data and problem dependent).

e.g. number of predictors to randomly select at each split:

- $\sqrt{N_j}$ for classification
- $\frac{N}{3}$ for regression

Using out-of-bag errors, training and cross validation can be done in a single sequence - we cease training once the out-of-bag error stabilizes

Variable Importance for RF

Same as with Bagging:

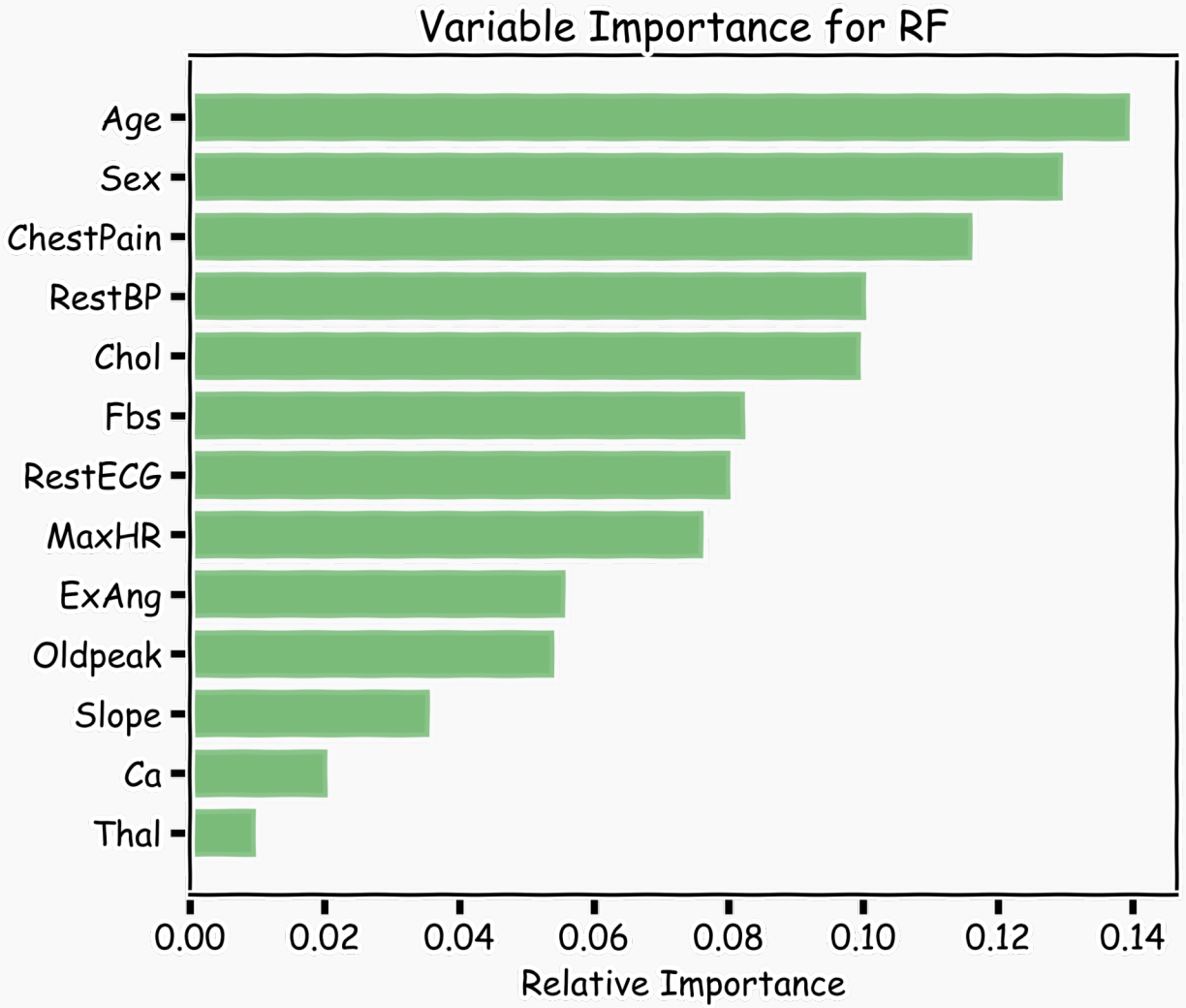
Calculate the total amount that the RSS (for regression) or Gini index (for classification) is decreased due to splits over a given predictor, averaged over all B trees.

Variable Importance for RF

Alternative:

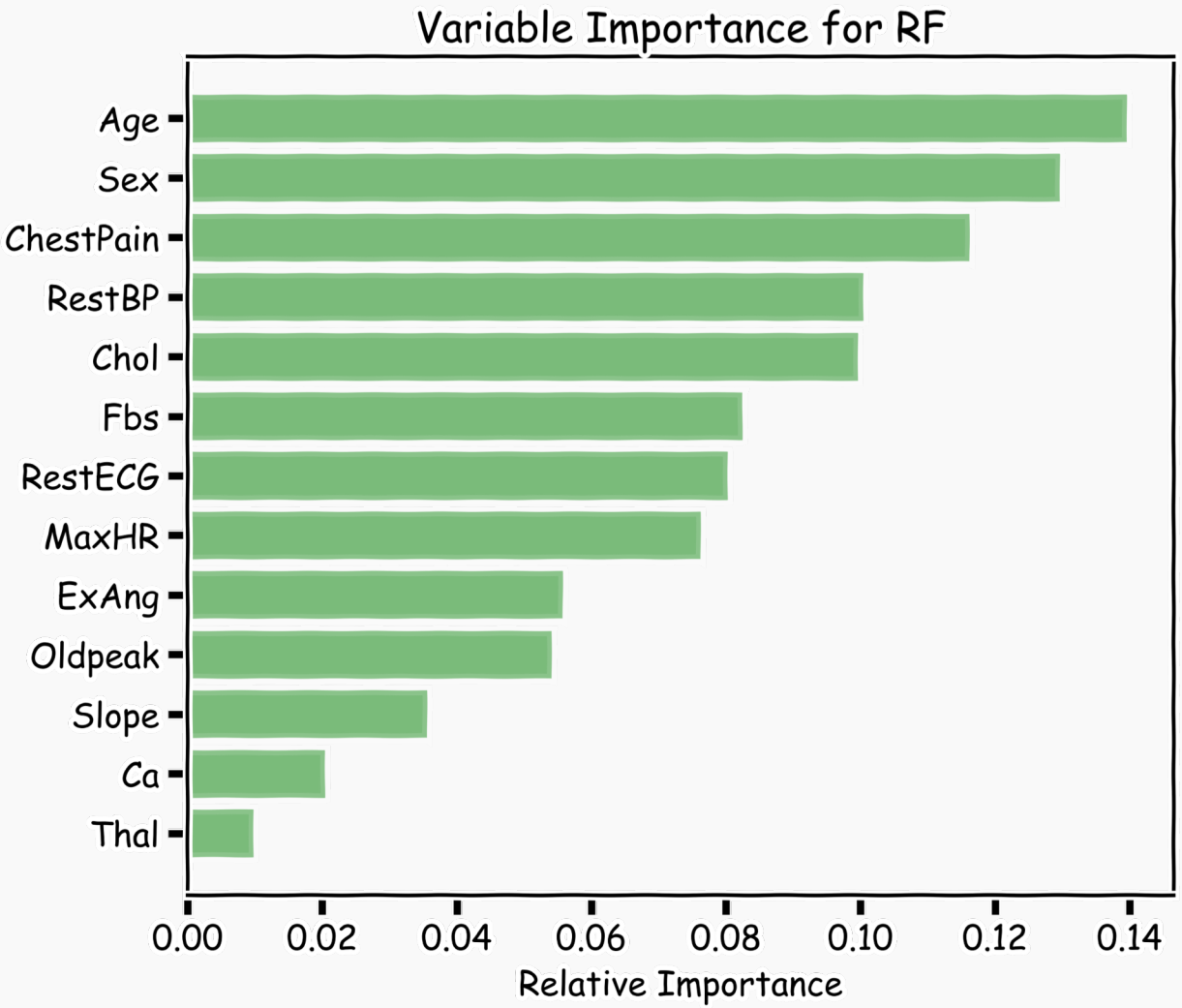
- Record the prediction accuracy on the *oob* samples for each tree.
- Randomly permute the data for column j in the *oob* samples the record the accuracy again.
- The decrease in accuracy as a result of this permuting is averaged over all trees, and is used as a measure of the importance of variable j in the random forest.

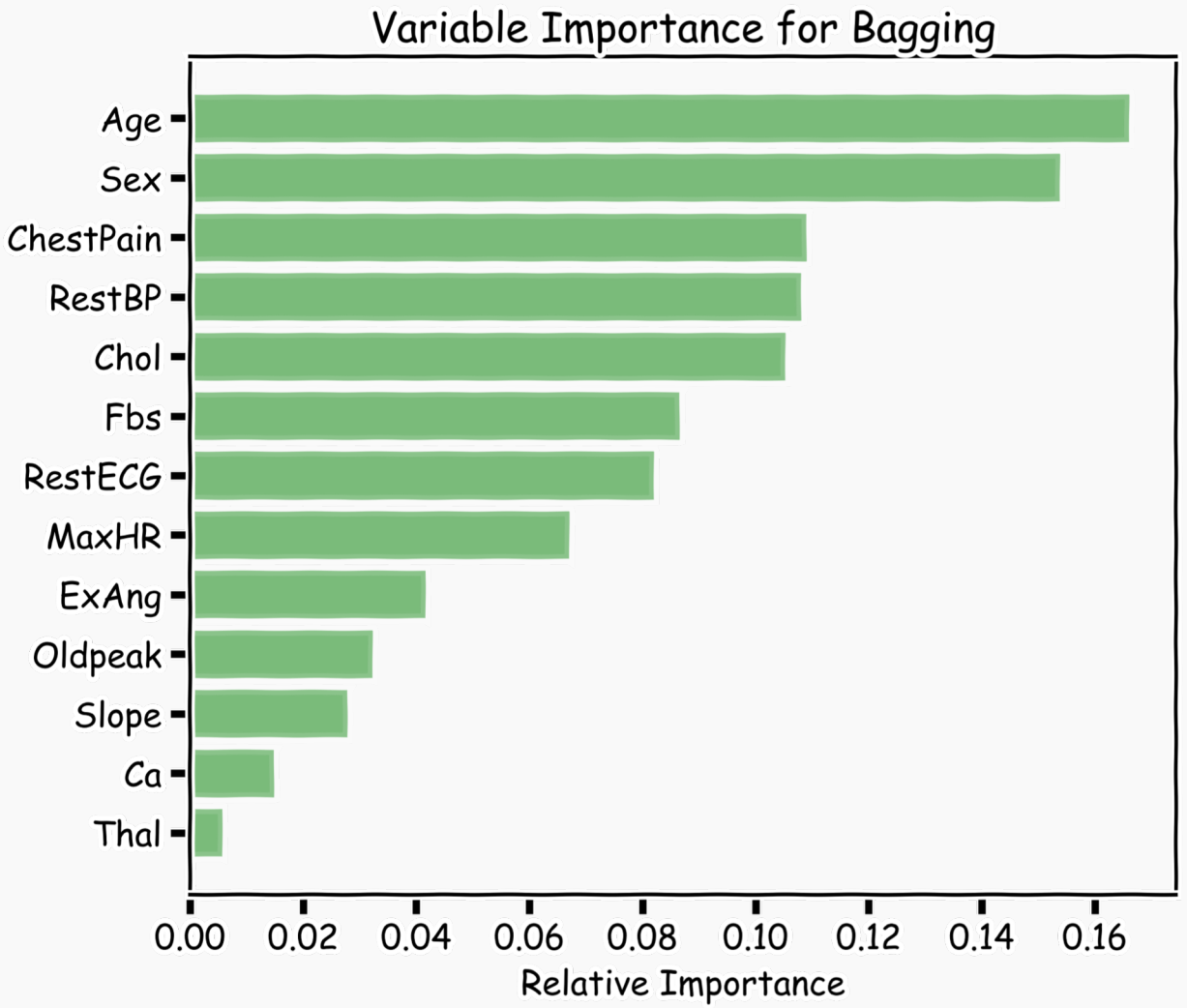
Variable Importance for RF



100 trees, max_depth=10

Variable Importance for RF





100 trees, max_depth=10

Final Thoughts on Random Forests

When the number of predictors is large, but the number of relevant predictors is small, random forests can perform poorly.

Question: Why?

In each split, the chances of selected a relevant predictor will be low and hence most trees in the ensemble will be weak models.

Final Thoughts on Random Forests (cont.)

Increasing the number of trees in the ensemble generally does not increase the risk of overfitting.

Again, by decomposing the generalization error in terms of bias and variance, we see that increasing the number of trees produces a model that is at least as robust as a single tree.

However, if the number of trees is too large, then the trees in the ensemble may become more correlated, increase the variance.

Final Thoughts on Random Forests (cont.)

Probabilities:

- Random Forrest Classifier (and bagging, and not fully complex trees) can return probabilities.
- **Question**: How?