SOICT

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Learning using Decision Trees

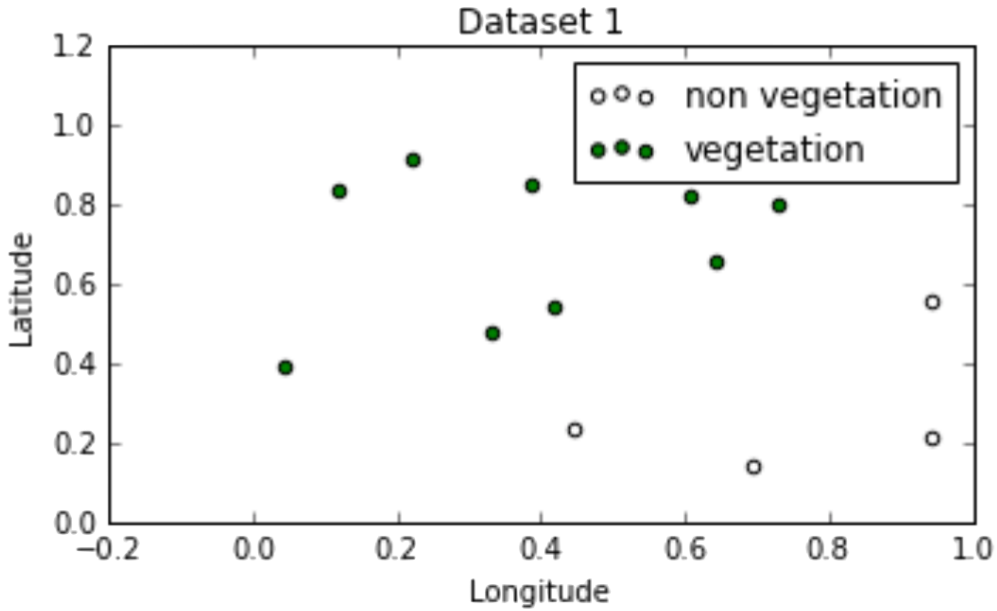
Dam Quang Tuan

Geometry of Data

Recall that:

logistic regression for building classification boundaries works best when:

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



Geometry of Data

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

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

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

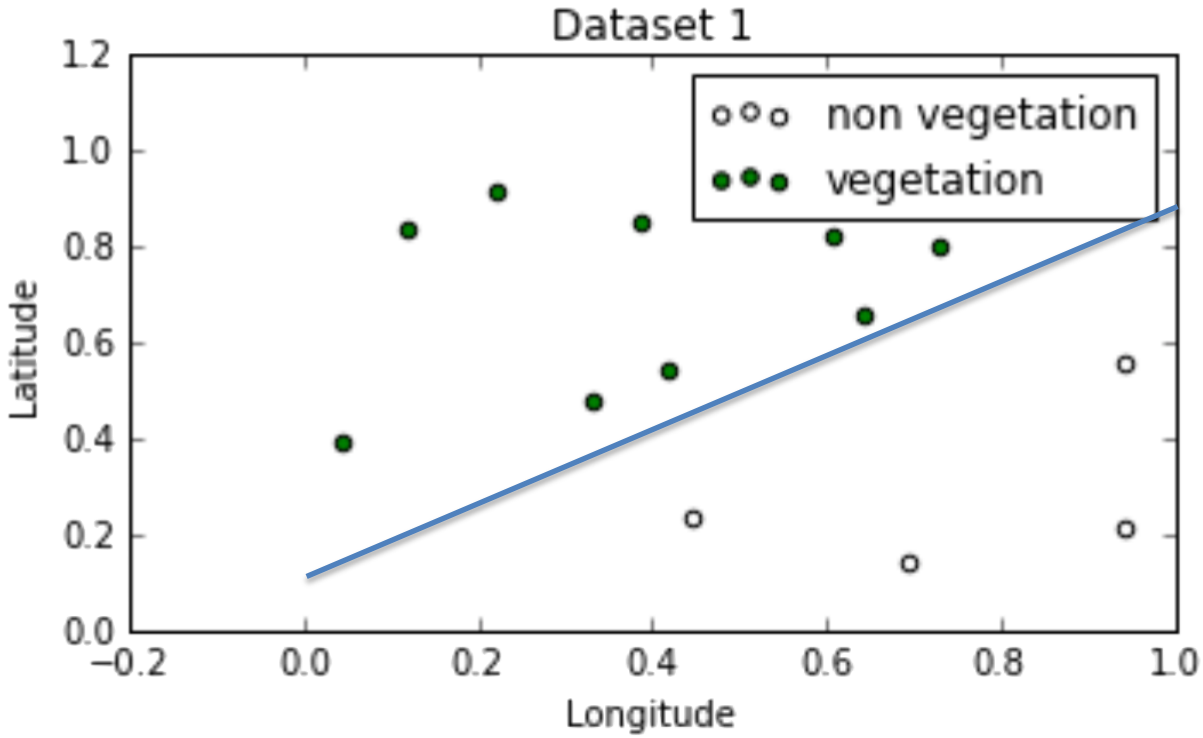
$$X\beta = 0$$

This equation defines a line or a hyperplane.

And it can be *generalized* with higher order polynomial terms.

Geometry of Data

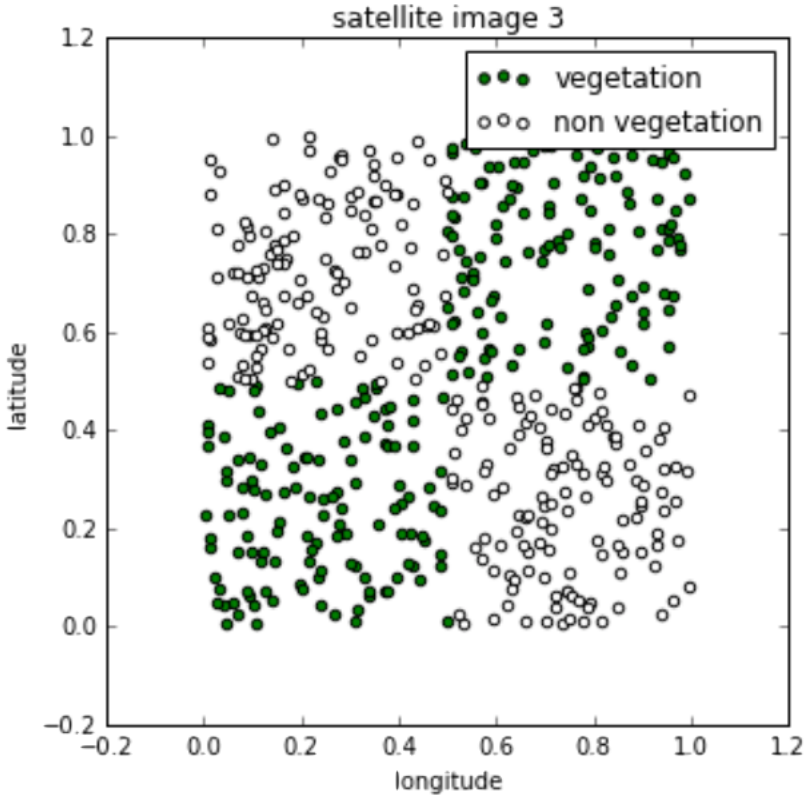
Question: Can you guess the equation that defines the decision boundary below?

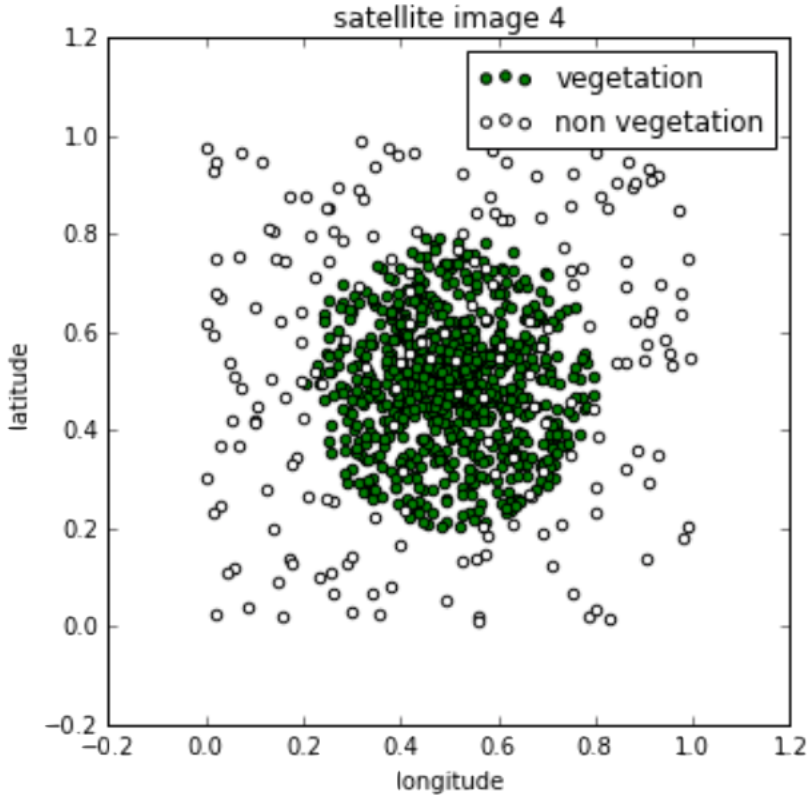


$Latitude = 0.8\ Long + 0.1\ \ or\ \ -0.8x_1 + x_2\ -\ 0.1 = 0$

Geometry of Data

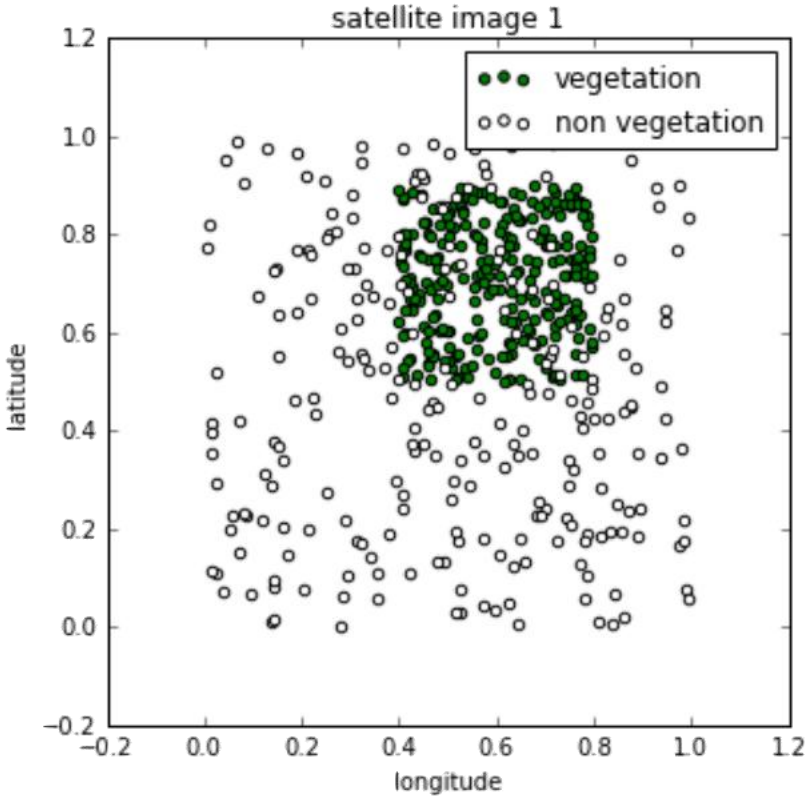
Question: How about these?

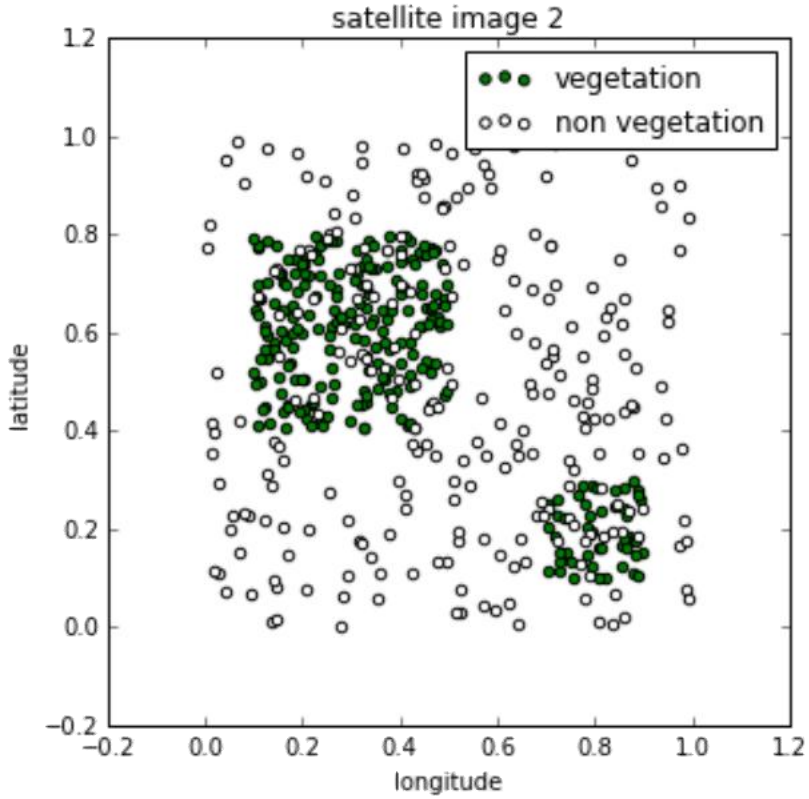




Geometry of Data

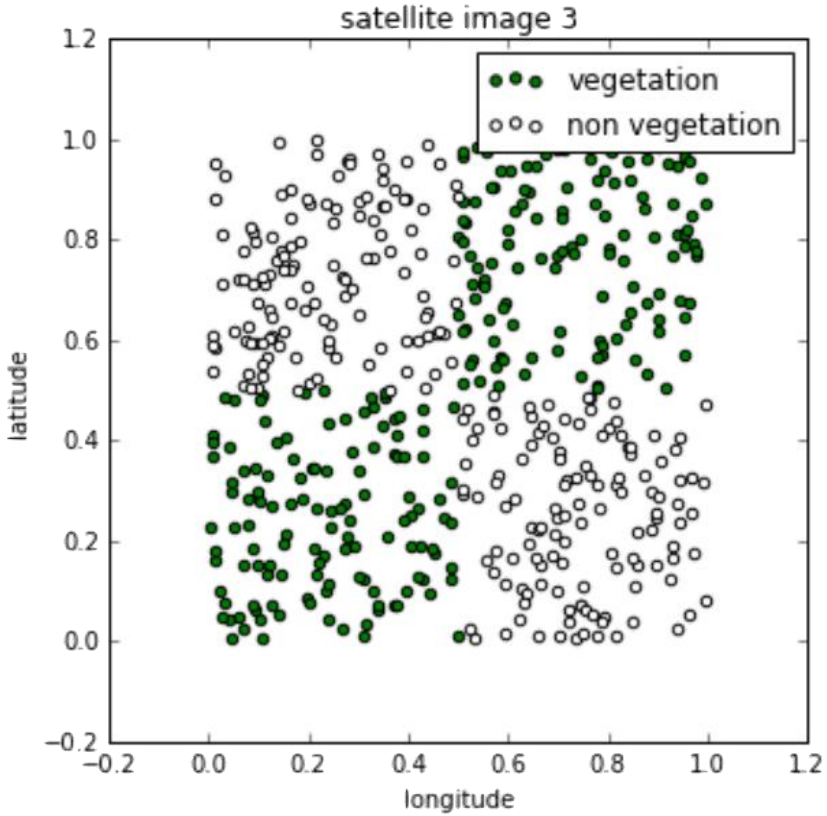
Question: Or these?





Geometry of Data

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



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:

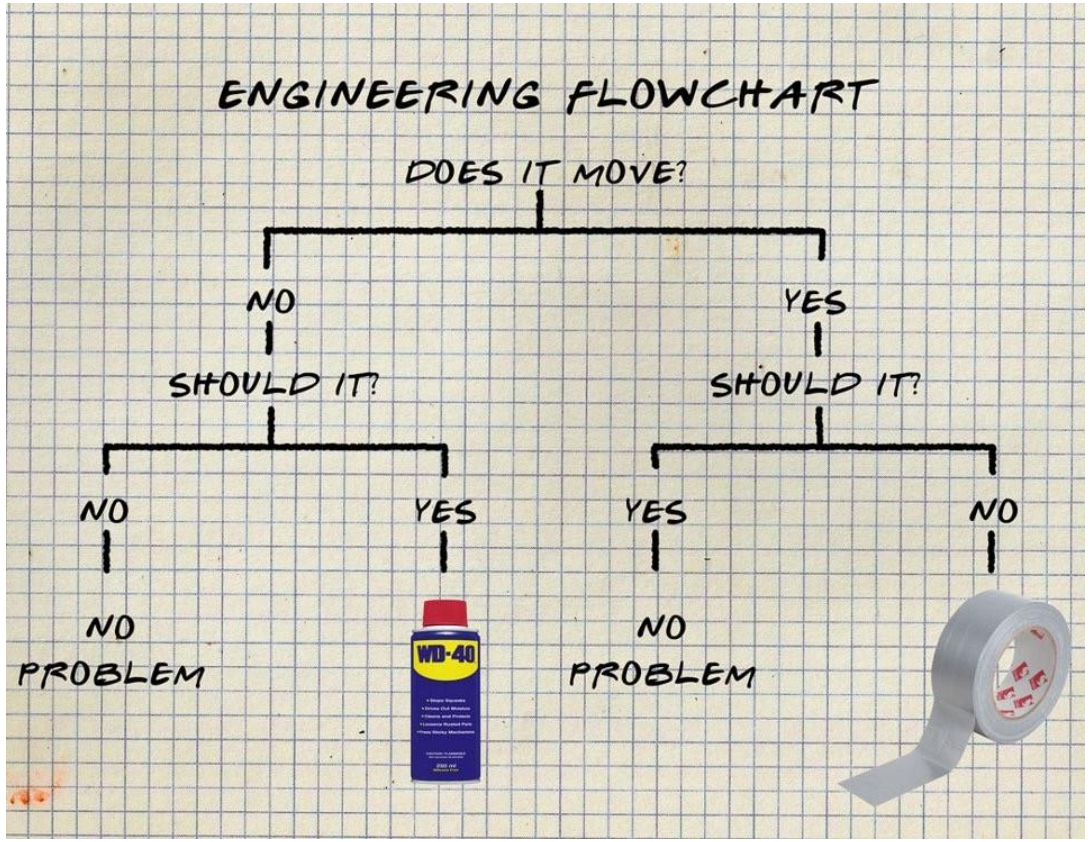
$$(x_3+2x_2) - x_1^2 + 10 = 0$$

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:



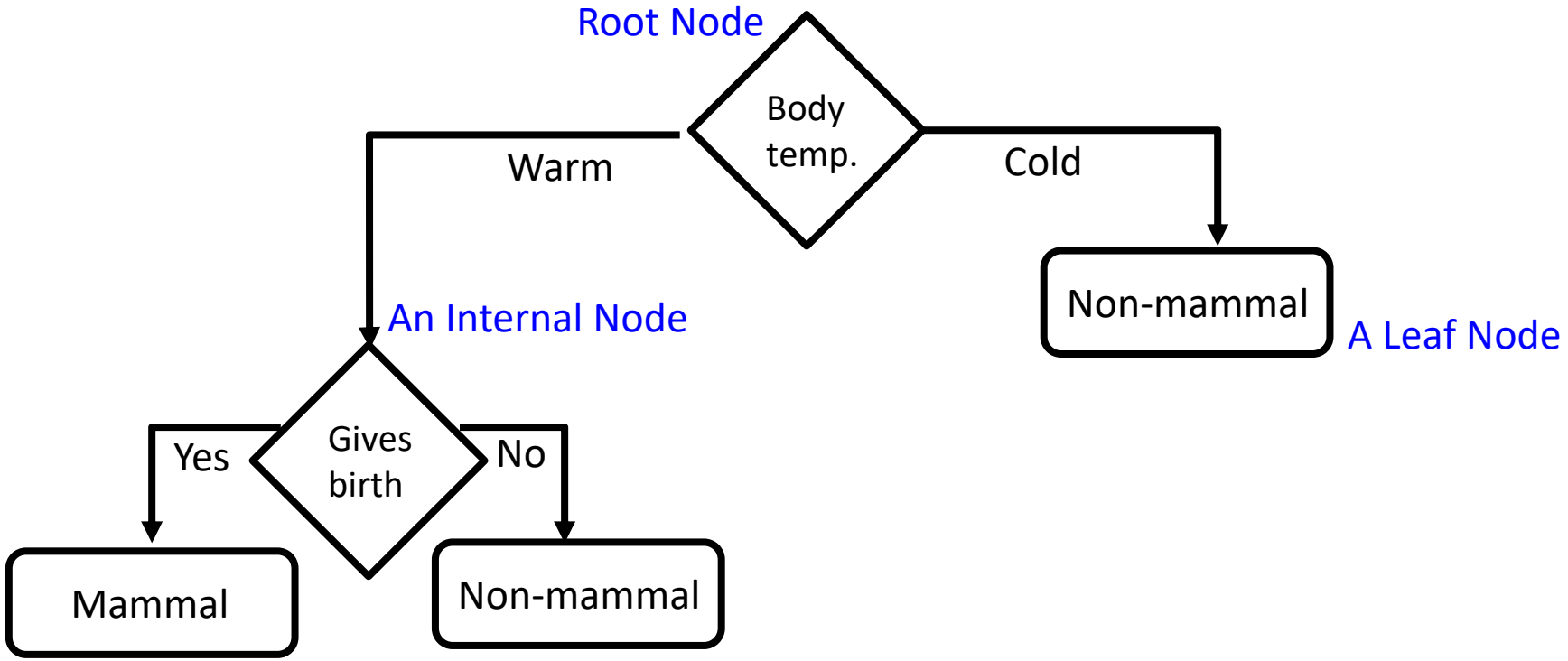
Decision Trees

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

- A Decision Tree (DT) defines a hierarchy of rules to make a prediction



- Root and internal nodes test rules. Leaf nodes make predictions

- Decision Tree (DT) learning is about learning such a tree from labeled data

A decision tree friendly problem

Loan approval prediction

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

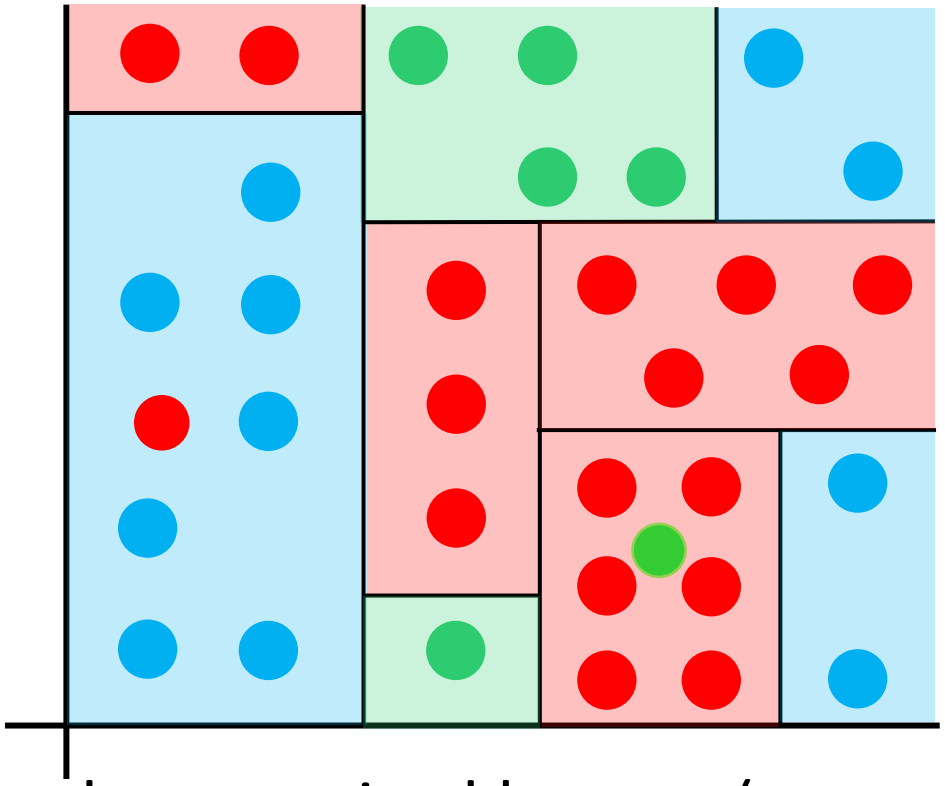
Learning Decision Trees with Supervision

- The basic idea is very simple

- Recursively partition the training data into homogeneous regions

What do you mean by "homogeneous" regions?

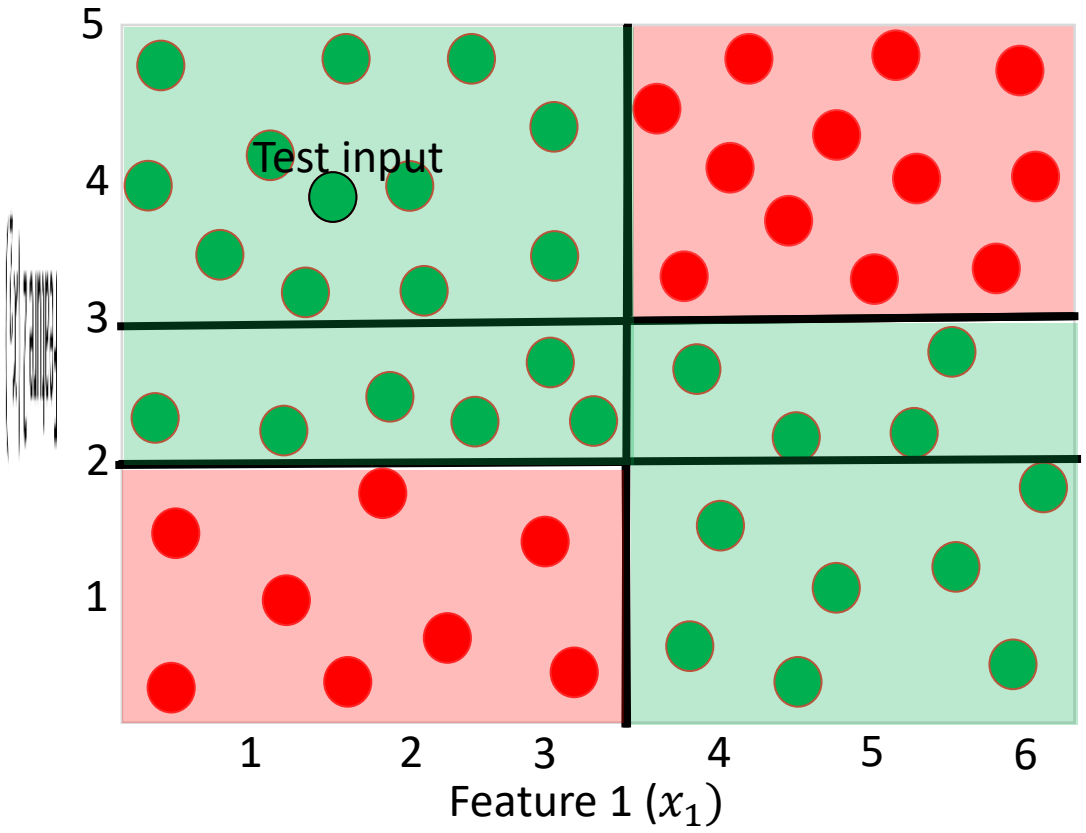
A homogeneous region will have all (or a majority of) training inputs with the same/similar outputs

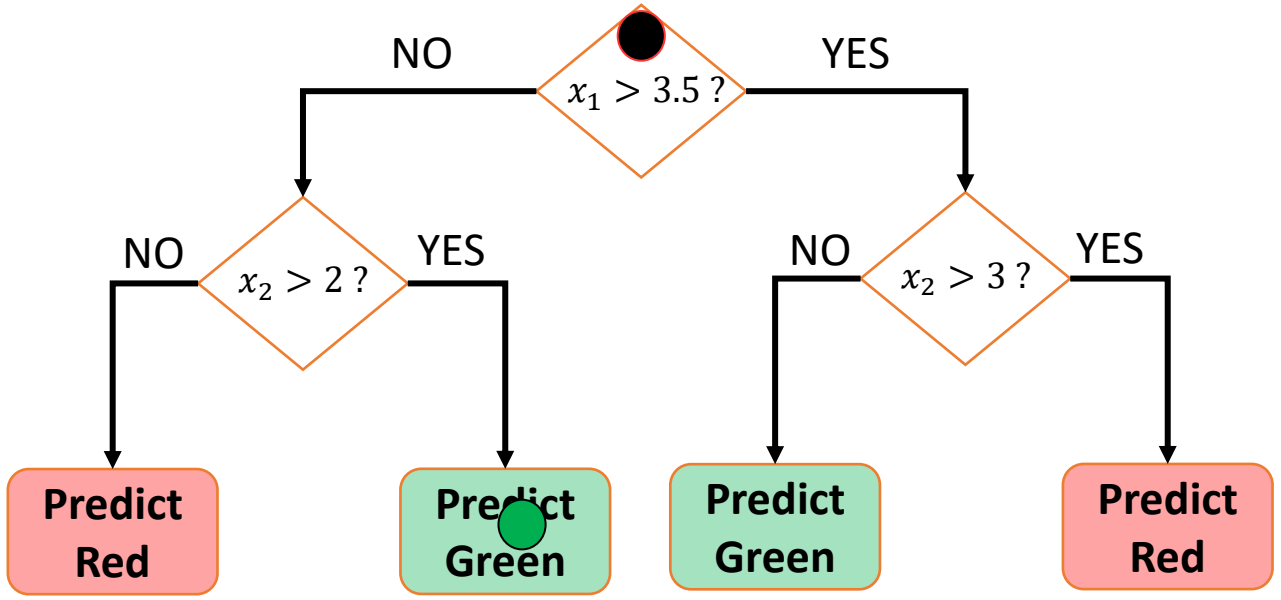


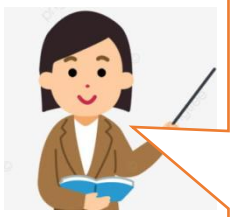
Even though the rule within each group is simple, we are able to learn a fairly sophisticated model overall (note in this example, each rule is a simple horizontal/vertical classifier but the overall decision boundary is rather sophisticated)

- Within each group, fit a simple supervised learner (e.g., predict the majority label)

Decision Trees for Classification







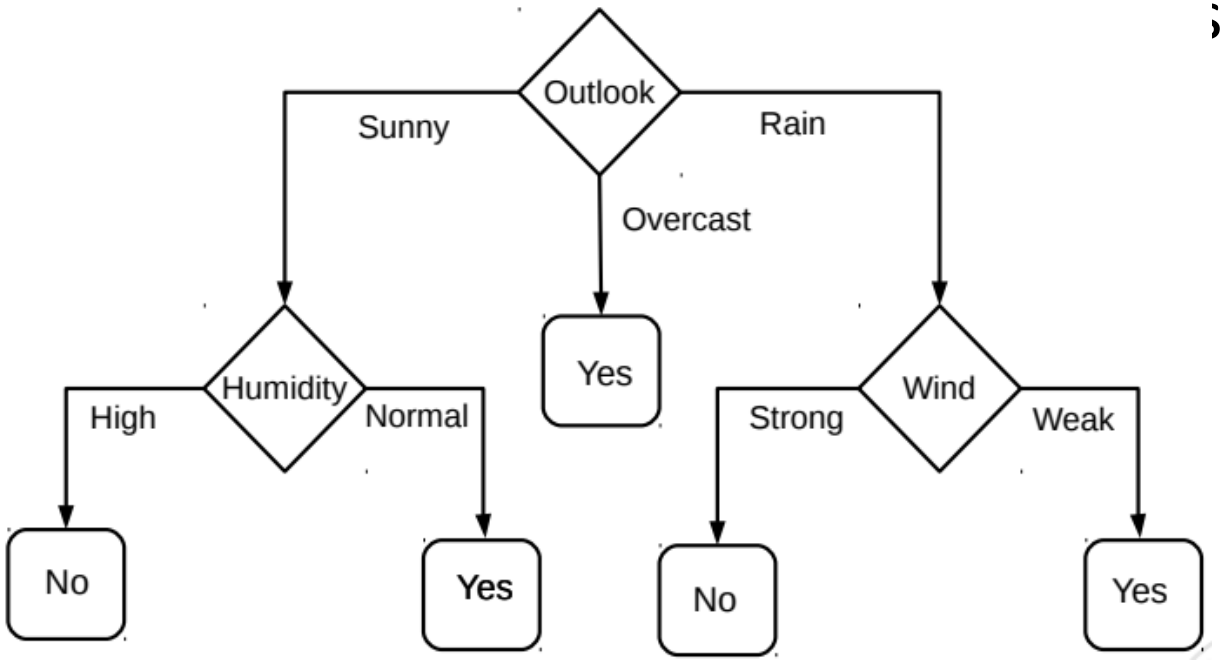
DT is very efficient at test time: To predict the label of a test point, nearest neighbors will require computing distances from 48 training inputs. DT predicts the label by doing just 2 feature-value comparisons! Way faster!

Remember: Root node contains all training inputs
Each leaf node receives a subset of training inputs

Decision Trees for Classification: Another Example

- Deciding whether to play or not to play Tennis on a Saturday
- Each input (Saturday) has 4 categorical features: Outlook, Temp., Humidity, Wind
- A binary classification problem (play vs no-play)
- Below Left: Training data, Below Rig[illegible] data

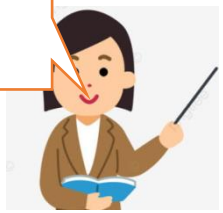
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



Example credit: Tom Mitchell

Decision Trees: Some Considerations

Usually, cross-validation can be used to decide size/shape



- What should be the size/shape of the DT?
  - Number of internal and leaf nodes
  - Branching factor of internal nodes
  - Depth of the tree

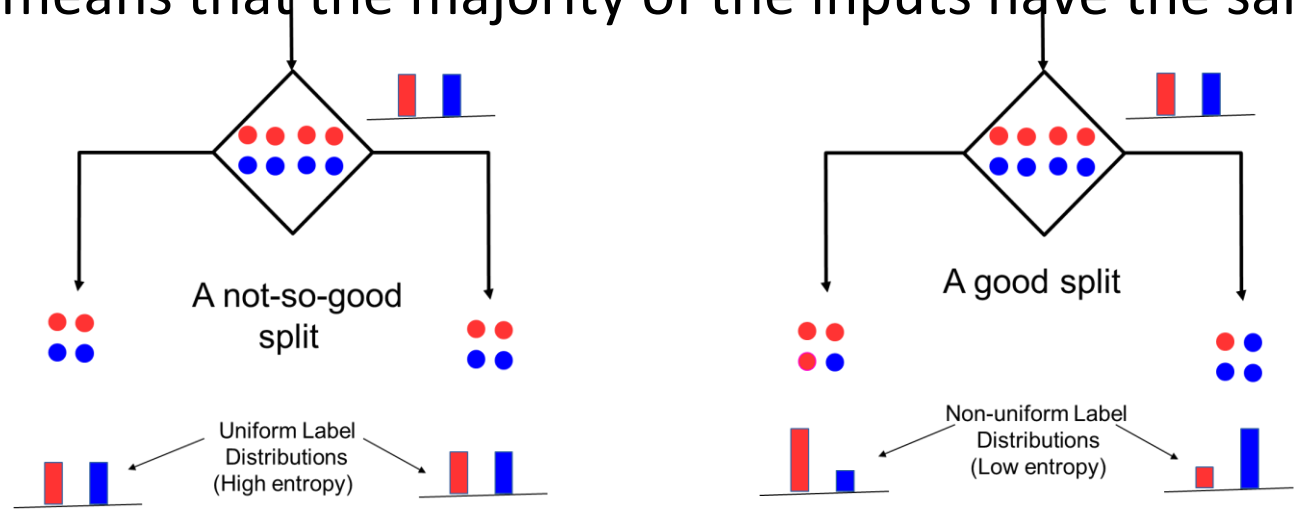
- Split criterion at internal nodes
  - Use another classifier?
  - Or maybe by doing a simpler test?

Usually, constant prediction at leaf nodes used since it will be very fast

- What to do at the leaf node? Some options:
  - Make a constant prediction for each test input reaching there
  - Use a nearest neighbor based prediction using training inputs at that leaf node
  - Train and predict using some other sophisticated supervised learner on that node

How to Split at Internal Nodes?

- Recall that each internal node receives a subset of all the training inputs
- Regardless of the criterion, the split should result in as "pure" groups as possible
  - A pure group means that the majority of the inputs have the same label/output

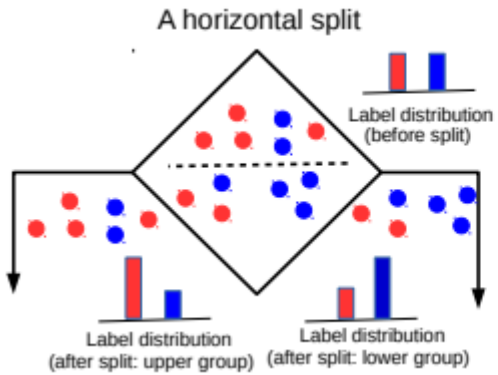


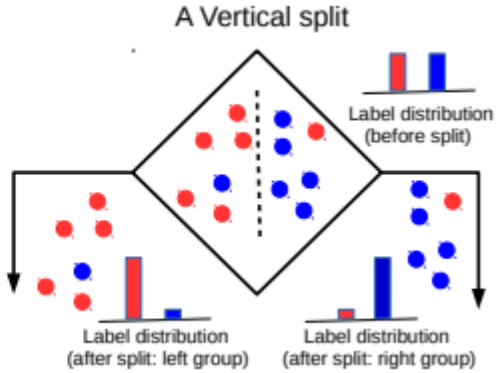
- For classification problems (discrete outputs), entropy is a measure of purity
  - Low entropy ⇒ high purity (less uniform label distribution)
  - Splits that give the largest reduction (before split vs after split) in entropy are preferred (this reduction is also known as "information gain")

Techniques to Split at Internal Nodes

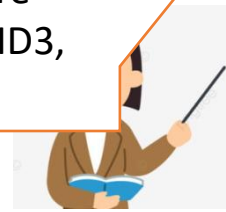
- Each internal node decides which outgoing branch an input should be sent to
- This decision/split can be done using various ways, e.g.,
  - Testing the value of a single feature at a time (such internal node called "Decision Stump")
  - See here for more details

With this approach, all features and all possible values of each feature need to be evaluated in selecting the feature to be tested at each internal node
(can be slow but can be made faster using some tricks)

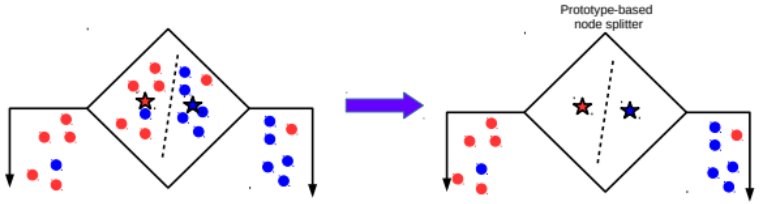




DT methods based on testing a single feature at each internal node are faster and more popular (e.g., ID3, C4.5 algos)

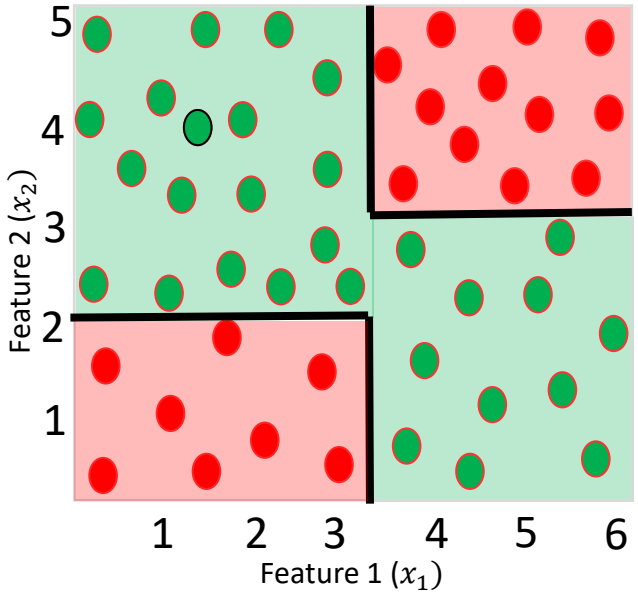


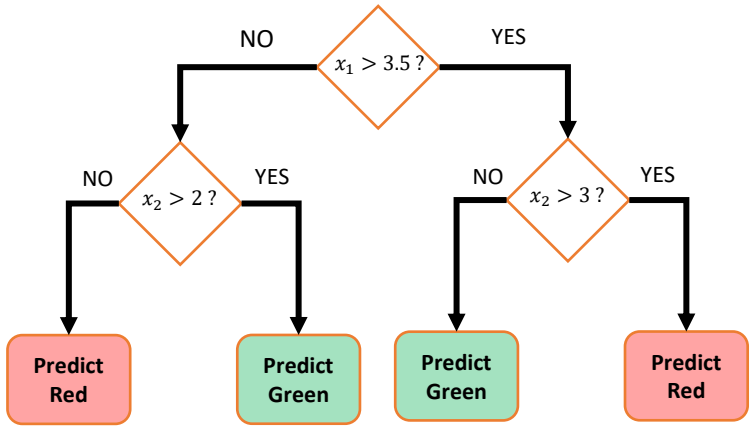
- Learning a classifier (e.g., LwP or some more sophisticated classifier)



DT methods based on learning and using a separate classifier at each internal node are less common. But this approach can be very powerful and are sometimes used in some advanced DT methods

Constructing Decision Trees





Given some training data, what's the "optimal" DT?

How to decide which rules to test for and in what order?

How to assess informativeness of a rule?

In general, constructing DT is an intractable problem (NP-hard)

Often we can use some "greedy" heuristics to construct a "good" DT

To do so, we use the training data to figure out which rules should be tested at each node

The same rules will be applied on the test inputs to route them along the tree until they reach some leaf node where the prediction is made

The rules are organized in the DT such that most informative rules are tested first

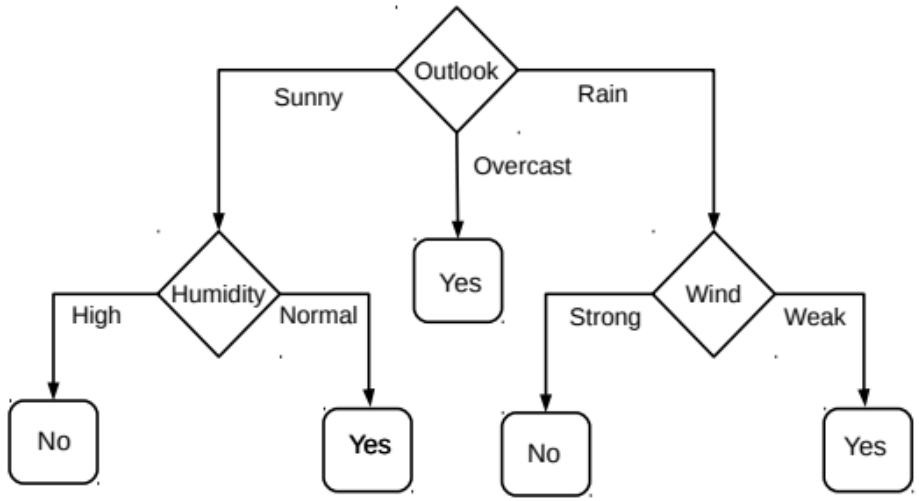
Informativeness of a rule is of related to the extent of the purity of the split arising due to that rule. More informative rules yield more pure splits

Hmm.. So DTs are like the "20 questions" game (ask the most useful questions first)

Decision Tree Construction: An Example

- Let's consider the playing Tennis example
- Assume each internal node will test the value of one of the features

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Question: Why does it make more sense to test the feature "outlook" first?
- Answer: Of all the 4 features, it's the most informative
  - It has the highest information gain as the root node

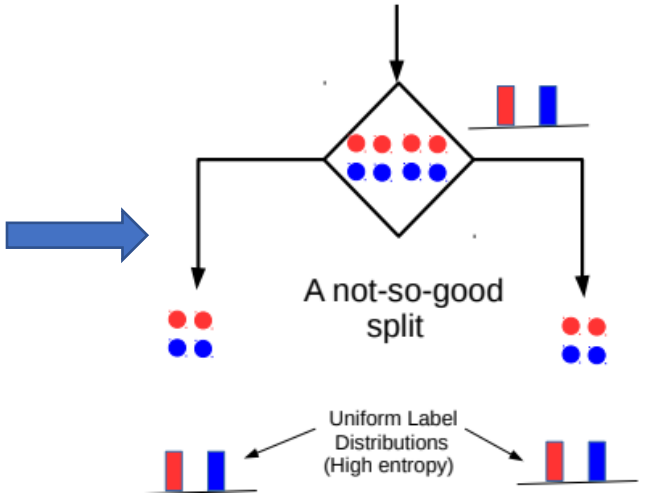
Entropy and Information Gain

- Assume a set of labelled inputs $\boldsymbol{S}$ **from** $\boldsymbol{C}$ classes, p_c as fraction of class c inputs
- **Entropy** of the set $\boldsymbol{S}$ is defined as $\mathrm{H}(\boldsymbol{S}) = -\sum_{c \in C} p_c \log p_c$
- Suppose a rule splits $\boldsymbol{S}$ into two smaller disjoint sets $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$
- Reduction in entropy after the split is called information gain

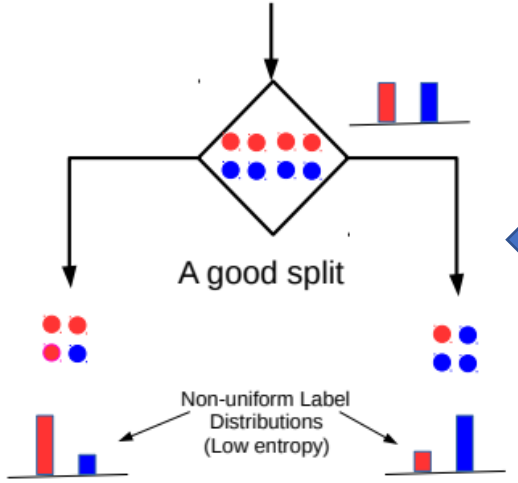
$$IG = H(S) - \frac{|S_1|}{|S|} H(S_1) - \frac{|S_2|}{|S|} H(S_2)$$

Uniform sets (all classes roughly equally present) have high entropy; skewed sets low

This split has a low IG (in fact zero IG)



VS



This split has higher IG

Entropy and Information Gain

- Let's use IG based criterion to construct a DT for the Tennis example
- At root node, let's compute IG of each of the 4 features
- Consider feature "wind". Root contains all examples $\boldsymbol{S}$ = [9+,5-]

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

$$H(S) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.94$$

$$\boldsymbol{S}_{\text{weak}} = [6+, 2-] \Rightarrow \text{H}(\boldsymbol{S}_{\text{weak}}) = 0.811$$

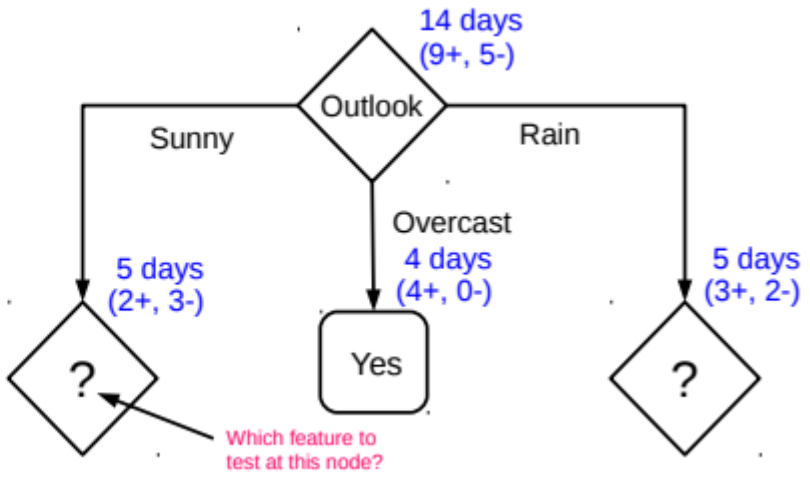
$$\boldsymbol{S}_{\text{strong}} = [3+, 3-] \Rightarrow \text{H}(\boldsymbol{S}_{\text{strong}}) = 1$$

$$IG(S, wind) = H(S) - \frac{|S_{\text{weak}}|}{|S|} H(S_{\text{weak}}) - \frac{|S_{\text{strong}}|}{|S|} H(S_{\text{strong}}) = 0.94 - 8/14 * 0.811 - 6/14 * 1 = 0.048$$

- Likewise, at root: IG(S, outlook) = 0.246, IG(S, humidity) = 0.151, IG(S,temp) = 0.029
- Thus we choose "outlook" feature to be tested at the root node
- Now how to grow the DT, i.e., what to do at the next level? Which feature to test next?
- Rule: Iterate - for each child node, select the feature with the highest IG

Growing the tree

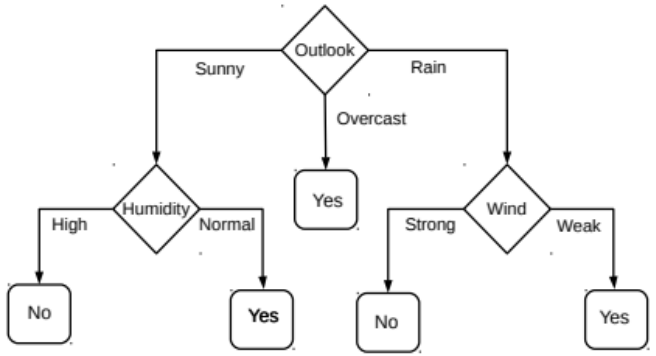
day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



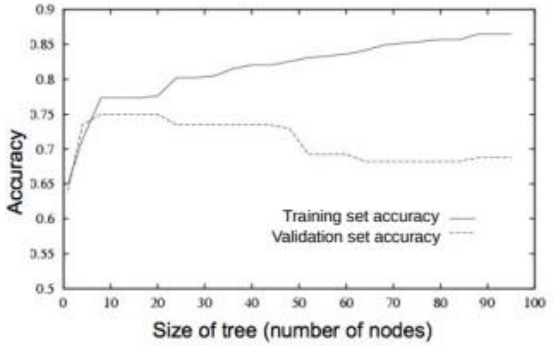
- Proceeding as before, for level 2, left node, we can verify that
  - IG(S,temp) = 0.570, IG(S, humidity) = 0.970, IG(S, wind) = 0.019
- Thus humidity chosen as the feature to be tested at level 2, left node
- No need to expand the middle node (already “pure” - all “yes” training examples)
- Can also verify that wind has the largest IG for the right node
- Note: If a feature has already been tested along a path earlier, we don’t consider it again

When to stop growing the tree?

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Stop expanding a node further (i.e., make it a leaf node) when
  - It consist of all training examples having the same label (the node becomes "pure")
  - We run out of features to test along the path to that node
  - The DT starts to overfit (can be checked by monitoring the validation set accuracy)



To help prevent the tree from growing too much!

- **Important:** No need to obsess too much for purity
  - It is okay to have a leaf node that is not fully pure, e.g., this

OR

  - At test inputs that reach an impure leaf, can predict probability of belonging to each class (in above example, p(red) = 3/8, p(green) = 5/8), or simply predict the majority label

Avoiding Overfitting in DTs

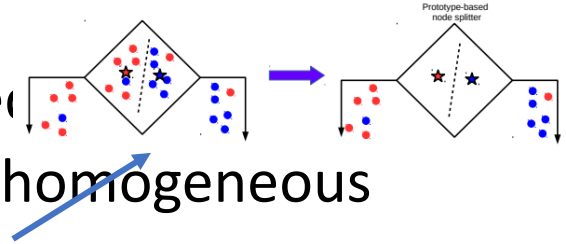
- Desired: a DT that is not too big in size, yet fits the training data reasonably
- Note: An example of a very simple DT is "decision-stump"
  - A decision-stump only tests the value of a single feature (or a simple rule)
  - Not very powerful in itself but often used in large ensembles of decision stumps
- Mainly two approaches to prune a complex DT
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)

Either can be done using a validation set

- Criteria for judging which nodes could potentially be pruned
  - Use a validation set (separate from the training set)
    - Prune each possible node that doesn't hurt the accuracy on the validation set
    - Greedily remove the node that improves the validation accuracy the most
    - Stop when the validation set accuracy starts worsening
  - Use model complexity control, such as Minimum Description Length (will see later)

Decision Trees: Some Comments

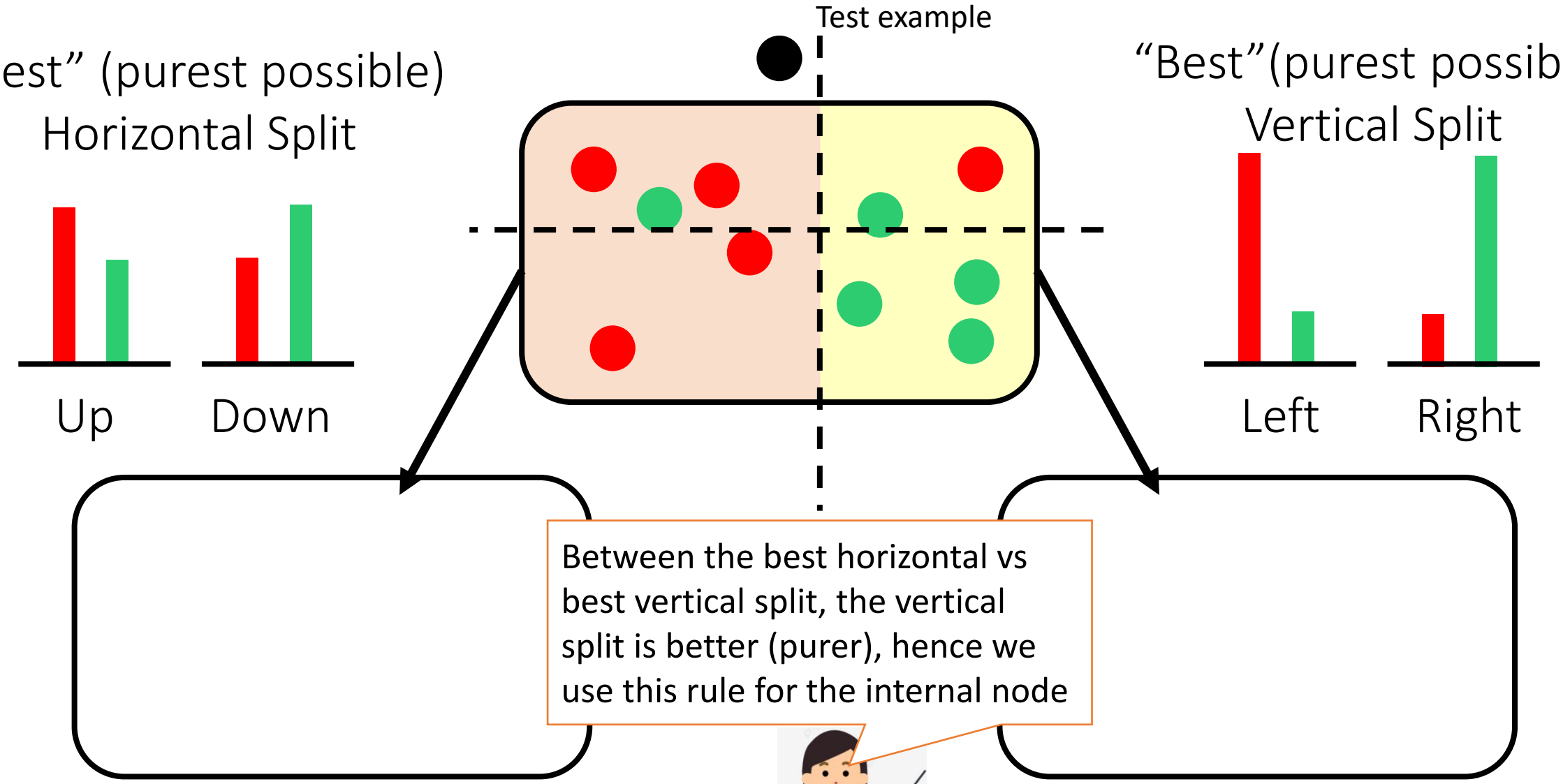
- Gini-index defined as $\sum_{c=1}^{C} p_c(1 - p_c)$ can be an alternative to IG
- For DT regression[1], variance in the outputs can be used to assess purity
- When features are real-valued (no finite possible values to try), things are a bit more tricky
  - Can use tests based on thresholding feature values (recall our synthetic data examples)
  - Need to be careful w.r.t. number of threshold points, how fine each range is, etc.
- More sophisticated decision rules at the internal nodes can also be use



  - Basically, need some rule that splits inputs at an internal node into homogeneous groups
  - The rule can even be a machine learning classification algo (e.g., LwP or a deep learner)
  - However, in DTs, we want the tests to be fast so single feature based rules are preferred
- Need to take care handling training or test inputs that have some features missing

[1]Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees

An Illustration: DT with Real-Valued Features

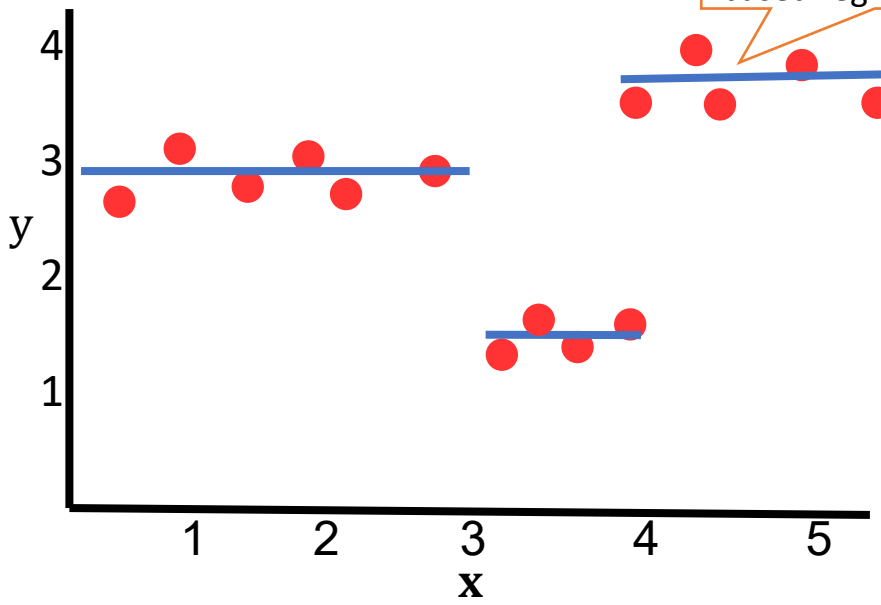


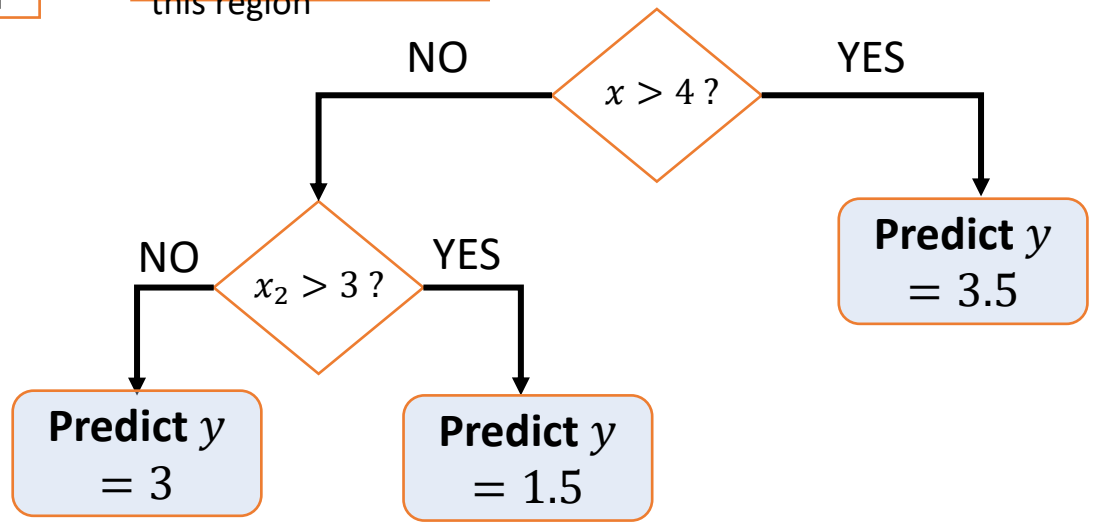
This illustration's credit: Purushottam Kar

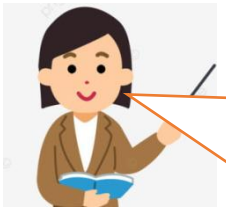
Decision Trees for Regression

Can use any regression model but would like a simple one, so let's use a constant prediction based regression model

Another simple option can be to predict the average output of the training inputs in this region







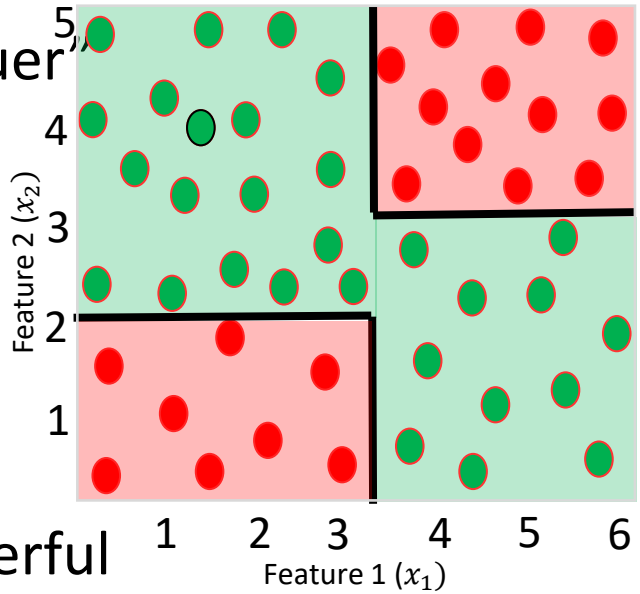
To predict the output for a test point, nearest neighbors will require computing distances from 15 training inputs. DT predicts the label by doing just at most feature-value comparisons! Way faster!

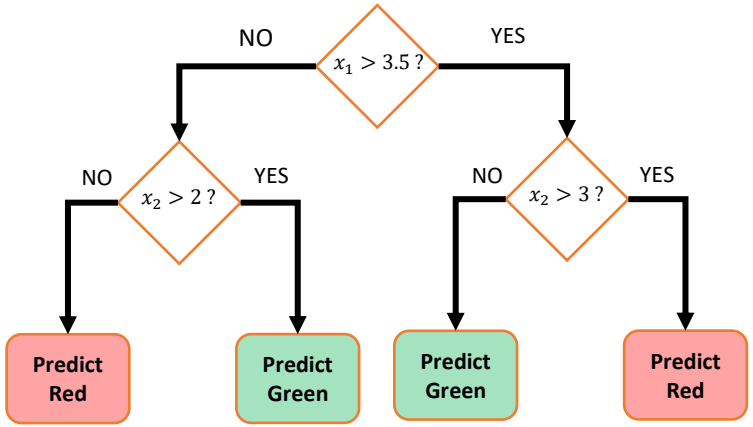
Decision Trees: A Summary

.. thus helping us learn complex rule as a combination of several simpler rules

Some key strengths:

- Simple and easy to interpret
- Nice example of "divide and conquer" paradigm in machine learning
- Easily handle different types of features (real, categorical, etc.)
- Very fast at test time
- Multiple DTs can be combined via ensemble methods: more powerful (e.g., Decision Forests; will see later)





Human-body pose estimation

- Used in several real-world ML applications, e.g., recommender systems, gaming (Kinect)

Some key weaknesses:

- Learning optimal DT is (NP-hard) intractable. Existing algos mostly greedy heuristics
- Can sometimes become very complex unless some pruning is applied

Pruning

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.

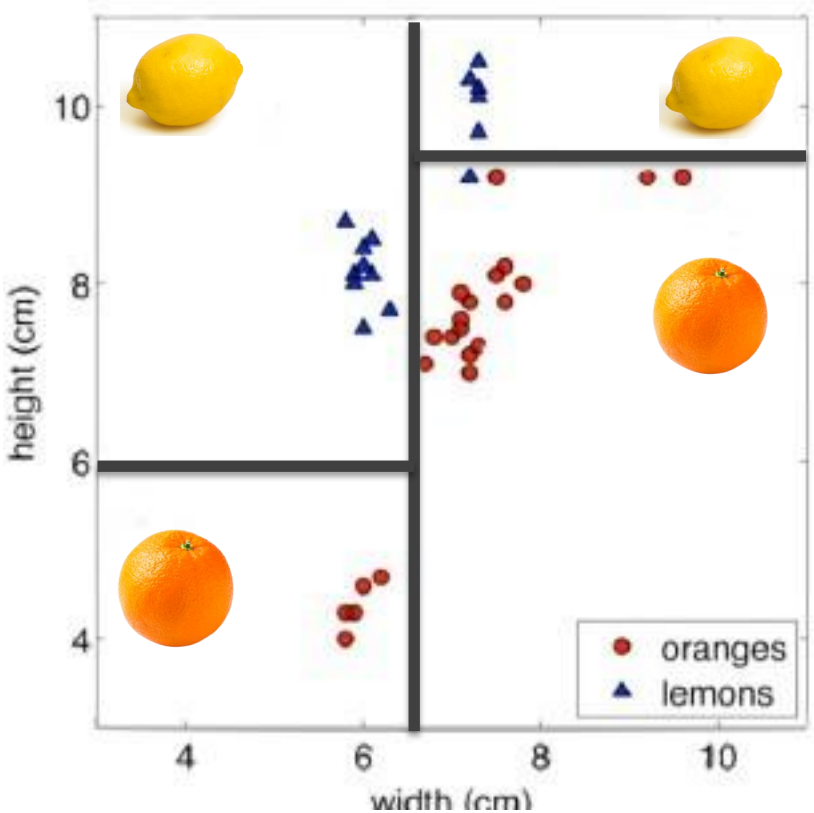
How can we fix this issue?

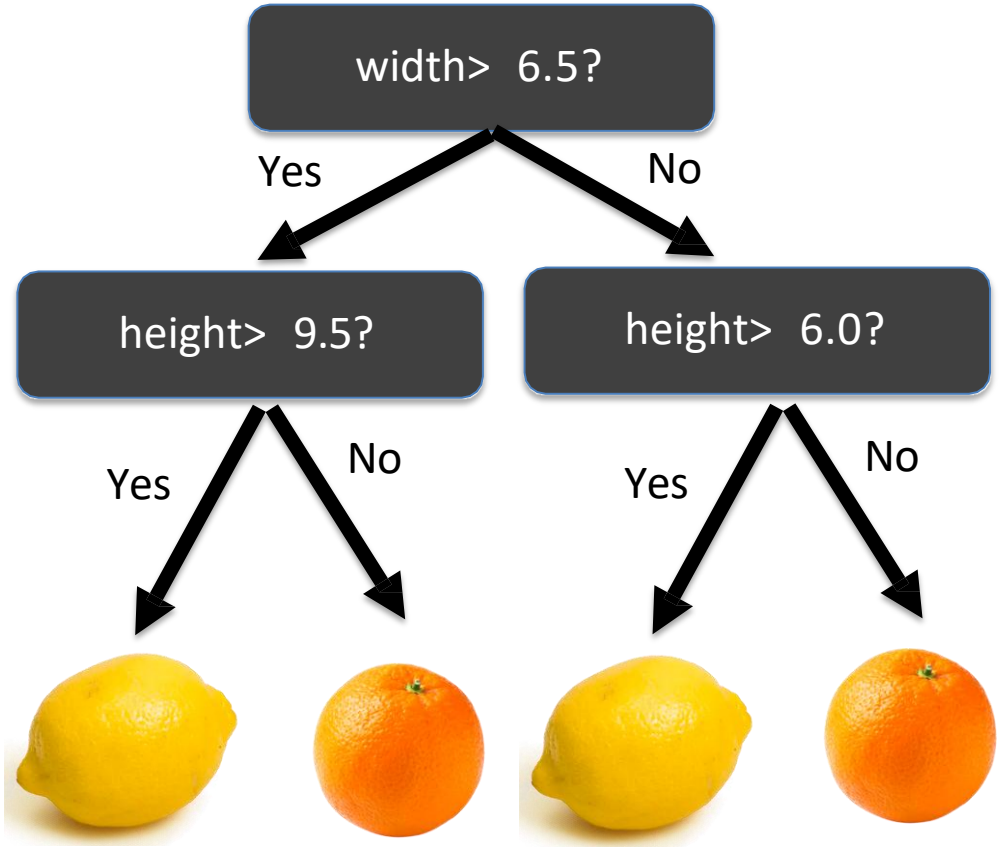
- choose several stopping criterion (set minimal Gain(R) at various levels) and cross-validate which is the best.

What is an alternative approach to this issue?

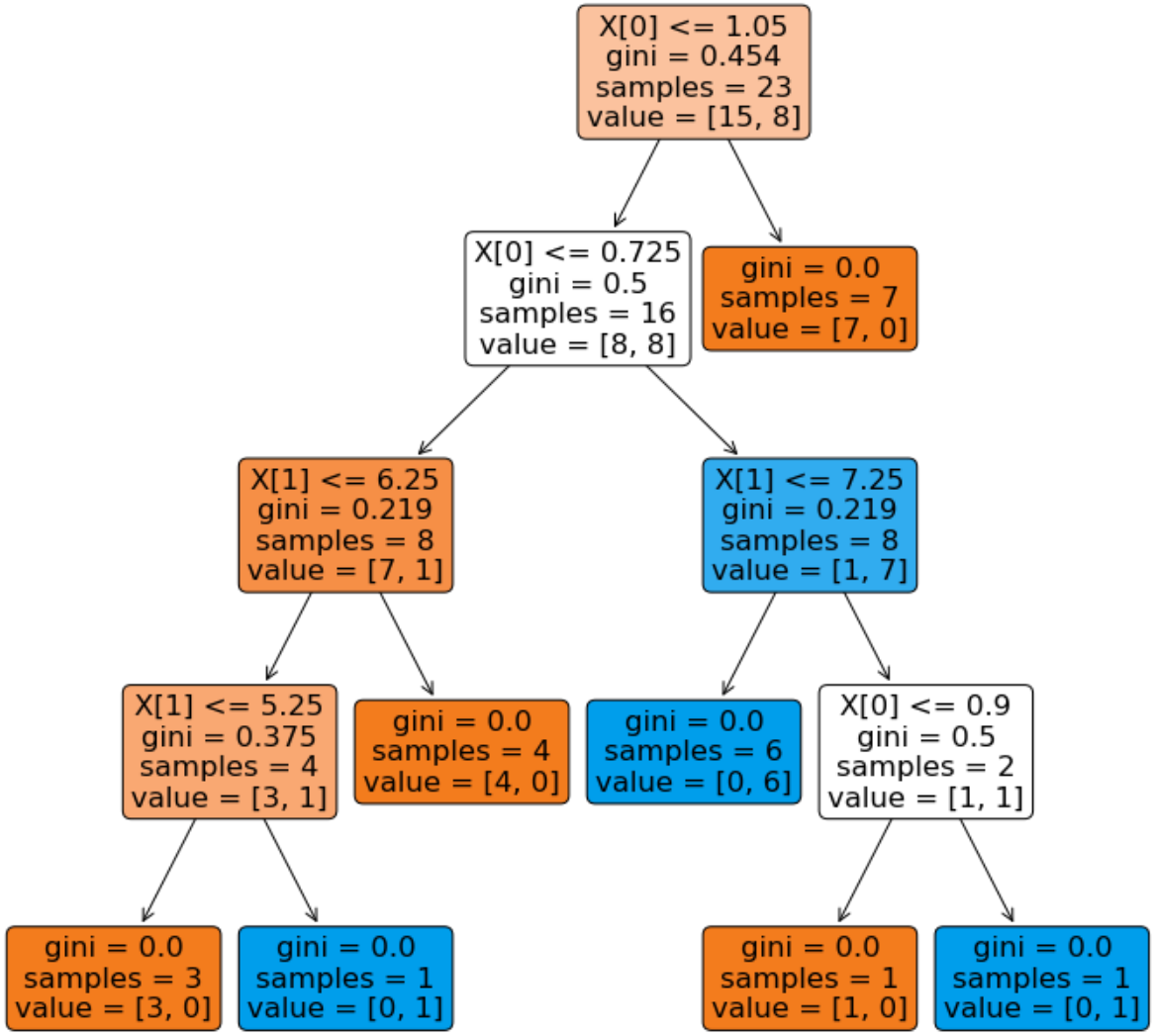
- Don't stop. Instead prune back!

Lemons or Oranges

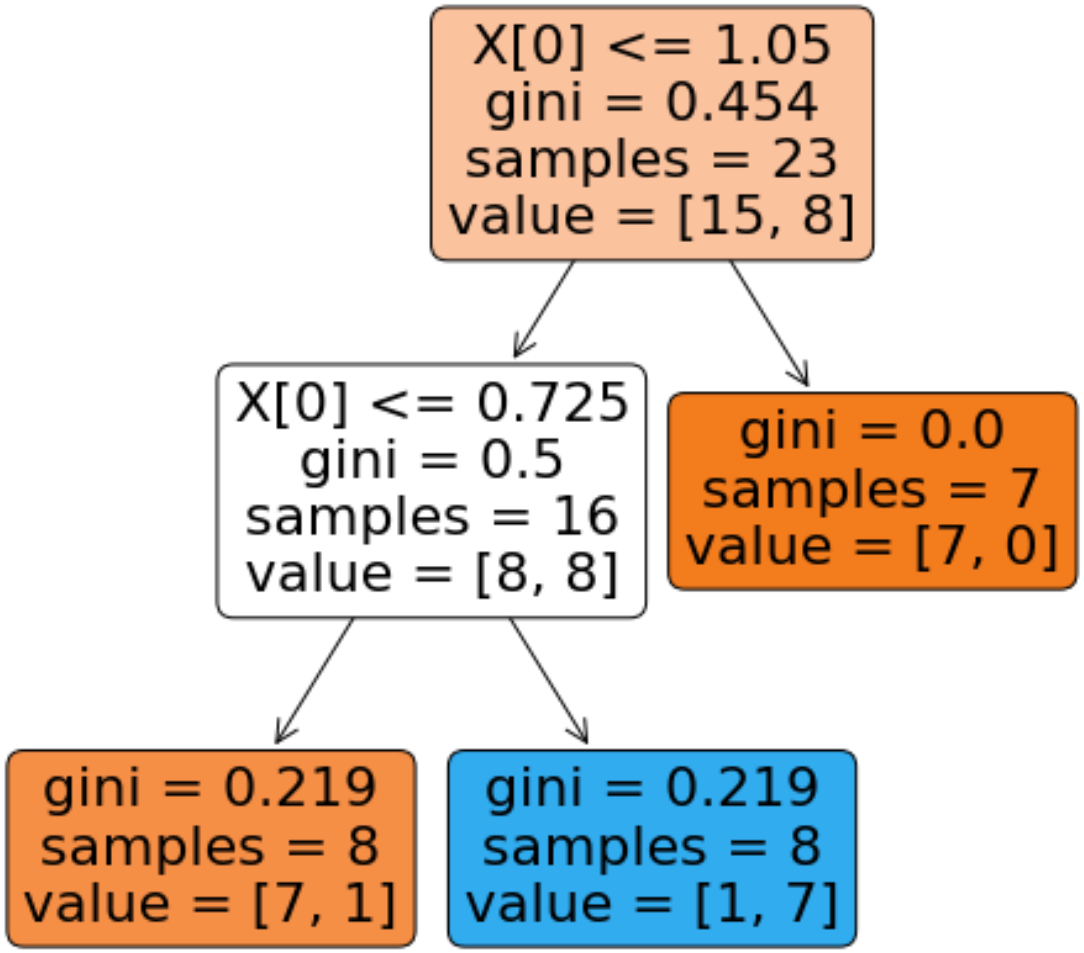




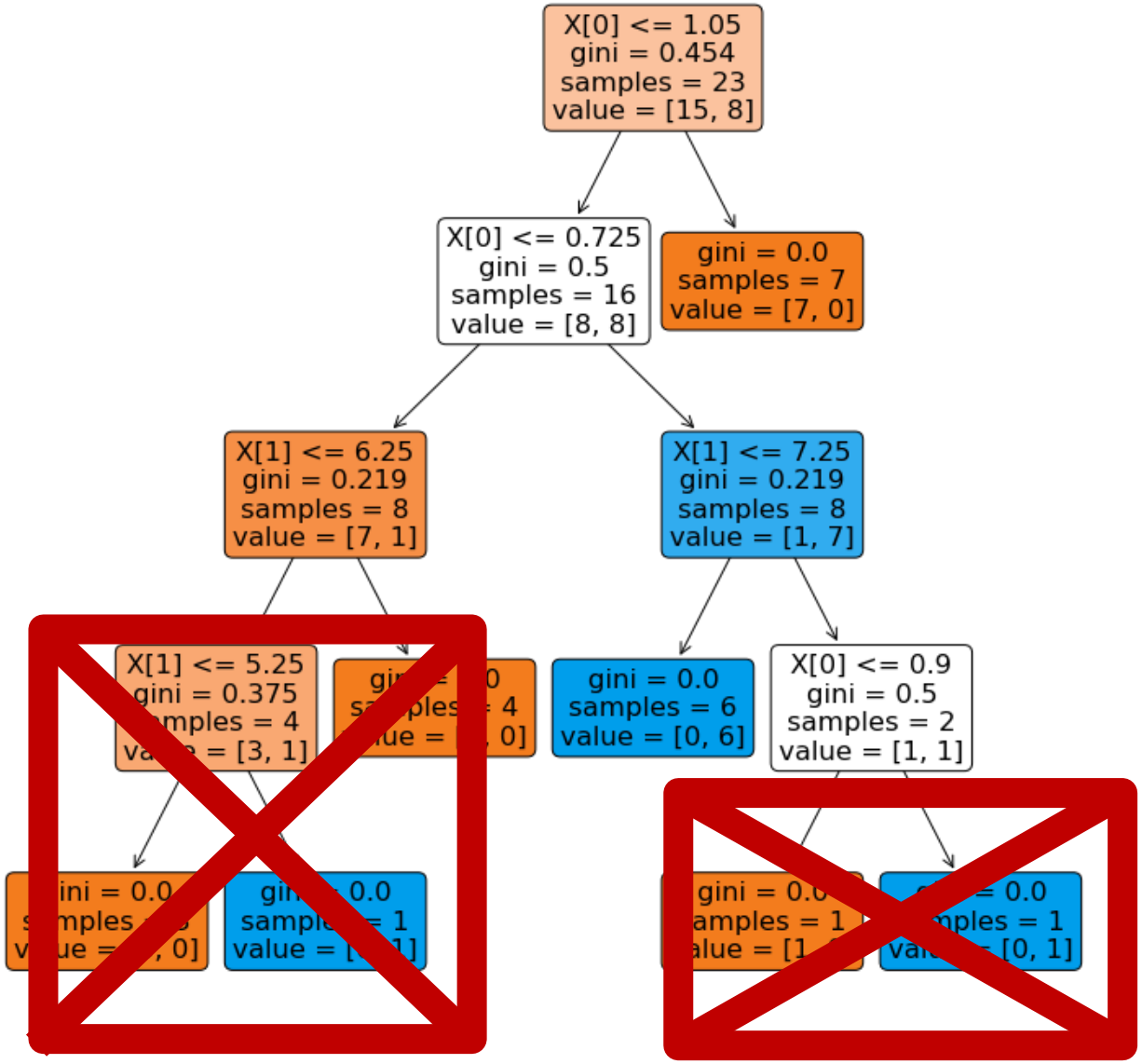
Motivation for Pruning



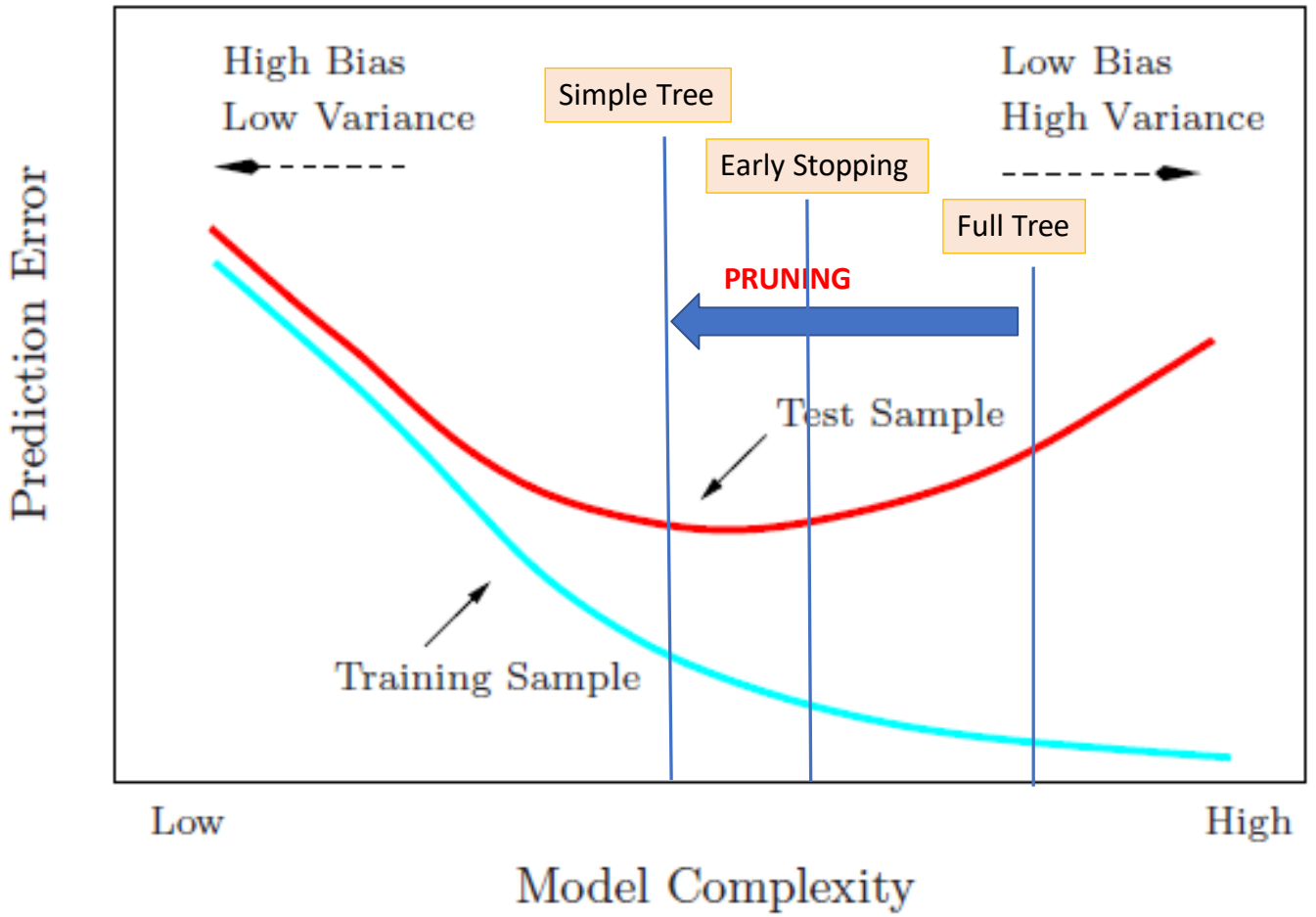
Motivation for Pruning



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.

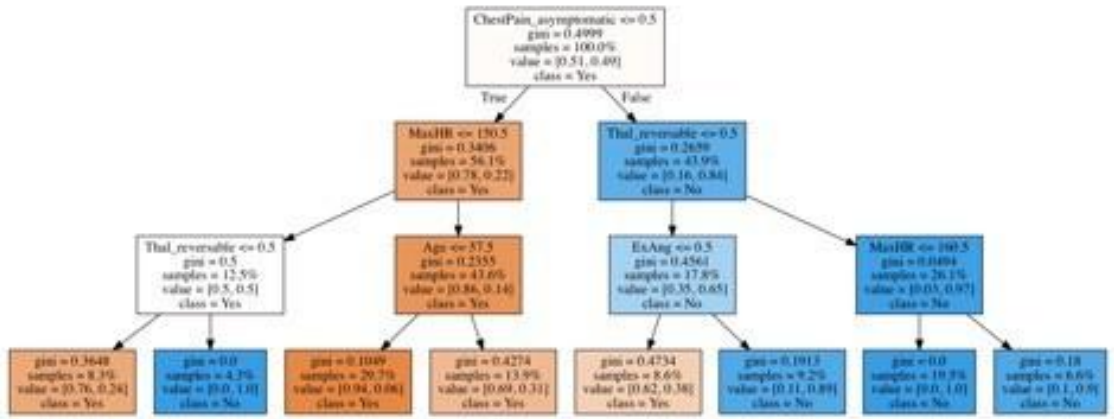
There are many method of pruning, a common one is ***cost complexity pruning***, where by we select from a array of smaller subtrees of the full model that optimizes a balance of performance and efficiency.

That is, we measure

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

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

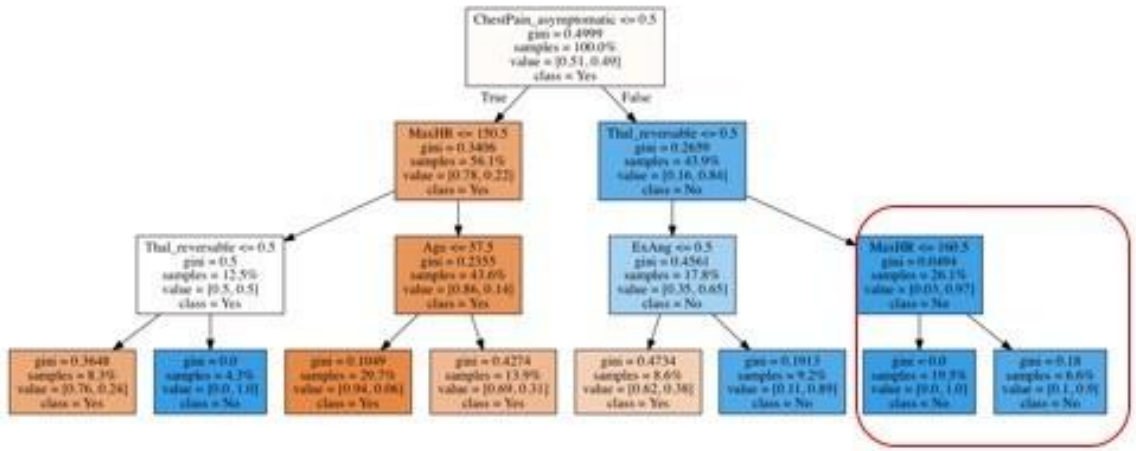
Pruning



$\alpha = 0.2$

Tree	Error	Num Leaves	Total

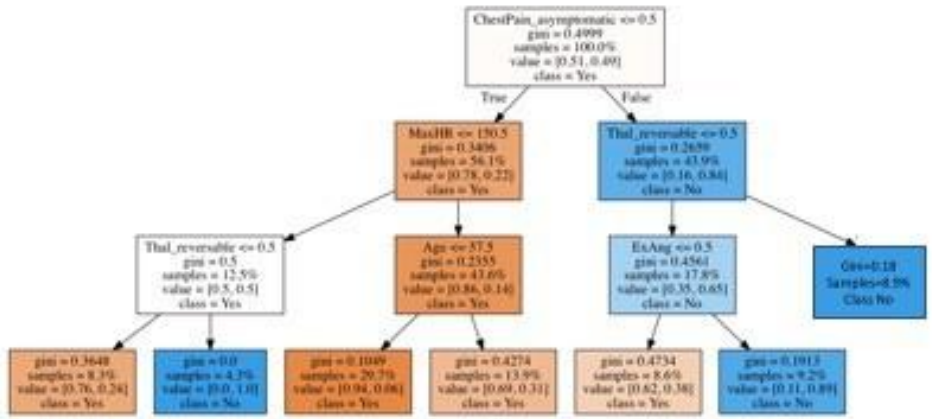
Pruning



$\alpha = 0.2$

Tree	Error	Num Leaves	Total
T	0.32	8	1.92

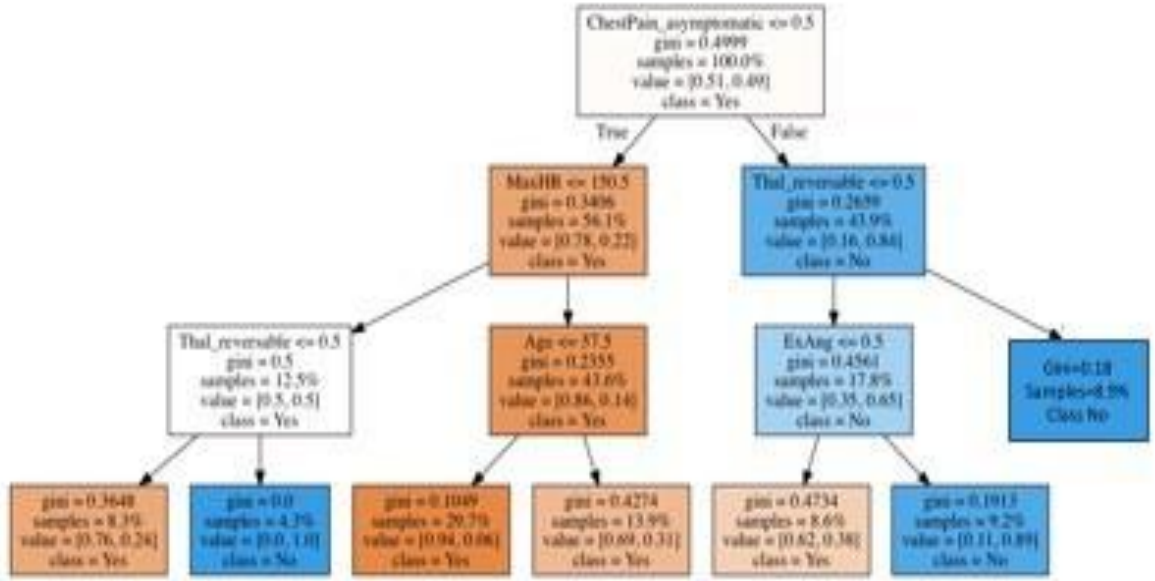
Pruning



$\alpha = 0.2$

Tree	Error	Num Leaves	Total
T	0.32	8	1.92
Tsmall	0.33	7	1.73

Pruning



$\alpha = 0.2$

Tree	Error	Num Leaves	Total
T	0.32	8	1.92
Tsmall	0.33	7	1.73

Smaller tree has larger error but less cost complexity score

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. Find best α using CV (what should be the error measure?)

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

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

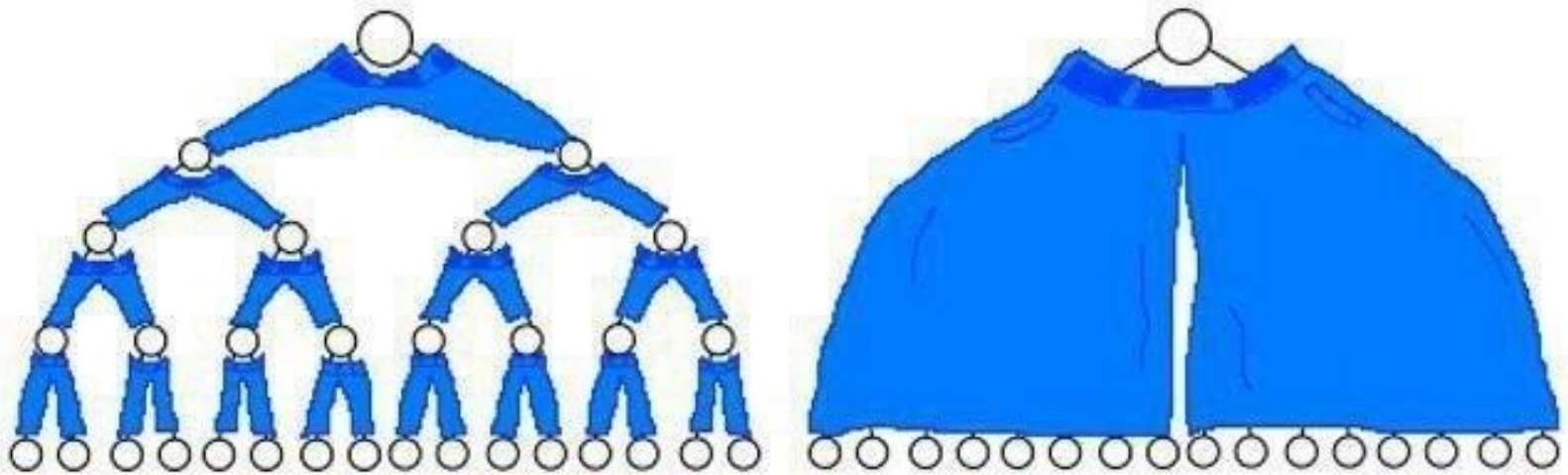
3. Iterate this pruning process to obtain $T_0, T_1, \ldots, T_L$ where T_L is the tree containing just the root of T_0
4. Select the optimal tree T_i by cross validation.

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.

bagging

If a binary tree wore pants would he wear them

like this or like this?



Outline

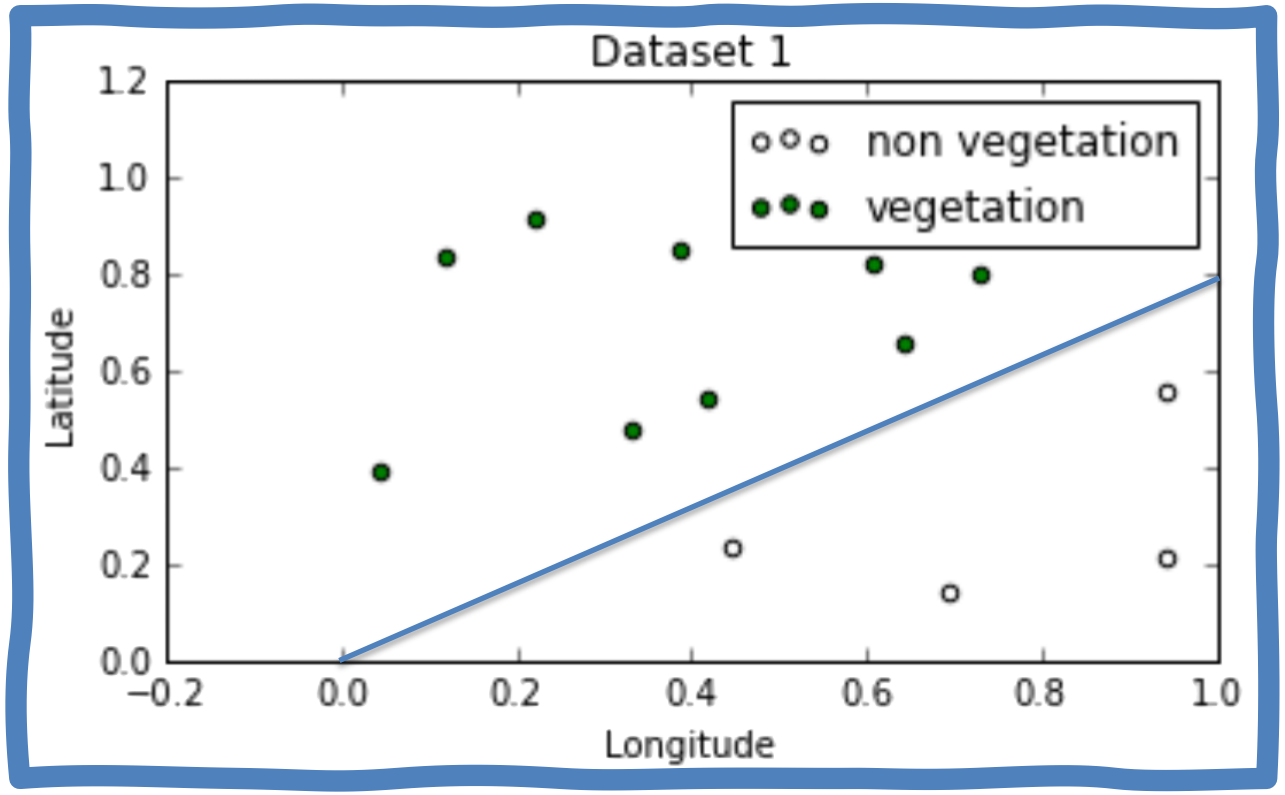
- Review of Decision Trees
- Bagging
- Out of Bag Error (OOB)
- Variable Importance

Outline

- **Review of Decision Trees**
- Bagging
- Out of Bag Error (OOB)
- Variable Importance

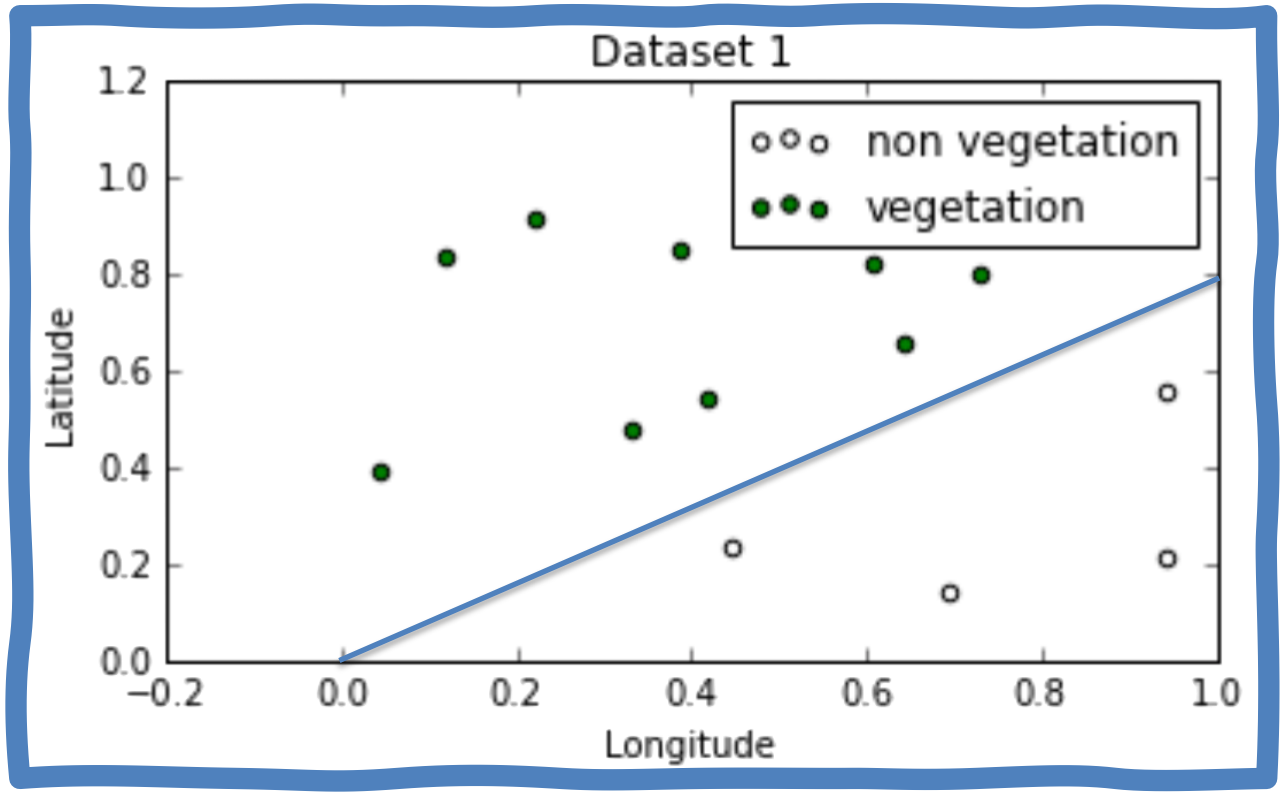
Geometry of Data

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



Geometry of Data

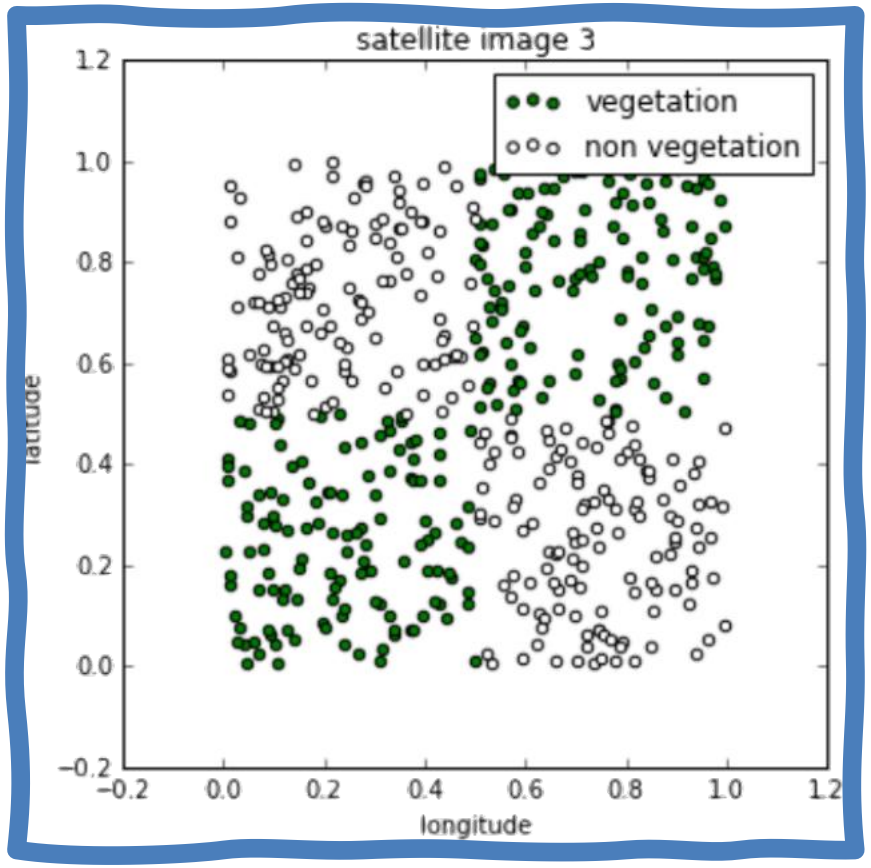
Question: Can you guess the equation that defines the decision boundary below?



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

Geometry of Data

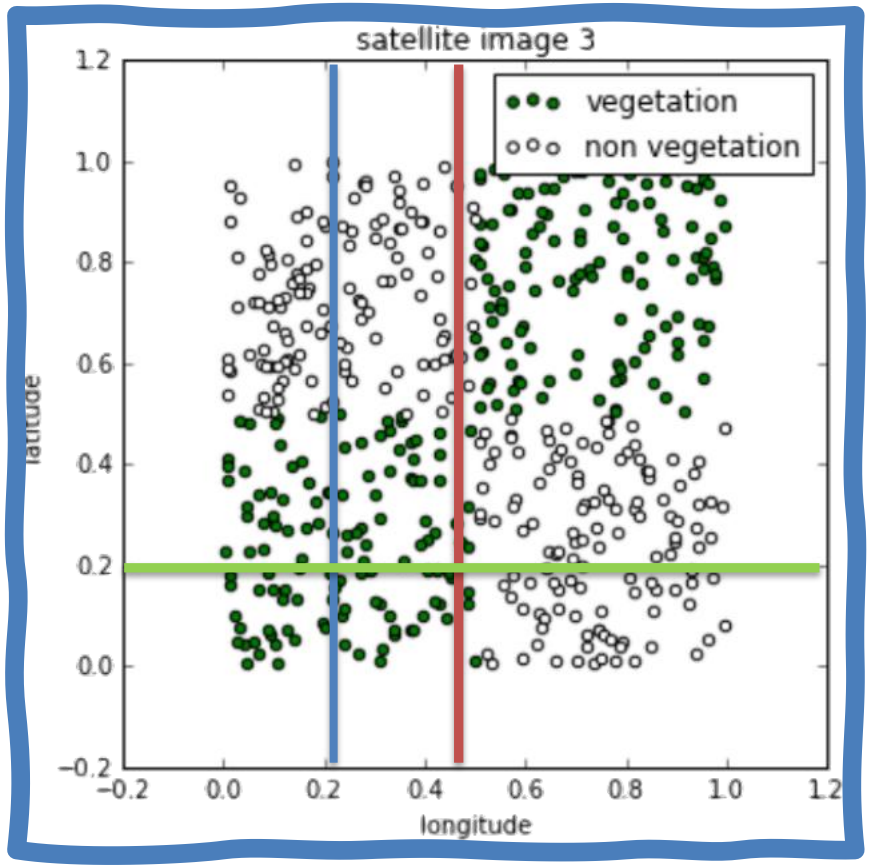
Complicate decision boundaries can not be explained with Log Regression.



Geometry of Data

But can be described using Trees.

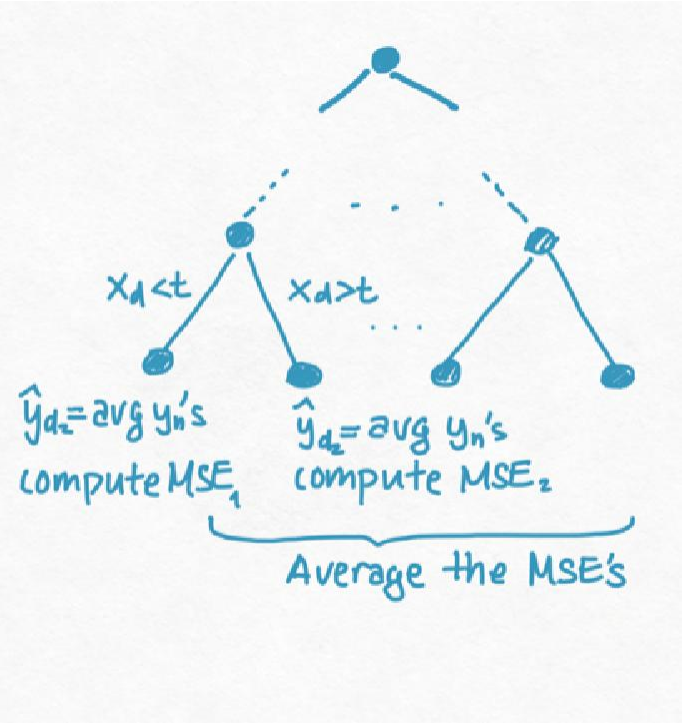
Which one represents the first split of a decision tree?



Decision Trees

To learn a decision tree model, we take a greedy approach:

1. Start with a node containing all the data.
2. If stopping condition is not met:
   A. Choose the 'optimal' predictor and threshold and divide the data in the node into two sets.
3. For each new node, repeat step 2.

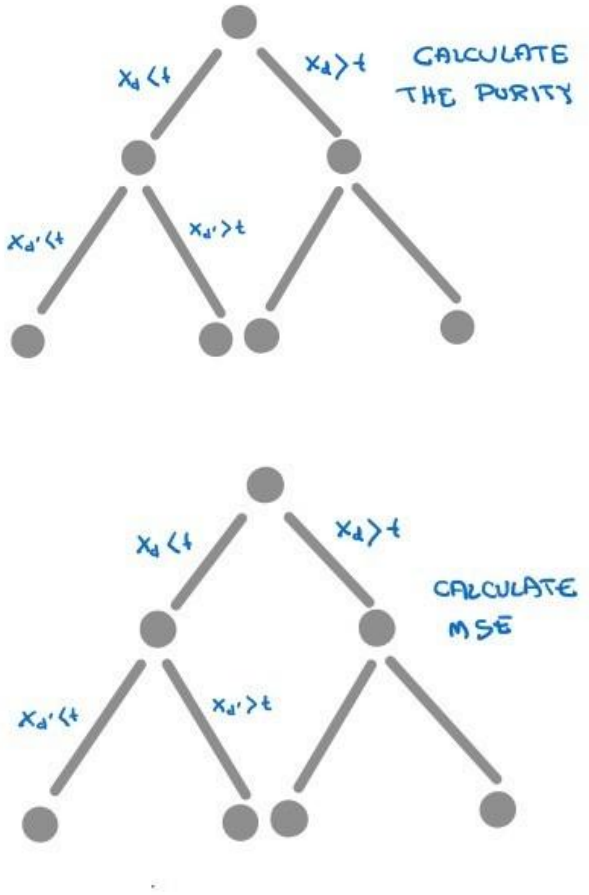


Decision Trees: Splitting Criteria

Splitting Criteria:

For classification, purity of the regions is a good indicator the performance of the model. Entropy as a splitting criterial minimizes the cross-entropy (greedy). Gini is also a splitting criteria.

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

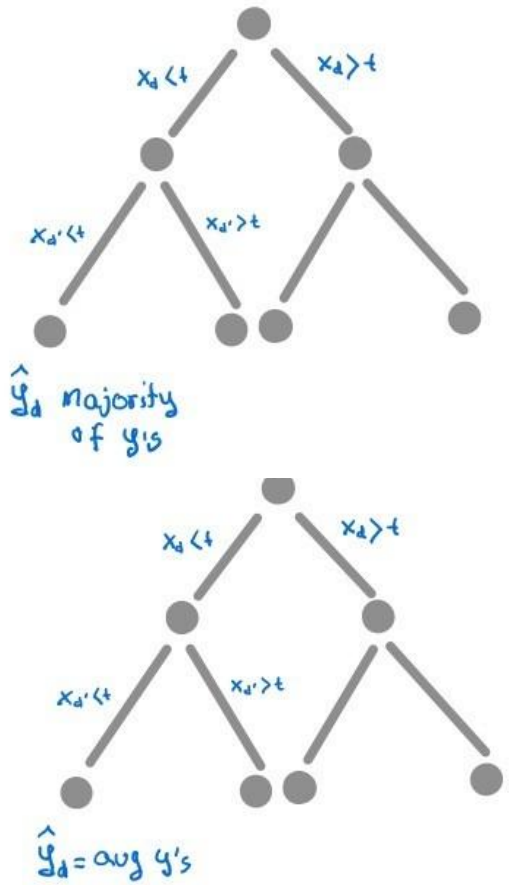


Decision Trees: Prediction

Prediction:

For classification, we label each region in the model with the label of the class to which the **plurality** of the points within the region belong.

For regression, we predict with the **average** of the output values of the training points contained in the region.



Stopping Conditions

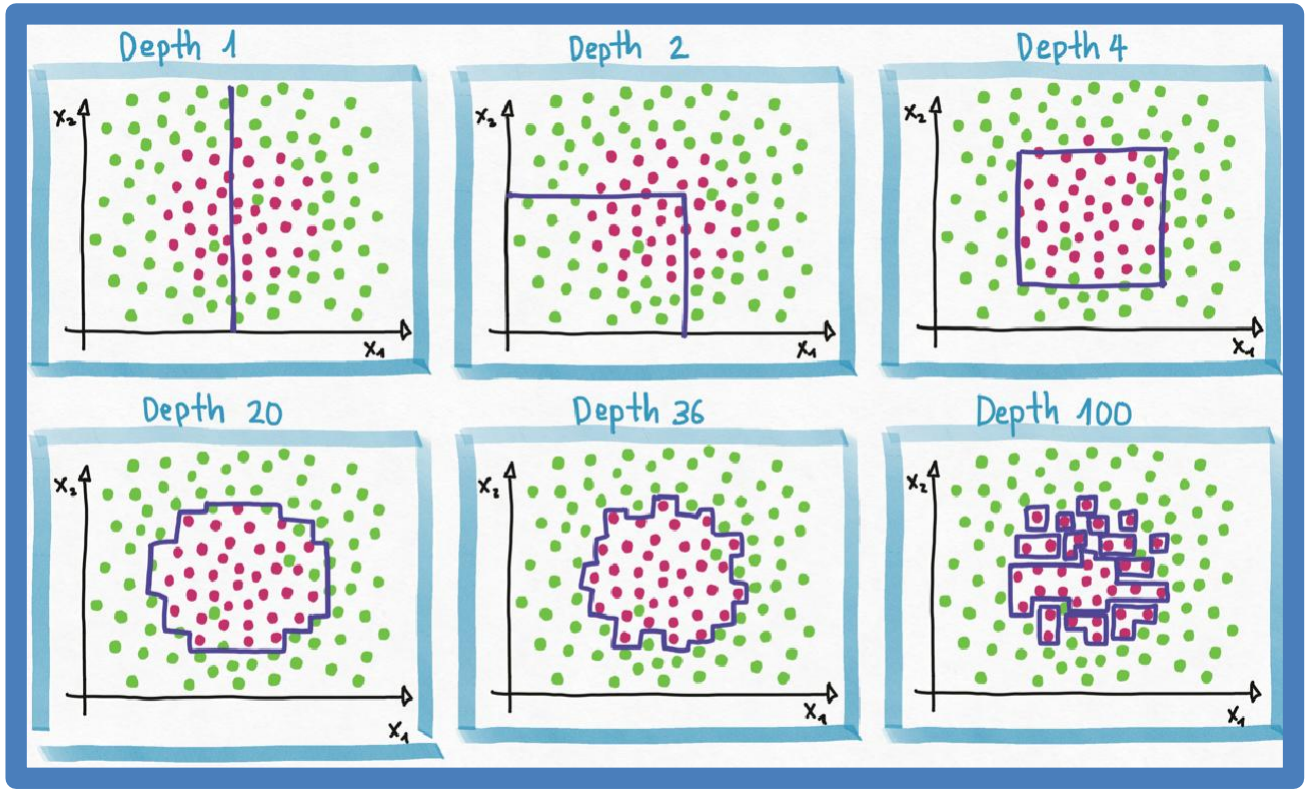
The stopping condition is usually a maximum depth or a minimum MSE.

But others common simple stopping conditions are:

- 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.
- Don't split if the gain in purity, information, reduction in entropy or MSE of splitting a region R into R_1 and R_2 is less than some pre-defined threshold.

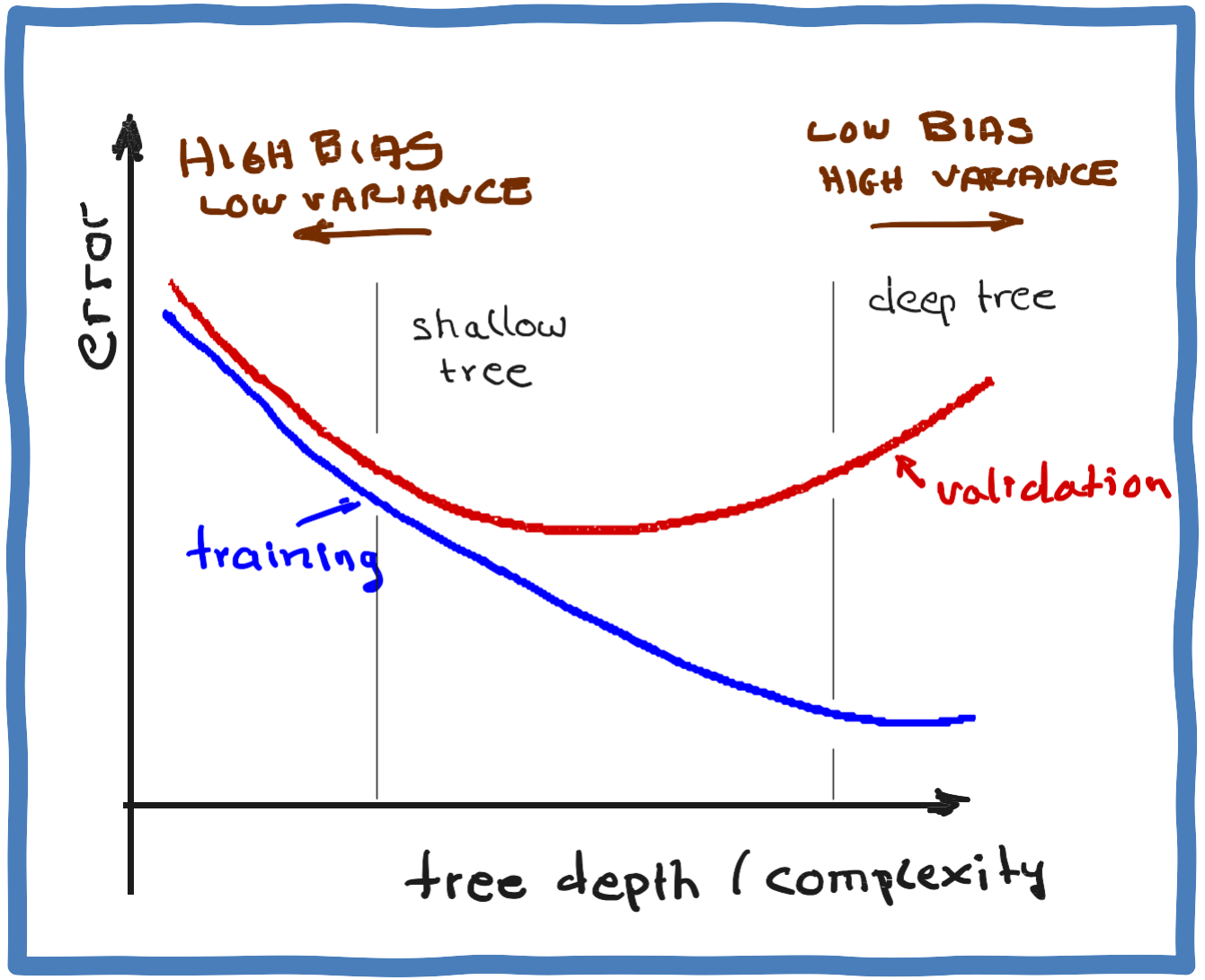
Overfitting

When a tree is too shallow, it cannot divide the input data into enough regions, so the model underfits. When the tree is too deep it cuts the input space into too many regions and fit to the noise of the data -> overfits.

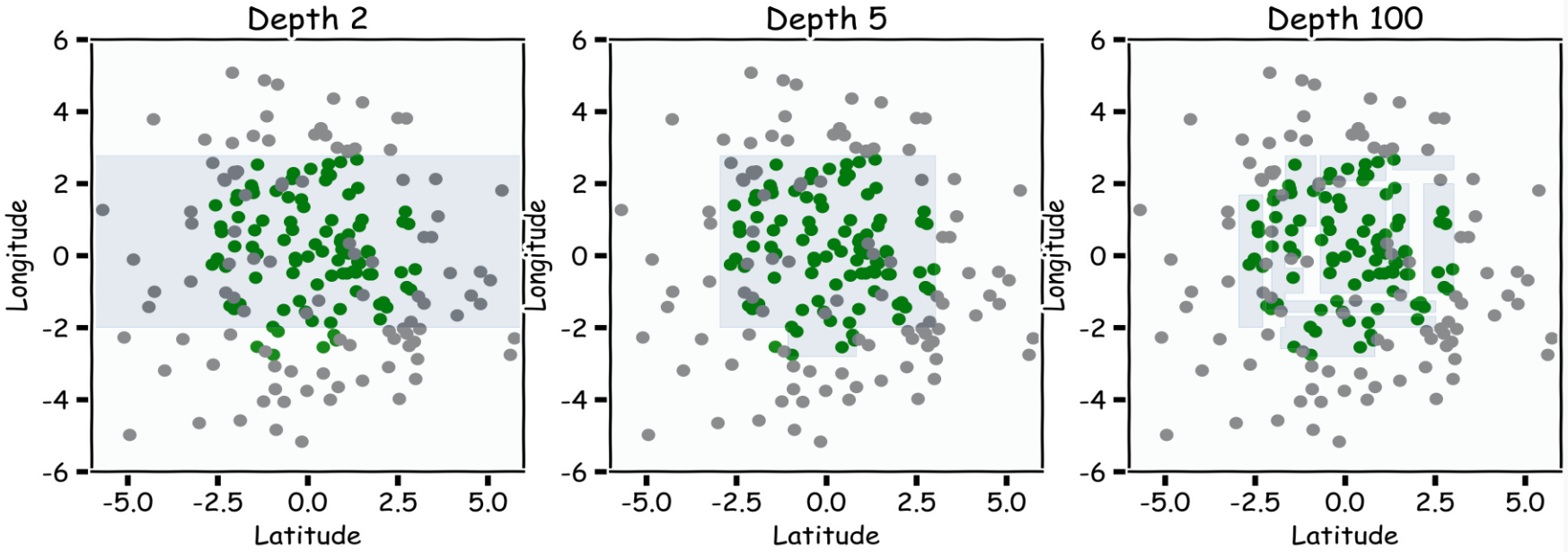


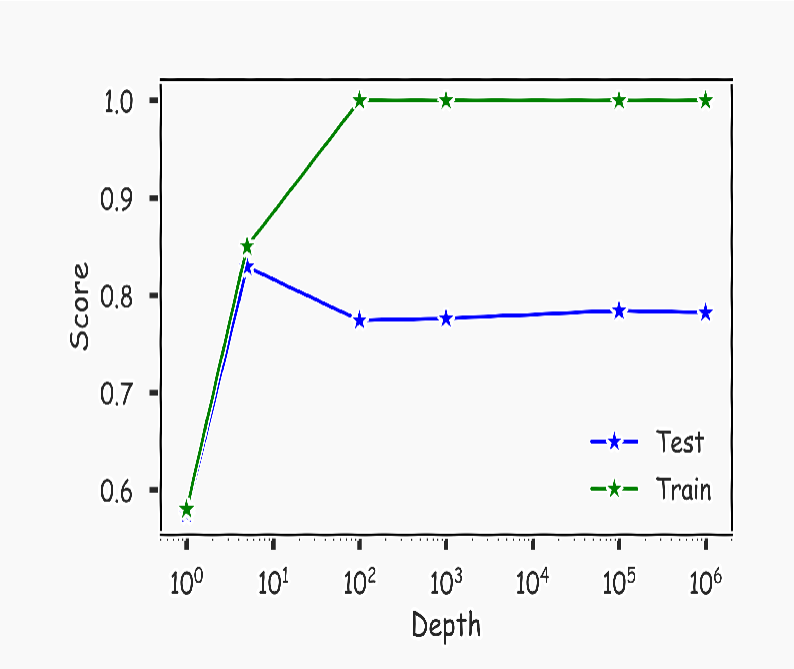
Overfitting

Avoid overfitting by pruning or limiting the depth of the tree and using CV.



Reduce the variance: Depth of the tree





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

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

Limitations of Decision Tree Models

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

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

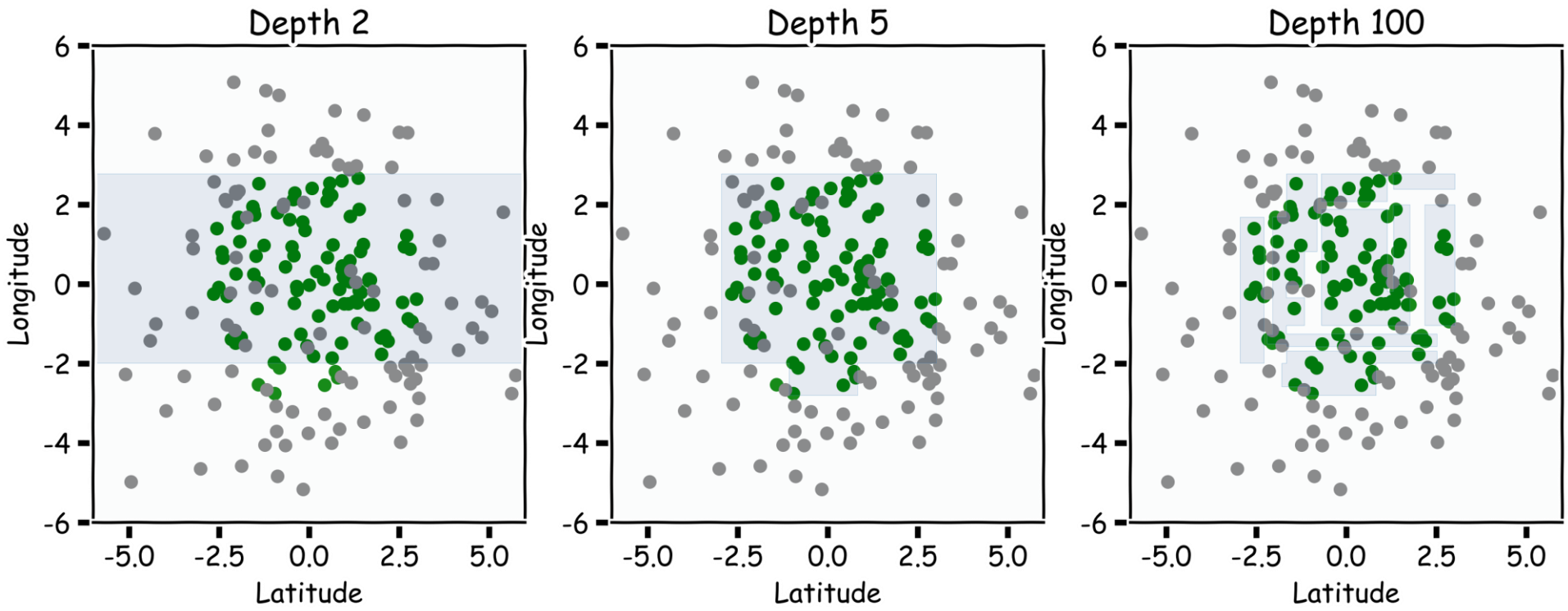
We've seen that large trees have high variance and are prone to overfitting.

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

Outline

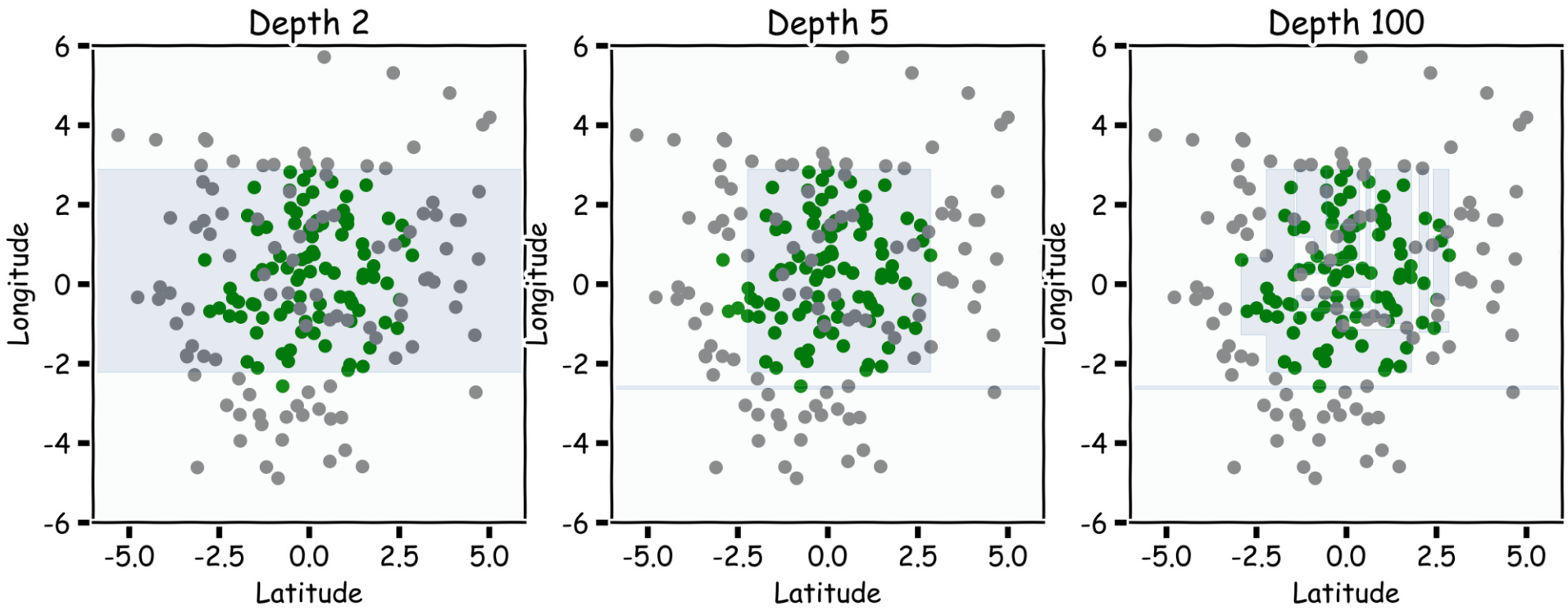
- Review of Decision Trees
- **Bagging**
- Out of Bag Error (OOB)
- Variable Importance

Bagging


Depth 2
Depth 5
Depth 100
Longitude
Latitude
6
4
2
0
-2
-4
-6
-5.0
-2.5
0.0
2.5
5.0

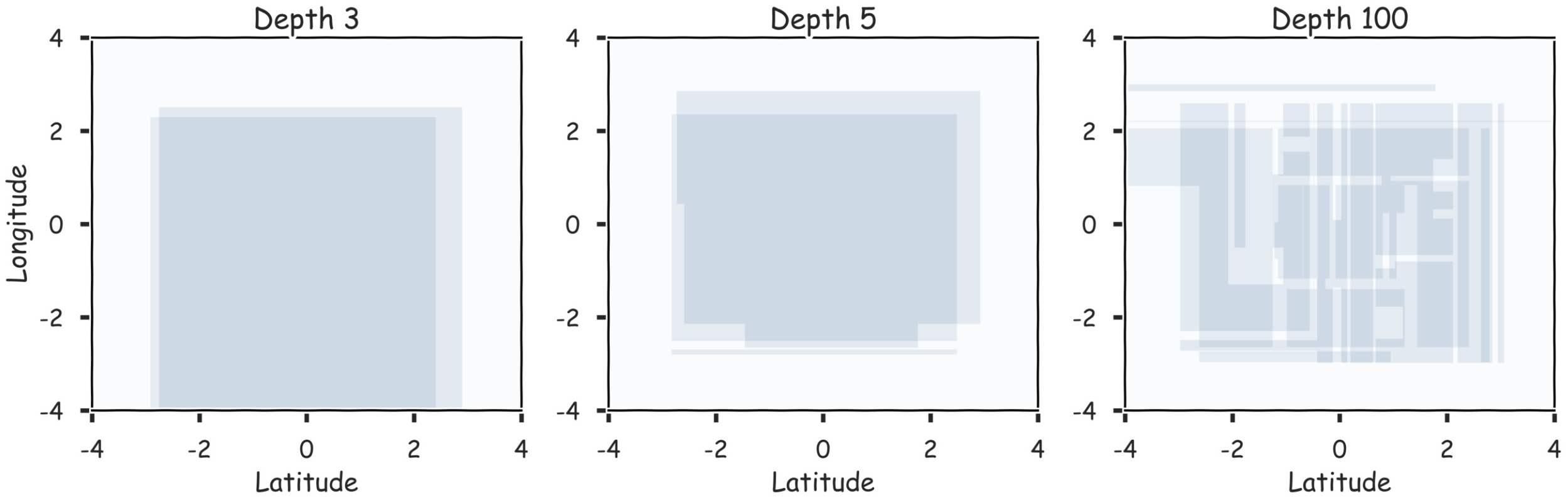
Bagging

My favorite reality: magic realism -> bootstrap

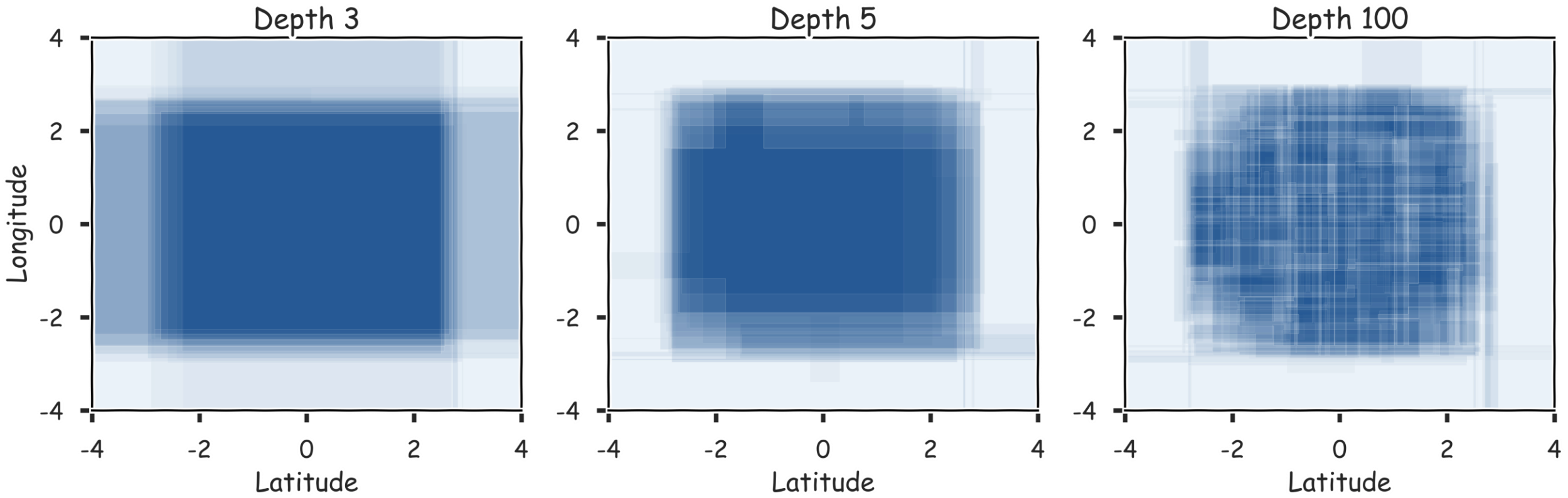


Two (2) magic realisms? What do I do with them?

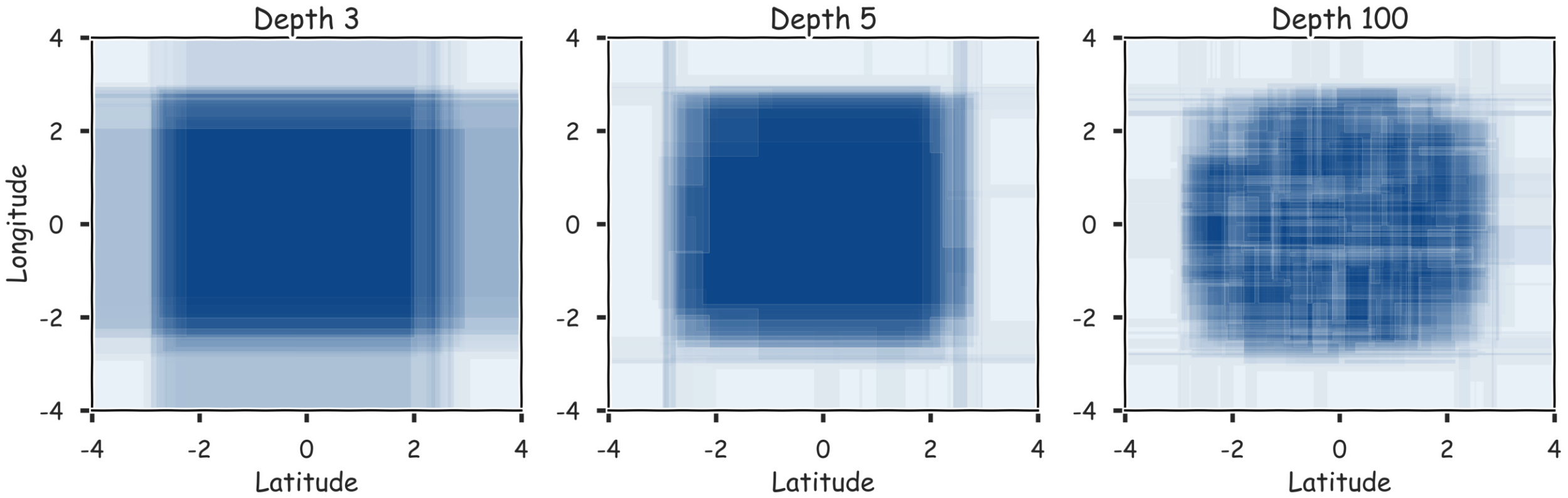
Combine them



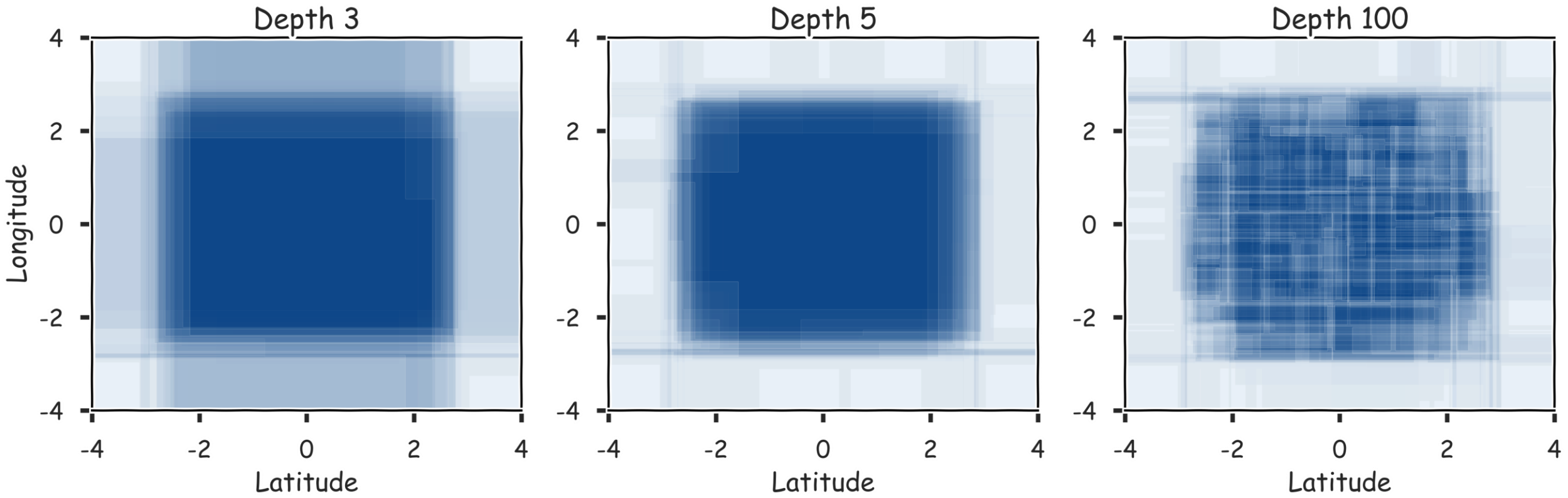
20 magic realisms



100 magic realisms



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 deeper 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.

This method is called ***Bagging*** (Breiman, 1996), short for, of course, **B**ootstrap **Agg**regating.

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.

Bagging

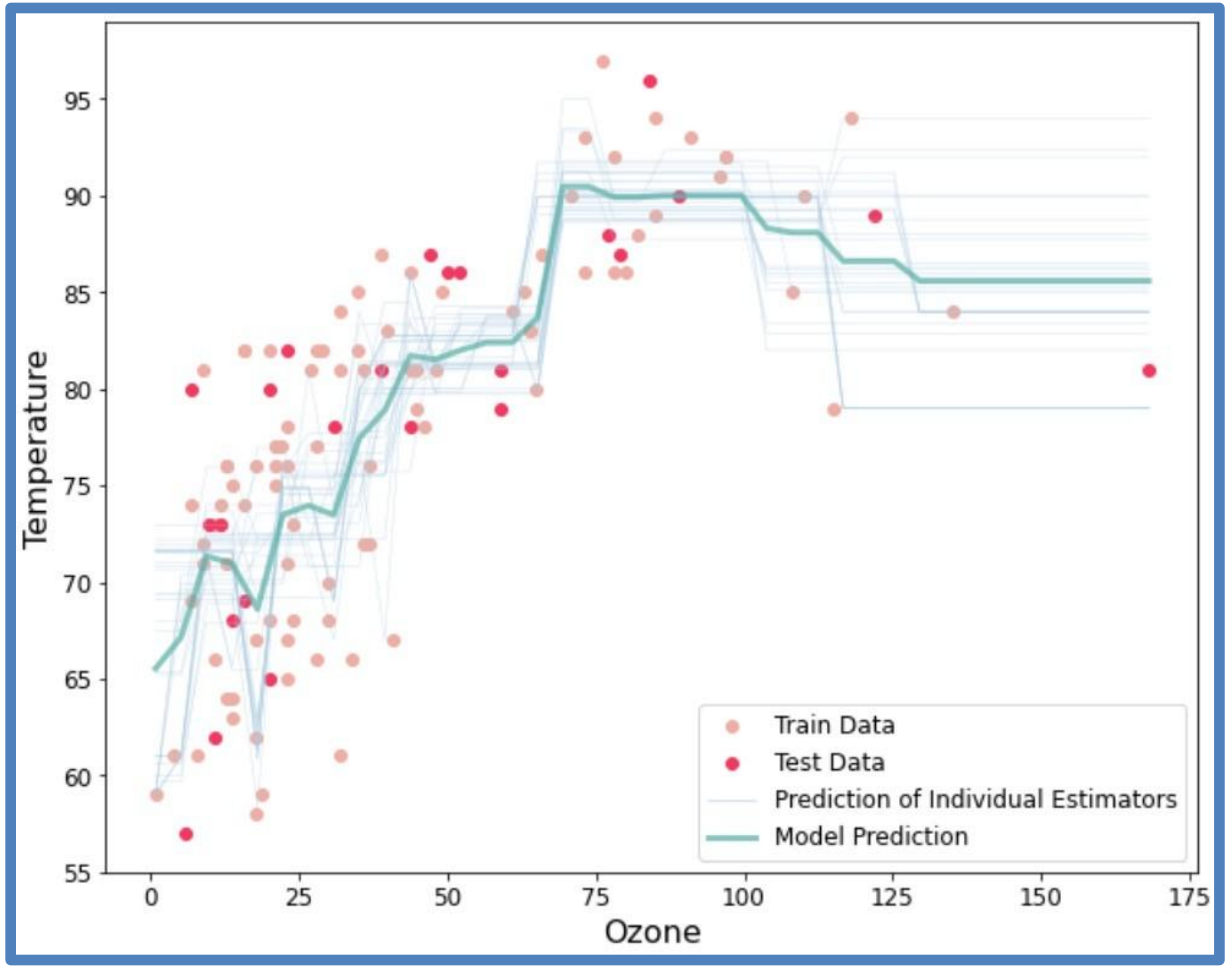
Bagging enjoys the benefits of:

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

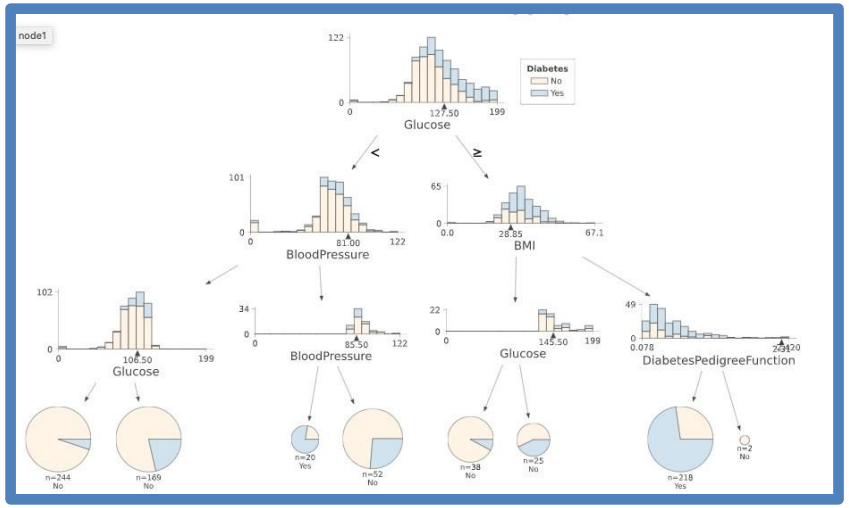
2. Low variance - averaging the prediction of all the models reduces the variance in the final prediction, assuming that we choose a sufficiently large number of trees.

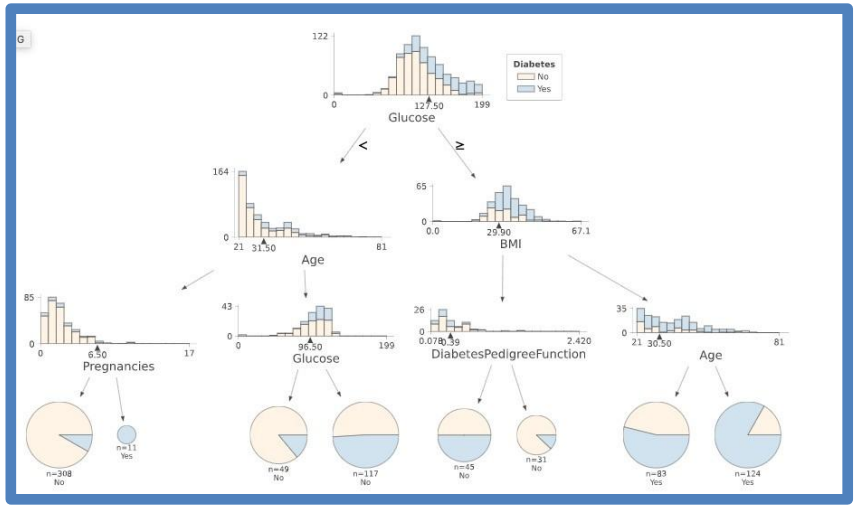
Bagging (regression)

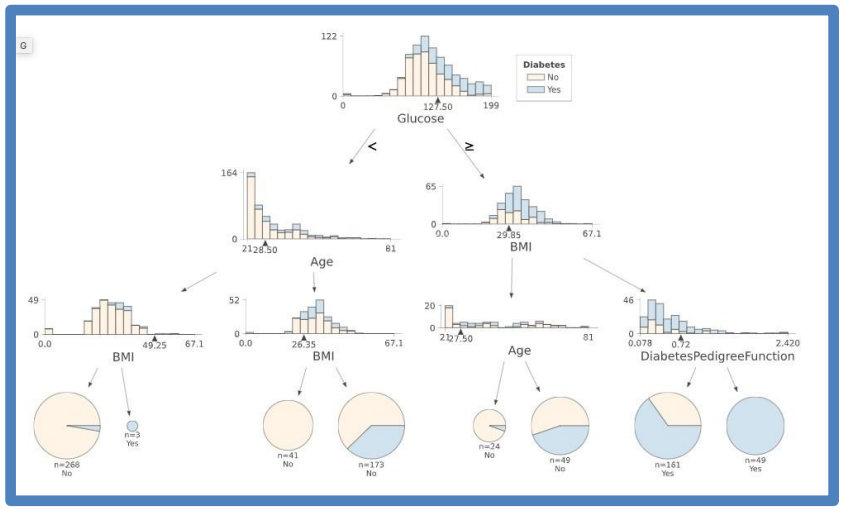
The resulting tree is the average of all tree (estimators).

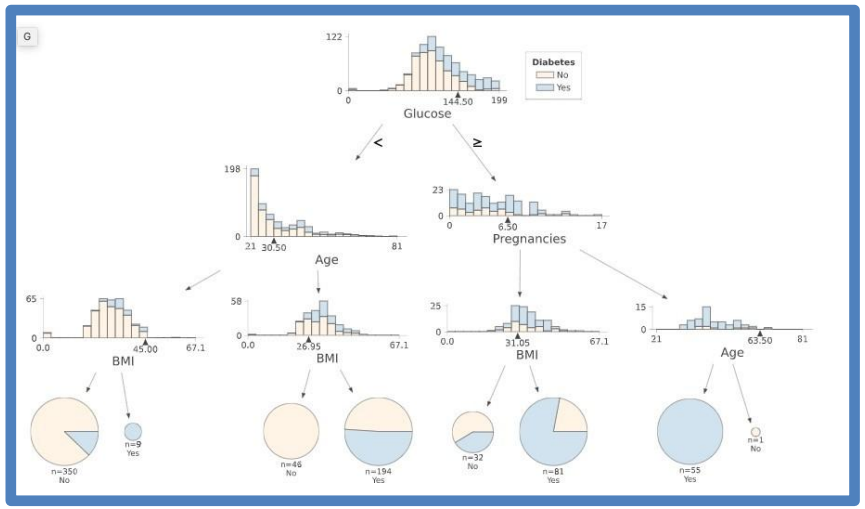


Bagging (classification)









For each bootstrap, we build a decision tree. The results is a combination (majority) of the predictions from all trees.

Bagging

Question: Do you see any problems?

Bagging

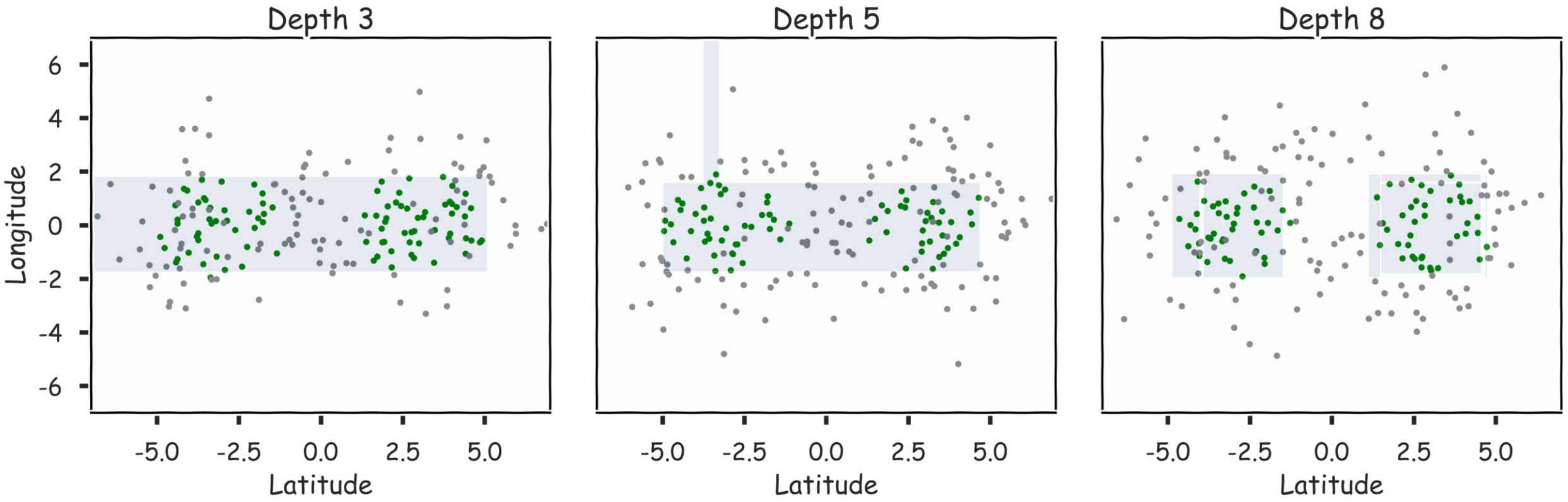
Question: Do you see any problems?

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

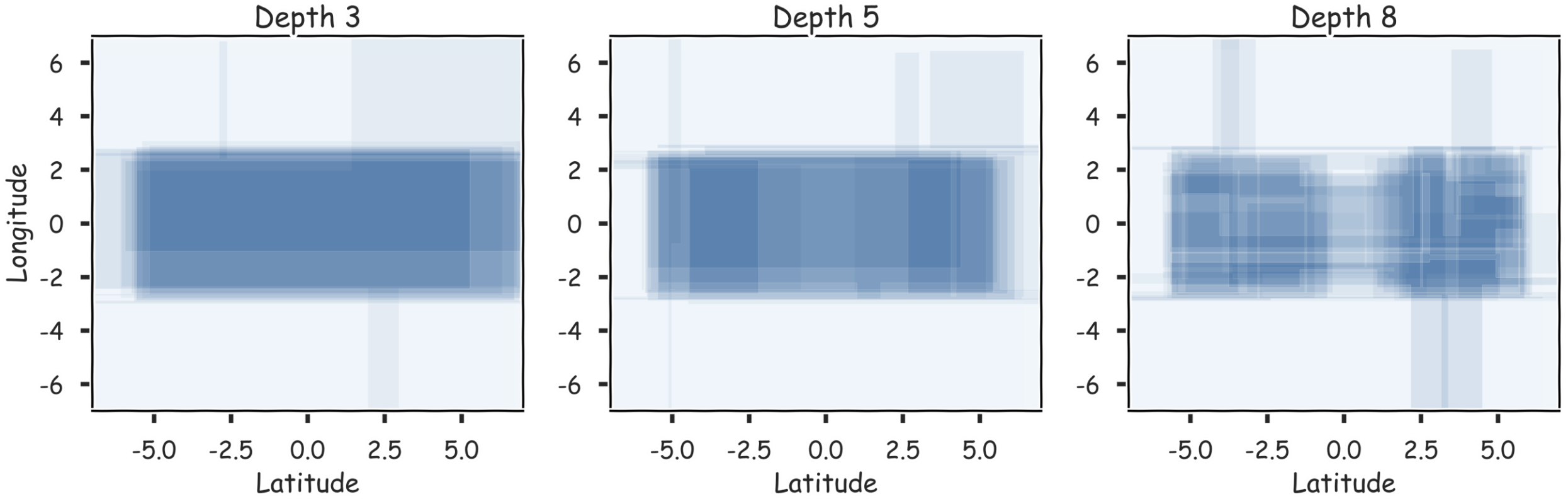
Case of underfitting

Consider the dataset below. To capture the pattern we need deeper tree.



Case of underfitting

Here we fit 100 trees using bootstrapped samples. Even with multiple estimators, the shallow tree will not be able to capture the real pattern.



Bagging

Question: Do you see any problems?

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

Question: How do we decide on the complexity of the model?

Cross Validation

Outline

- Review of Decision Trees
- Bagging
- **Out of Bag Error (OOB)**
- Variable Importance

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

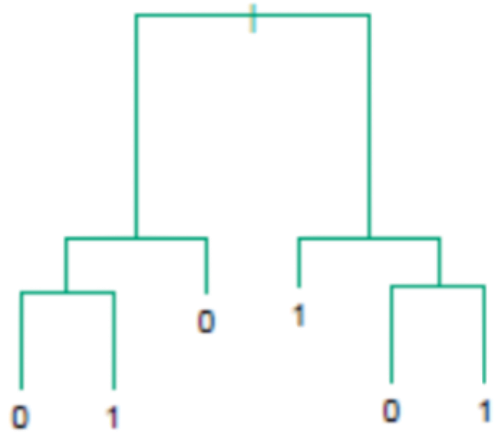
Original Data

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

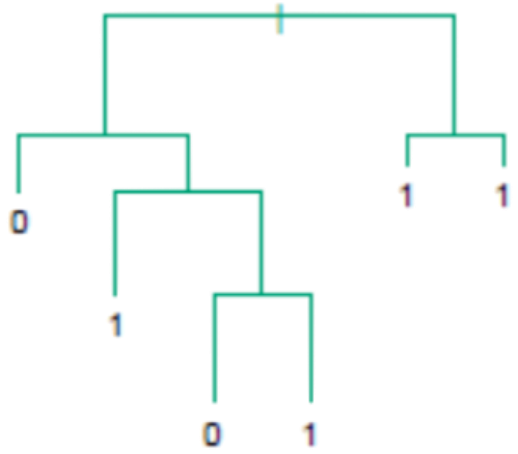
Original Data

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

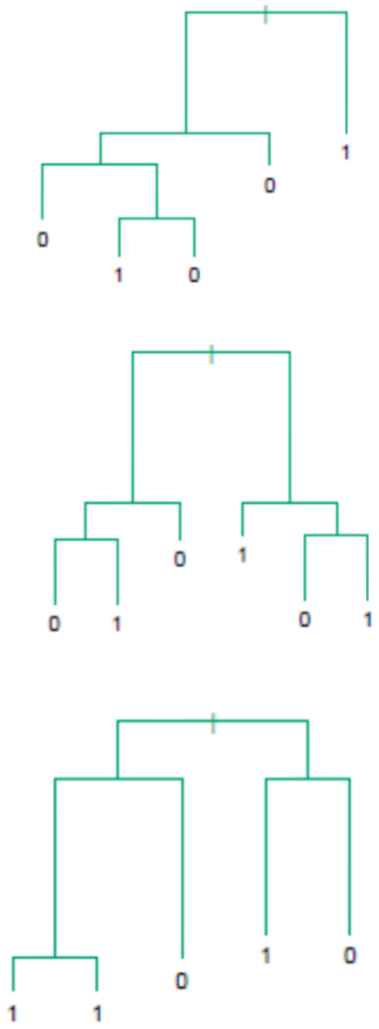
Point-wise out-of-bag error

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

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

Regression

$$\hat{y}_{i,pw} = \sum_{j \in B} \hat{y}_{i,j} \qquad e_i = (y_i - \hat{y}_{i,pw})^2$$

OOB Error

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

Classification

$$Error_{OOB} = \frac{1}{B}\sum_{i}^{B} e_i = \frac{1}{B}\sum_{i}^{B} \mathbb{I}(\hat{y}_{i,pw} \neq y_i)$$

Regression

$$Error_{OOB} = \frac{1}{B}\sum_{i}^{B} e_i = \frac{1}{B}\sum_{i}^{B} (y_i - \hat{y}_{i,pw})^2$$

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?

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

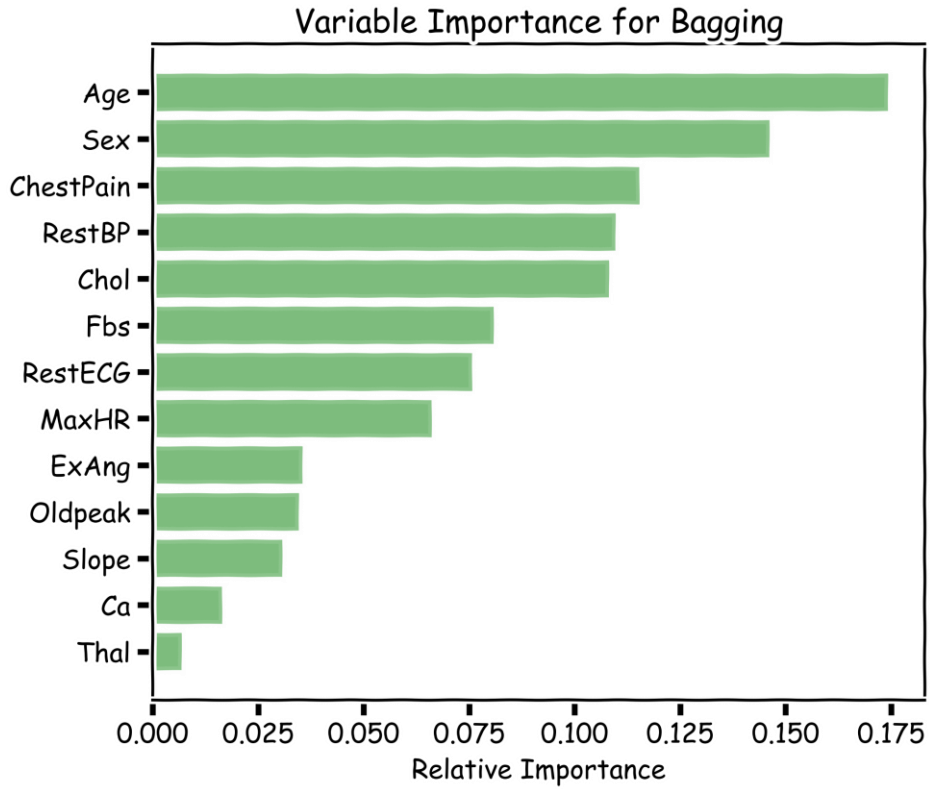
  The **major drawback** of bagging (and other ***ensemble methods*** that we will study) is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the 'logic' of an output through a series of decisions based on predictor values!

Outline

- Review of Decision Trees
- Bagging
- Out of Bag Error (OOB)
- **Variable Importance**

Variable Importance for Bagging

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.

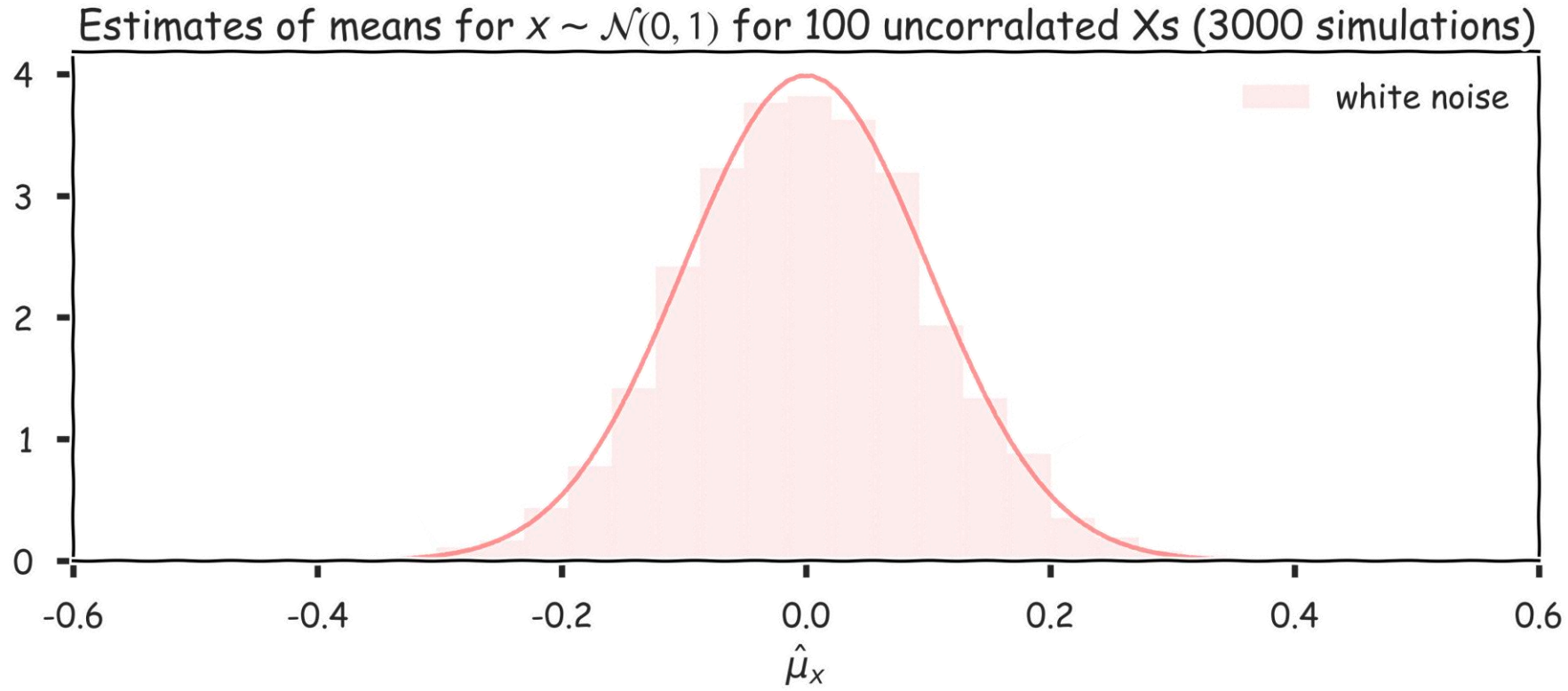
Suppose we have an extremely strong predictor, x_j , in the training set amongst moderate predictors. Then the greedy learning algorithm ensures that most of the models in the ensemble will choose to split on x_j in early iterations.

However, we assumed that each tree in the ensemble is independently and 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 :

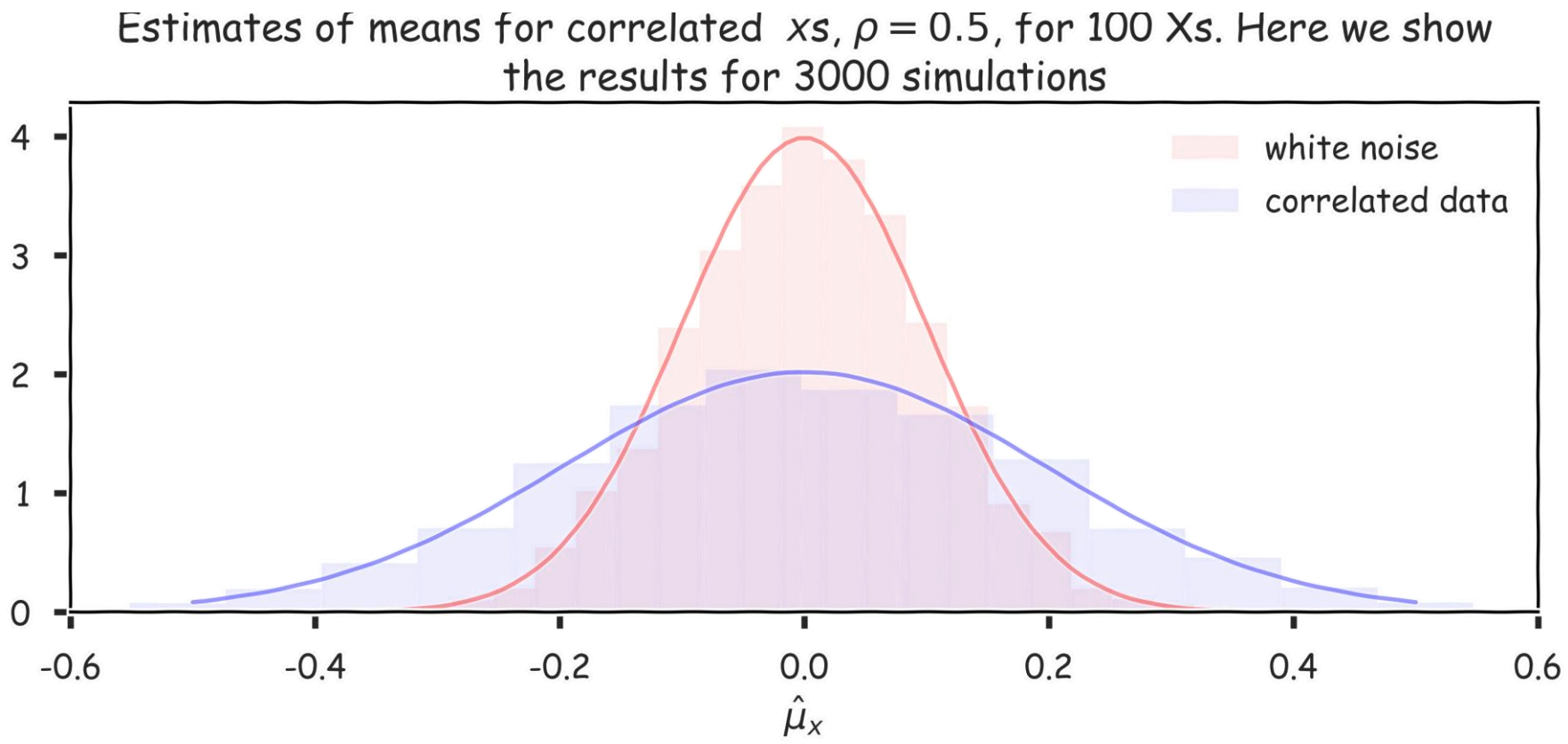
$$\text{var}(\hat{\mu}_x) = \frac{\sigma^2}{B}$$



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

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



Random Forest


I Entered
A Random Forest
Now I see
The Future

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 maximum depth or 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 or max_depth 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

1. Mean Decrease in Impurity (MDI)

- Same as Bagging.
- Record the prediction accuracy on the *oob* samples for each tree.
- 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 trees.
- 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.
- The default in Scikit-learn `feature_importances_`

Variable Importance for RF

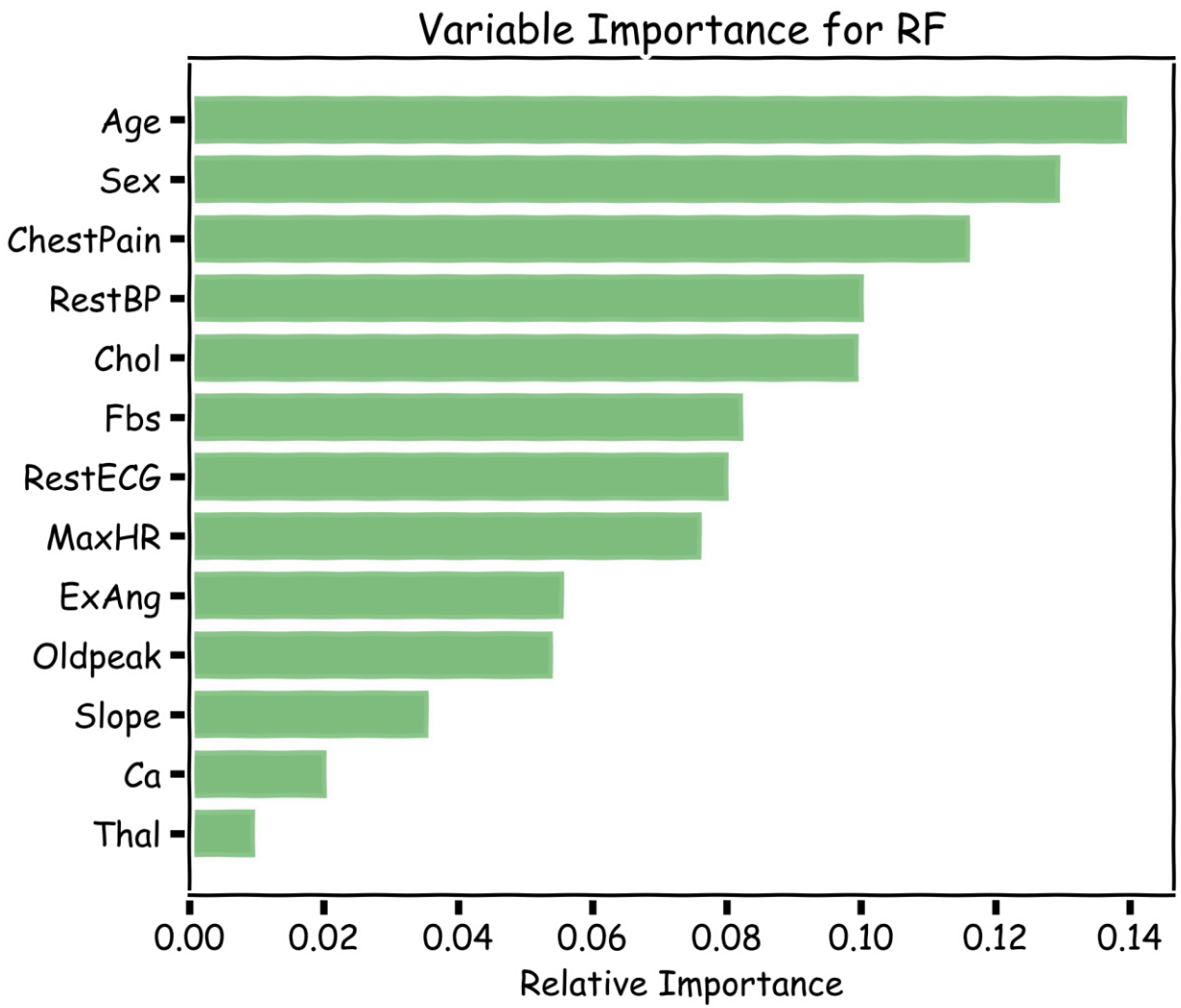
2. Permutation Importance

- 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.

3. One step further (SHAP values, LIME)

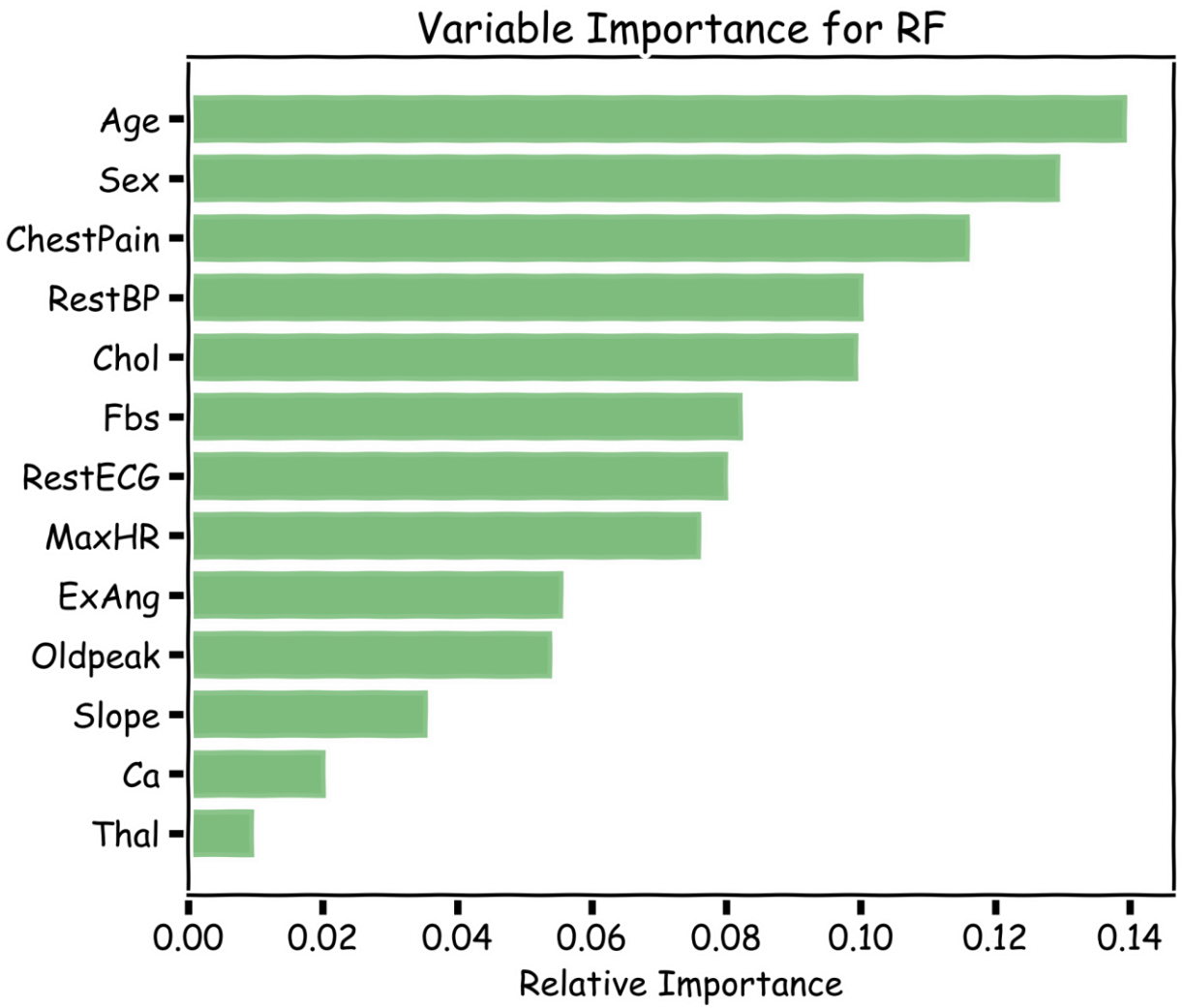
- We will see these methods in later lectures.

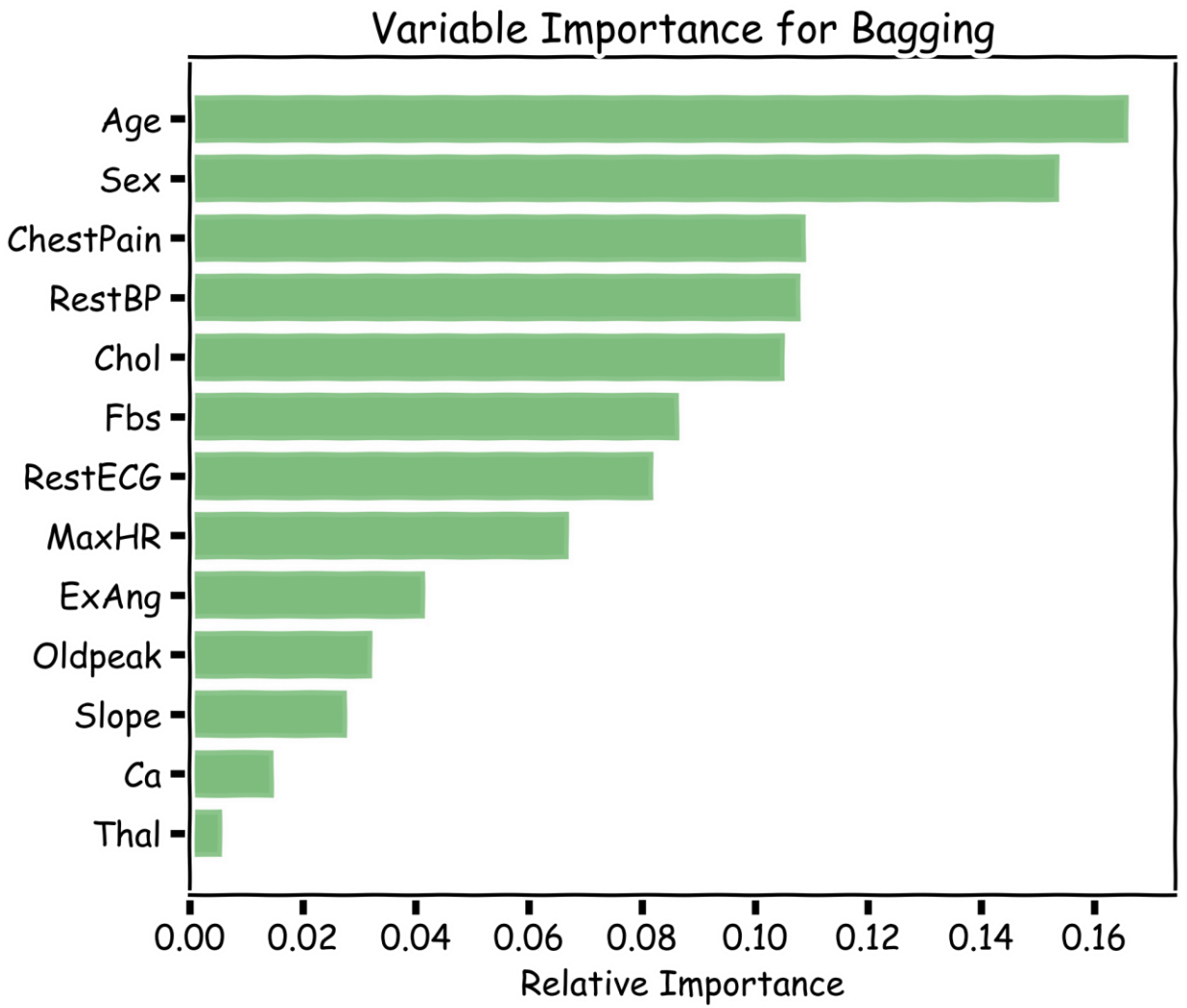
Variable Importance for RF



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

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) can return probabilities.
- **Question**: How?

Jordan Goldmeier
@Option_Explicit

Where do data scientists go camping?

In random forests

Question 4

In a decision tree:

A) Only leaves test rules; internal nodes predict

B) Only the root predicts; leaves pass inputs upward

C) Root/internal nodes test rules; leaf nodes make predictions

D) All nodes only store training data; a separate model predicts

Question 5

Compared with k-NN, a trained decision tree at test time typically requires:

A) Computing distances to all training points

B) Just a few feature–value comparisons along a path

C) Solving an optimization for each test point

D) Averaging over all training labels

Question 6

In tree learning, a "homogeneous" region is one where:

A) Features are standardized

B) Inputs are evenly spaced

C) Most examples share the same label/output

D) The entropy is maximized

Question 7

For **classification**, a popular split rule prefers the split that:

A) Maximizes information gain (largest entropy reduction)

B) Minimizes training accuracy

C) Maximizes the number of children

D) Minimizes the number of features

Question 8

For **regression trees**, a common leaf prediction is to:

A) Return the median of the full dataset

B) Return the parent node's mean

C) Return the average of outputs in that region

D) Return the nearest neighbor's value

Question 9

Which statement about **entropy** is true?

A) Higher entropy ⇒ higher purity

B) Lower entropy ⇒ higher purity

C) Entropy is irrelevant to information gain

D) Entropy only applies to regression

Question 10

A typical **stopping condition** when growing a tree is:

A) Stop after exactly 3 levels

B) Stop if a node's examples are all the same class (pure)

C) Stop whenever training accuracy is below 50%

D) Never stop; always prune later

Question 11

If a tree is **too deep**, it most likely:

A) Underfits due to high bias

B) Has low variance

C) Overfits by modeling noise (high variance)

D) Cannot achieve low training error

Question 12

In **cost-complexity pruning**, the criterion $C(T) = Error(T) + \alpha \mid T \mid$ uses $\mid T \mid$ to denote:

A) The number of internal nodes

B) The number of leaves in the tree

C) The number of features

D) The depth of the tree

Question 13

Bagging a high-variance learner involves:

A) Bootstrapping the data and averaging/majority-voting predictions

B) Removing outliers and retraining one model

C) Randomly deleting features once

D) Training on the full data and averaging parameters

Question 14

A primary benefit of bagging deep trees is:

A) Lower bias due to shallower trees

B) Lower variance in the aggregated predictor

C) Perfect interpretability of the ensemble

D) Guaranteed zero training error

Question 15

A common **drawback of bagging/ensembles** compared with a single tree is:

A) Slower training for each base learner

B) Inability to handle categorical features

C) Reduced interpretability of the overall model

D) Necessity of linear decision boundaries

Question 16

The **out-of-bag (OOB) error** for bagging is computed by:

A) Cross-validating with new external data

B) Evaluating each training point using only the trees that did **not** include it in their bootstrap sample

C) Measuring training error on all trees

D) Using only the deepest tree in the ensemble

Question 17

Random forests de-correlate trees mainly by:

A) Using shallower trees

B) Penalizing splits on repeated features

C) Considering a random subset of features at each split

D) Sharing leaf predictions across trees

Question 18

A common rule of thumb for the number of features considered per split in random forests is:

A) p for classification; $\sqrt{p}$for regression

B) $\sqrt{p}$ for both

C) $\frac{p}{2}$ for both

D) $\sqrt{p}$ for classification; $p/3$ for regression

Question 19

Mean Decrease in Impurity (MDI) importance for trees/forests measures:

A) Total reduction in impurity attributed to a feature, averaged over all trees

B) Change in training error if the feature is removed

C) The gradient magnitude of the loss w.r.t. the feature

D) The number of times the feature appears in the dataset

Question 20

Permutation importance (RF) is estimated by:

A) Counting how often a feature is used at the root

B) Shuffling the feature's values in OOB samples and measuring the drop in prediction accuracy

C) Removing the feature permanently and retraining the forest

D) Comparing split thresholds to the feature's mean

References

CS771: Intro to Machine Learning (Fall 2021), Nisheeth Srivastava, IIT Kanpur
CS109a: Introduction to Data Science, Pavlos Protopapas and Natesh Pillai, Havard University