Teoria degli Algoritmi

Corso di Laurea Magistrale in Matematica Applicata

a.a. 2020-21

Gabriele Tolomei

Dipartimento di Informatica

Sapienza Università di Roma

tolomei@di.uniroma1.it

SAPIENZA UNIVERSITÀ DI ROMA

Recap from Last Lectures

- We presented 2 linear models: linear regression and logistic regression
- Those hypotheses work well whenever there exists a linear relationship between the features (input) and the response (output)
- Model's parameter estimation done either analytically (OLS) or iteratively (Gradient Descent)

Tree-based Methods

- Suitable for both regression and classification tasks

Tree-based Methods

- Suitable for both regression and classification tasks
- Work by repeatedly splitting the input feature space into a number of regions

Tree-based Methods

- Suitable for both regression and classification tasks
- Work by repeatedly splitting the input feature space into a number of regions
- A prediction is computed by taking the mean (regression) or the mode (classification) of the points in the region which the observation belongs

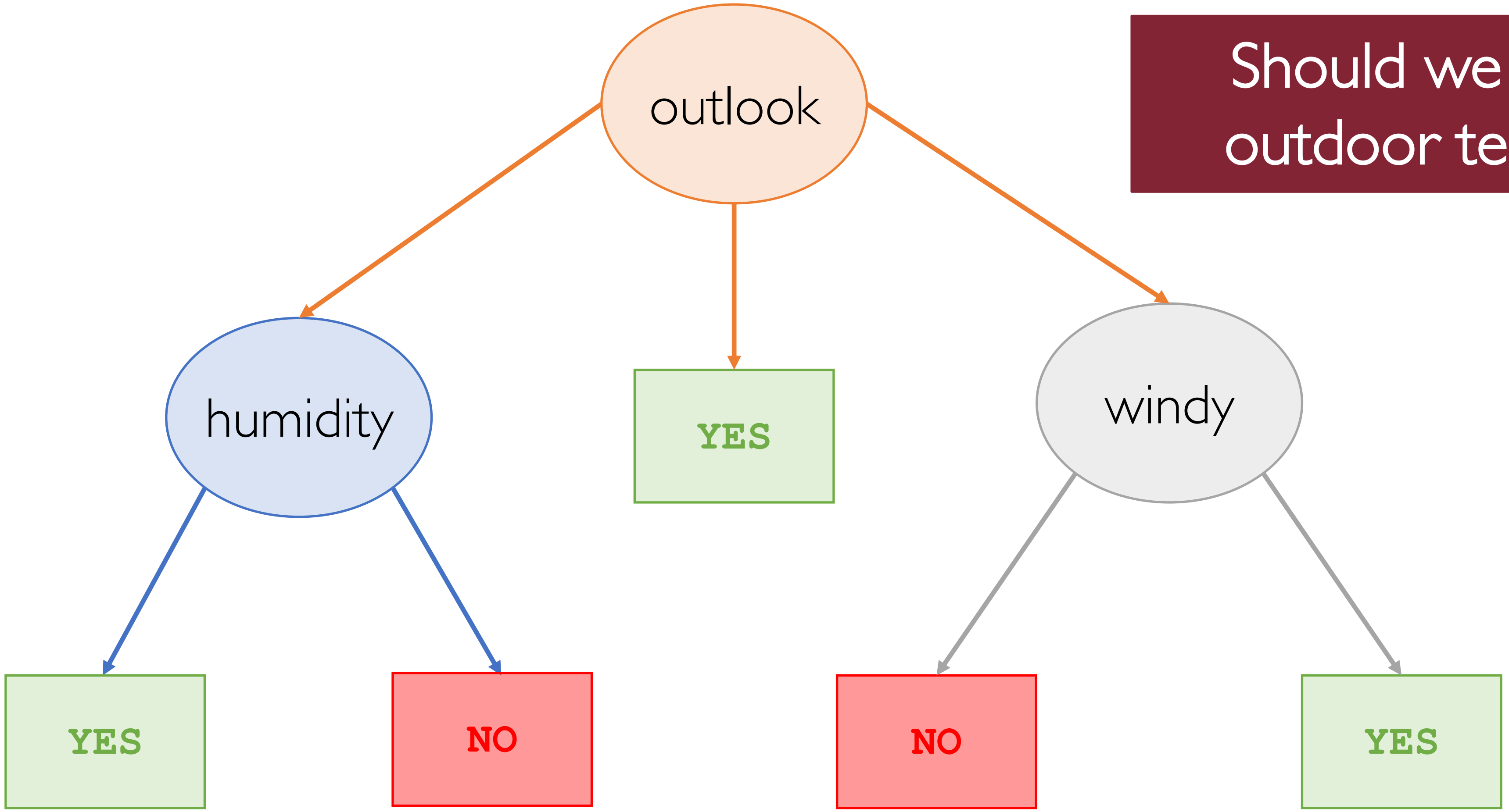
Tree-based Methods

- Suitable for both regression and classification tasks
- Work by repeatedly splitting the input feature space into a number of regions
- A prediction is computed by taking the mean (regression) or the mode (classification) of the points in the region which the observation belongs
- The set of splitting rules can be represented as a tree (decision tree)

Tree-based Methods

- Suitable for both regression and classification tasks
- Work by repeatedly splitting the input feature space into a number of regions
- A prediction is computed by taking the mean (regression) or the mode (classification) of the points in the region which the observation belongs
- The set of splitting rules can be represented as a tree (decision tree)
- Highly human-interpretable models

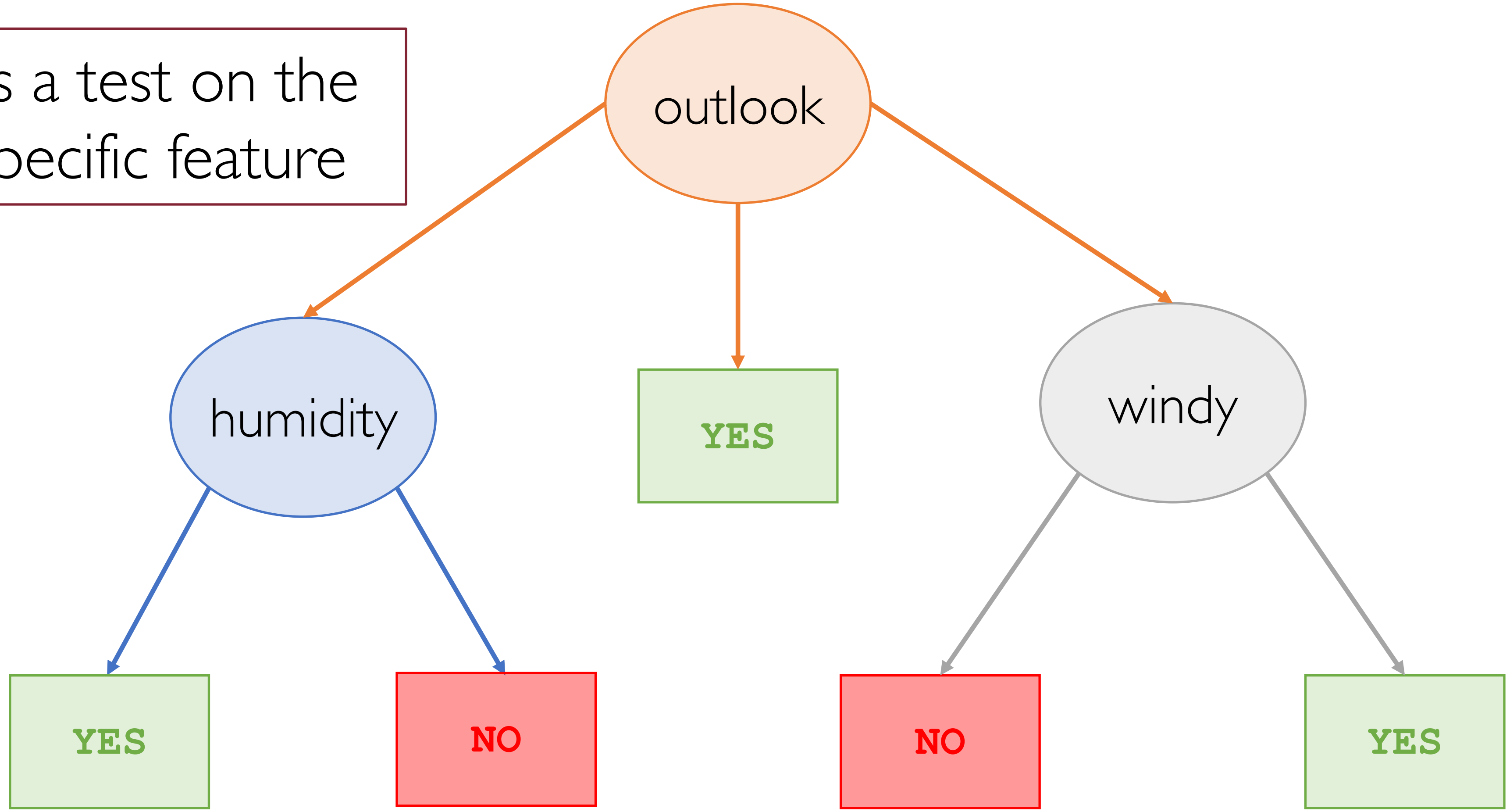
Decision Tree: An Example



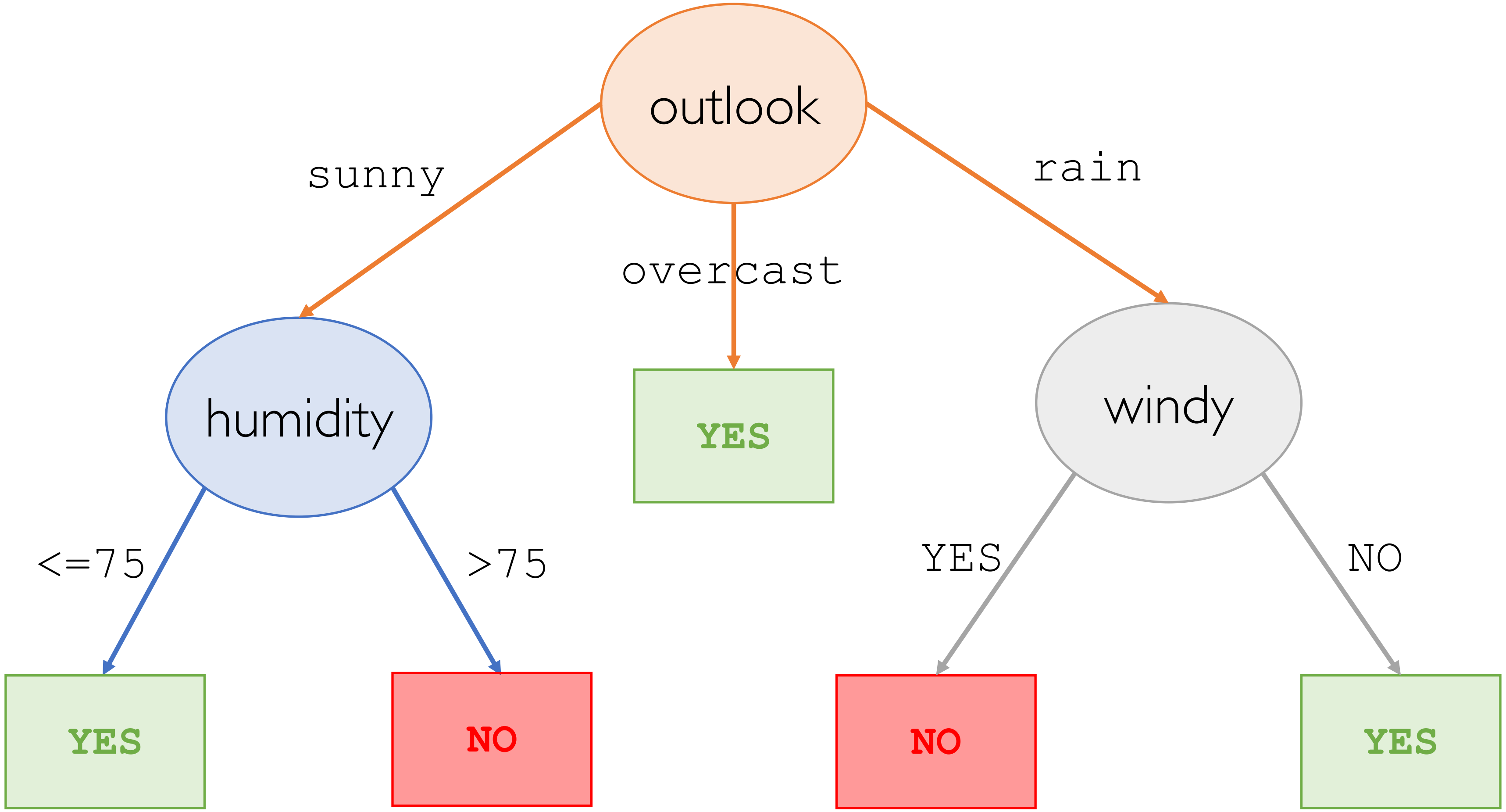
Should we play outdoor tennis?

Decision Tree: An Example

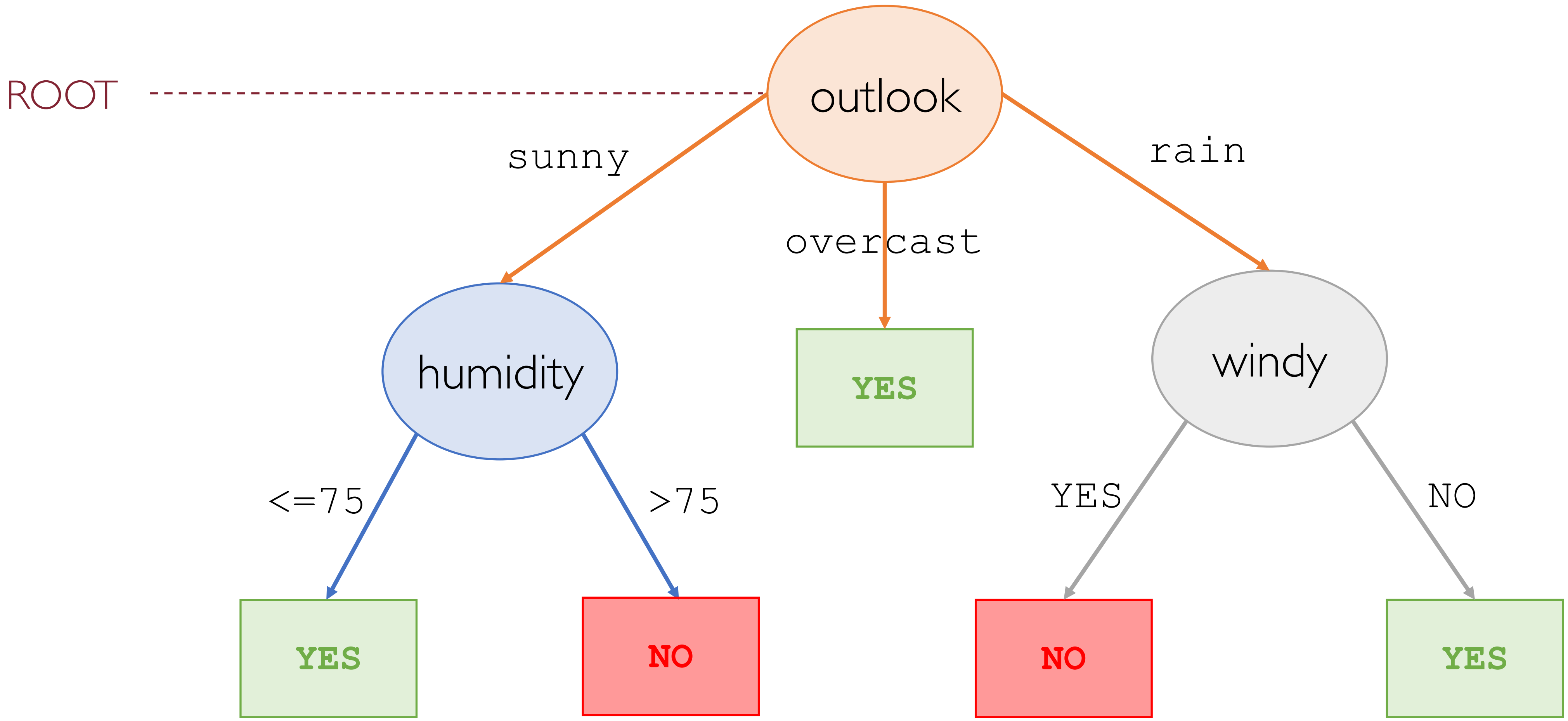
Each node is a test on the value of a specific feature



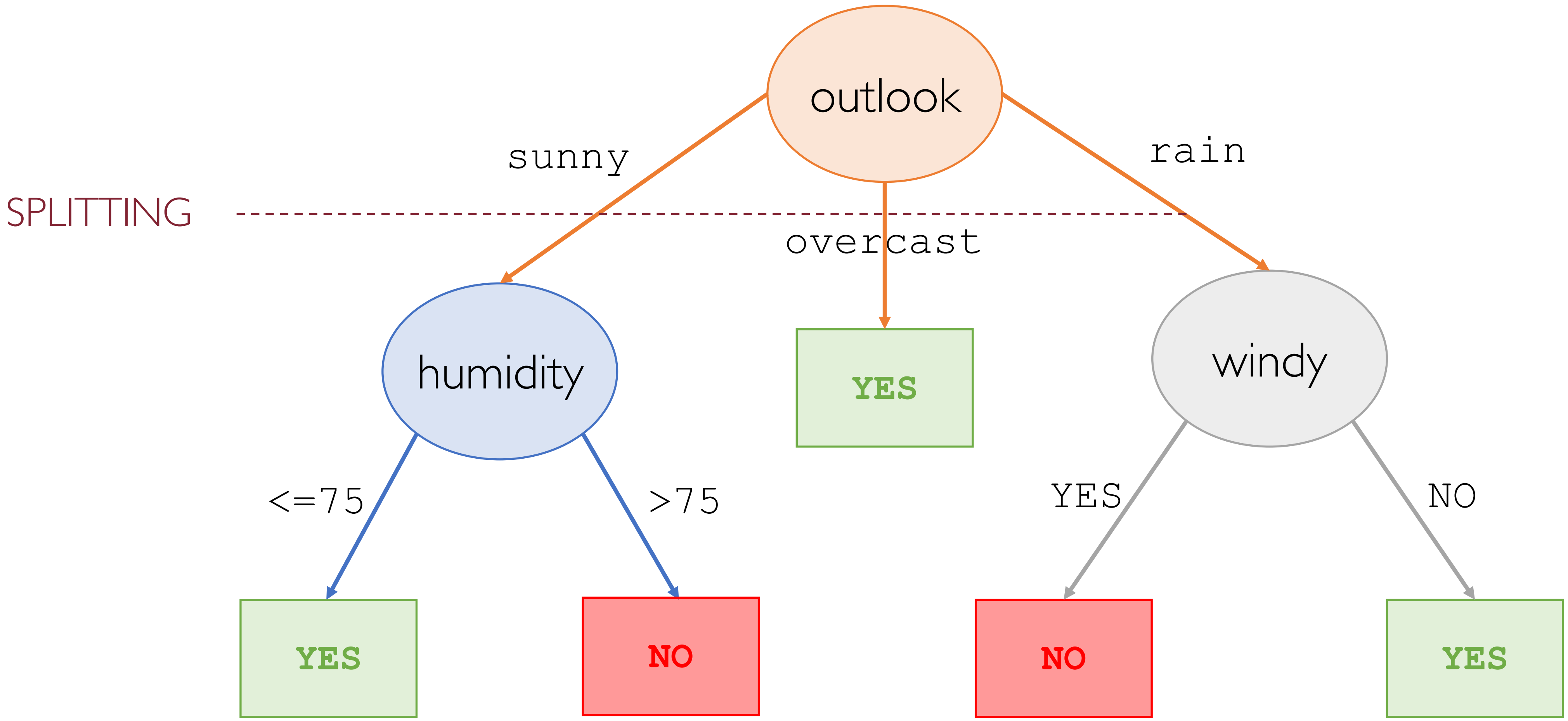
Decision Tree: An Example



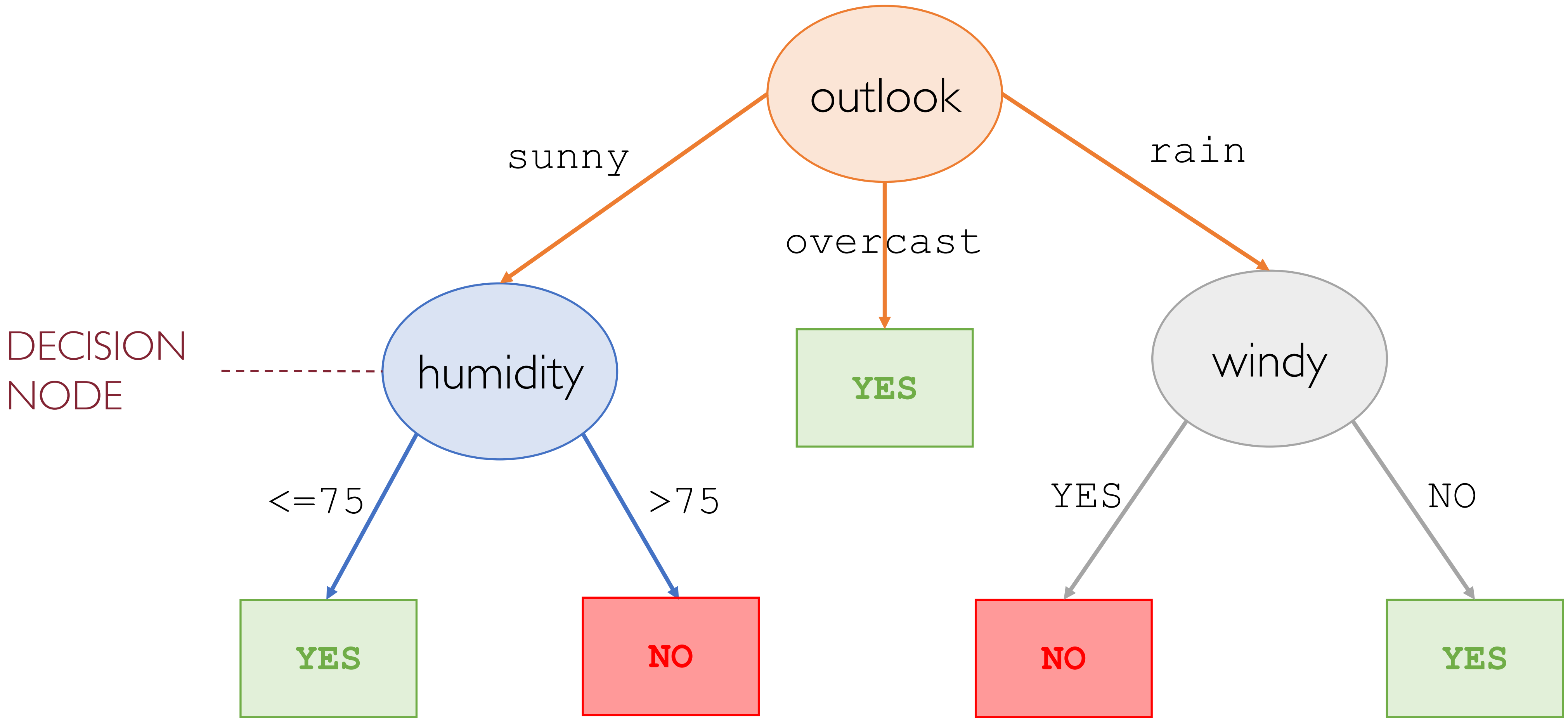
Decision Tree: An Example



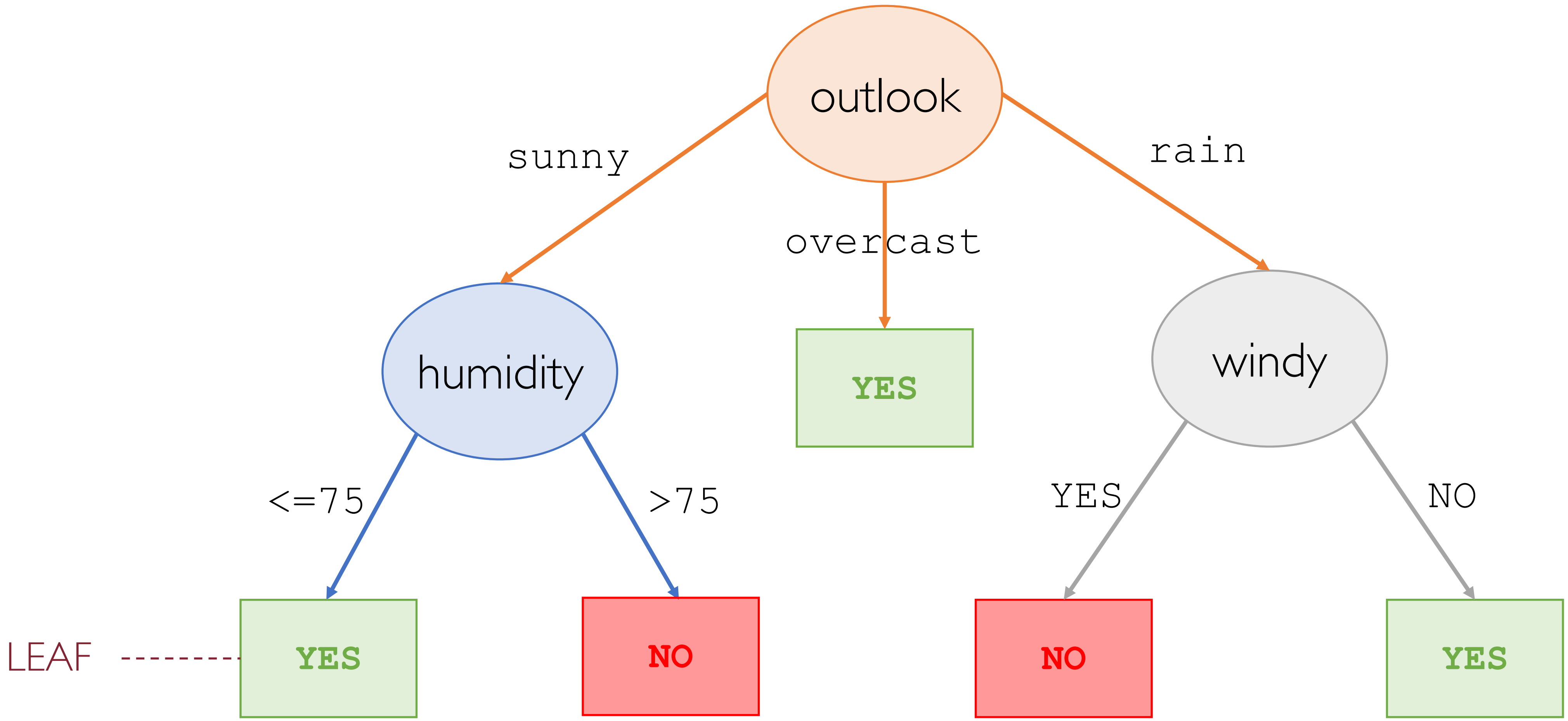
Decision Tree: An Example



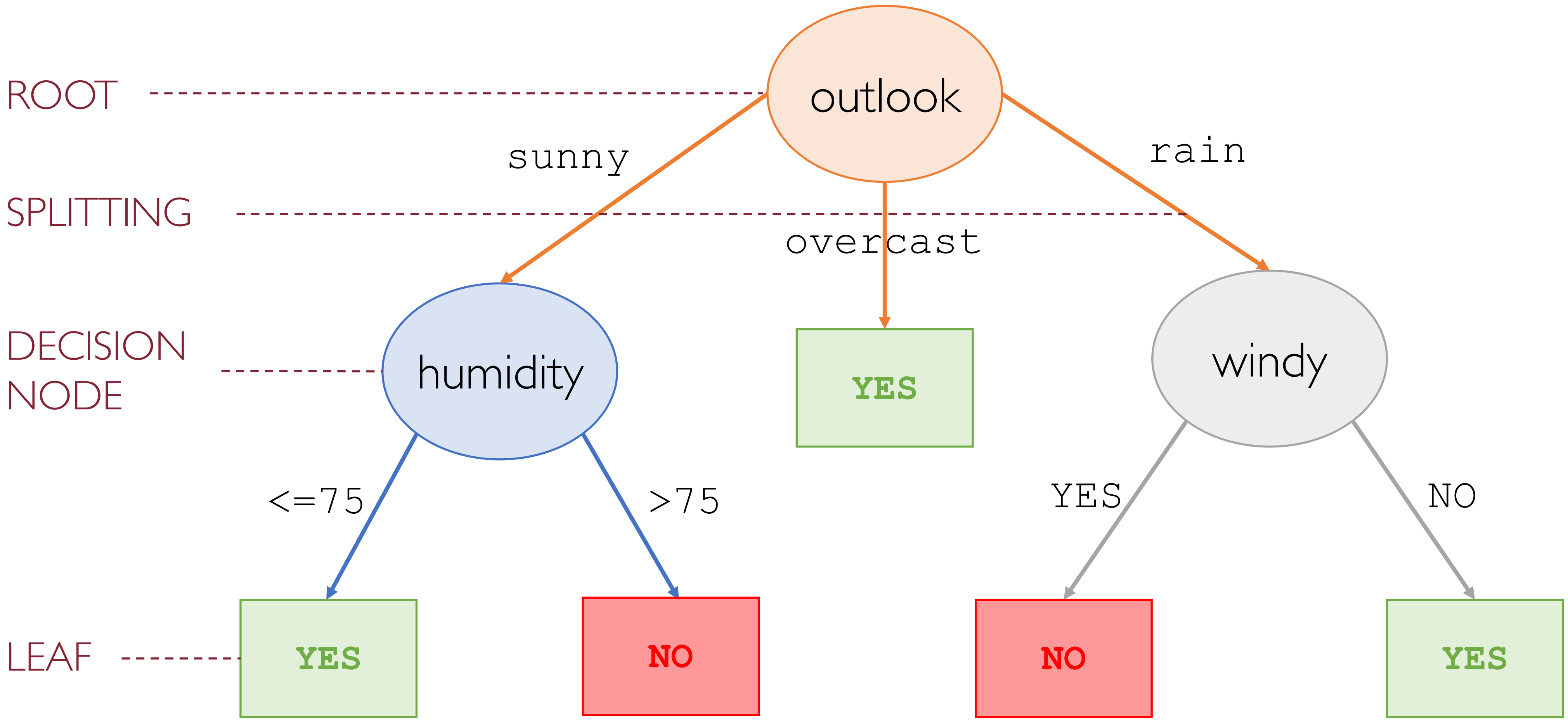
Decision Tree: An Example



Decision Tree: An Example

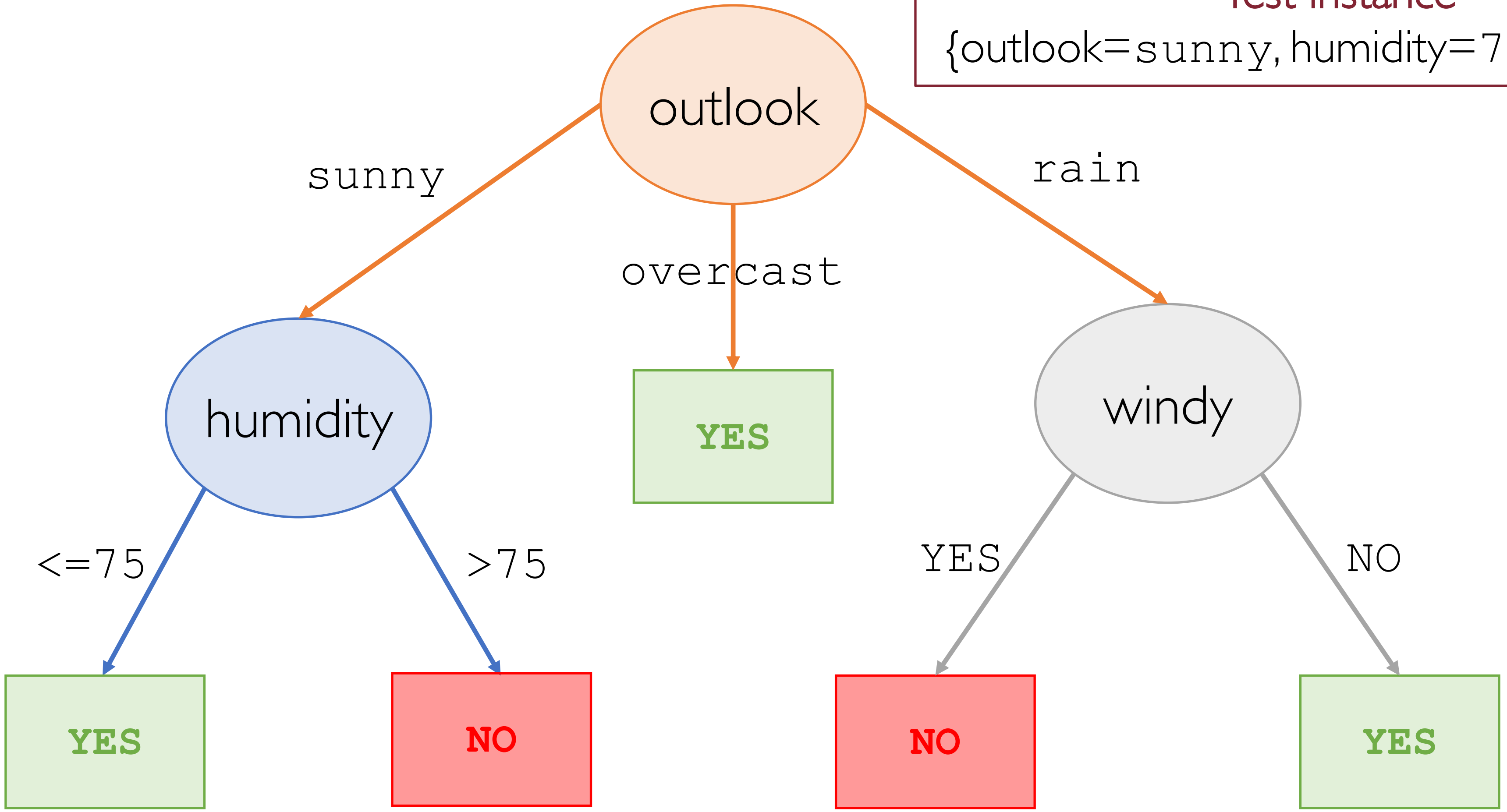


Decision Tree: An Example



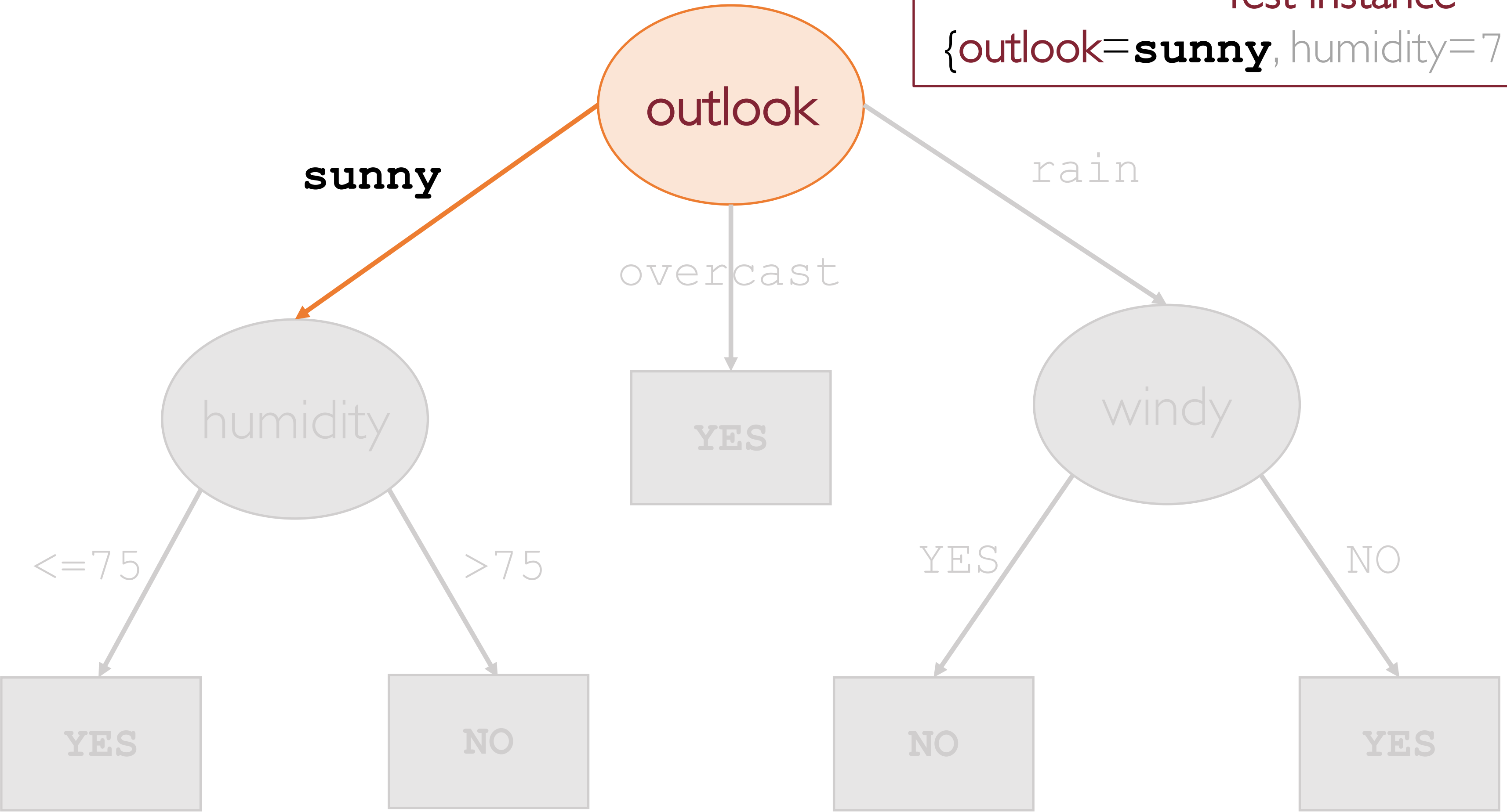
Decision Tree: An Example

Test instance
{outlook=sunny, humidity=72, windy=Y}



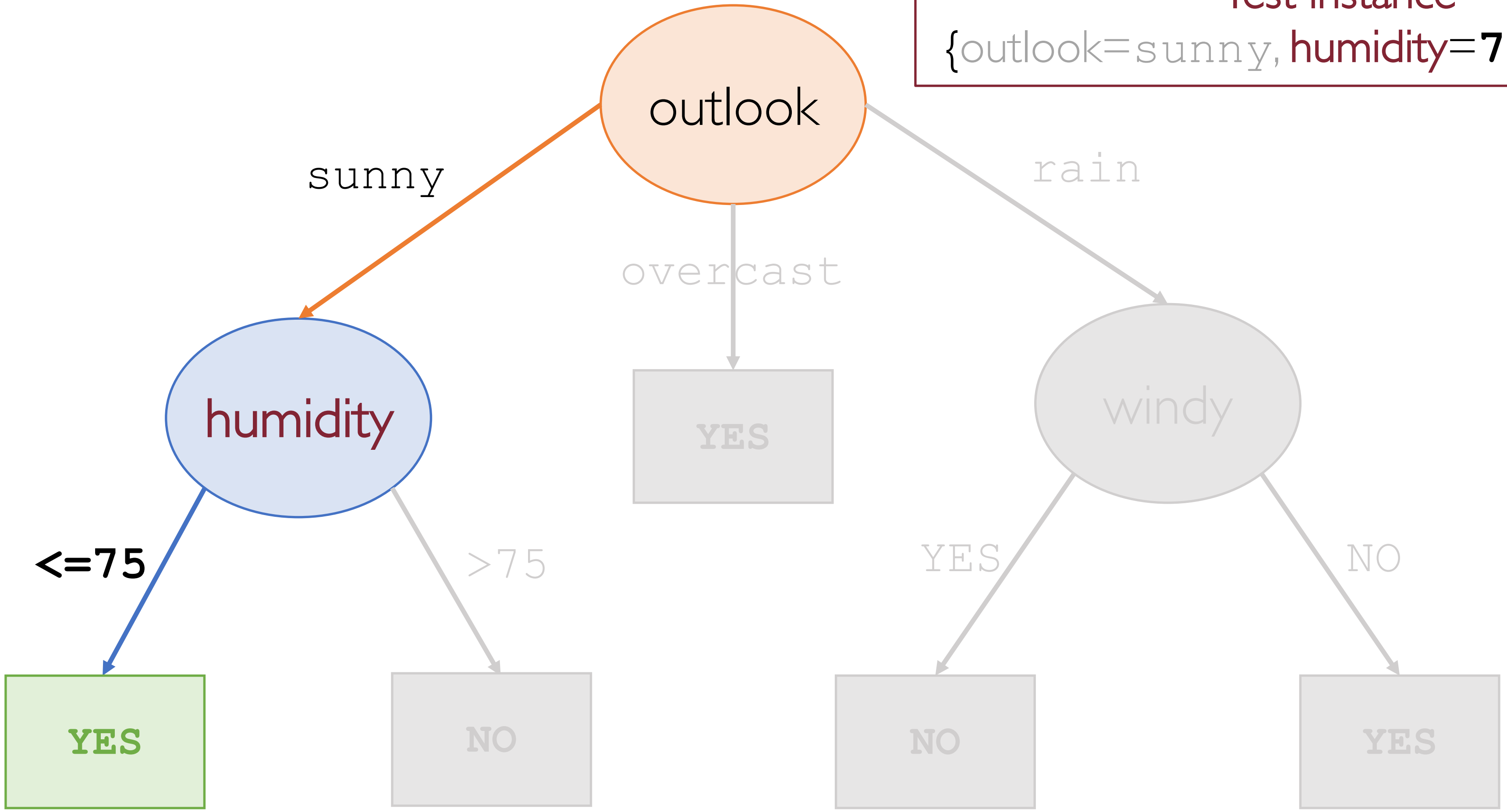
Decision Tree: An Example

Test instance
{outlook=**sunny**, humidity=72, windy=Y}



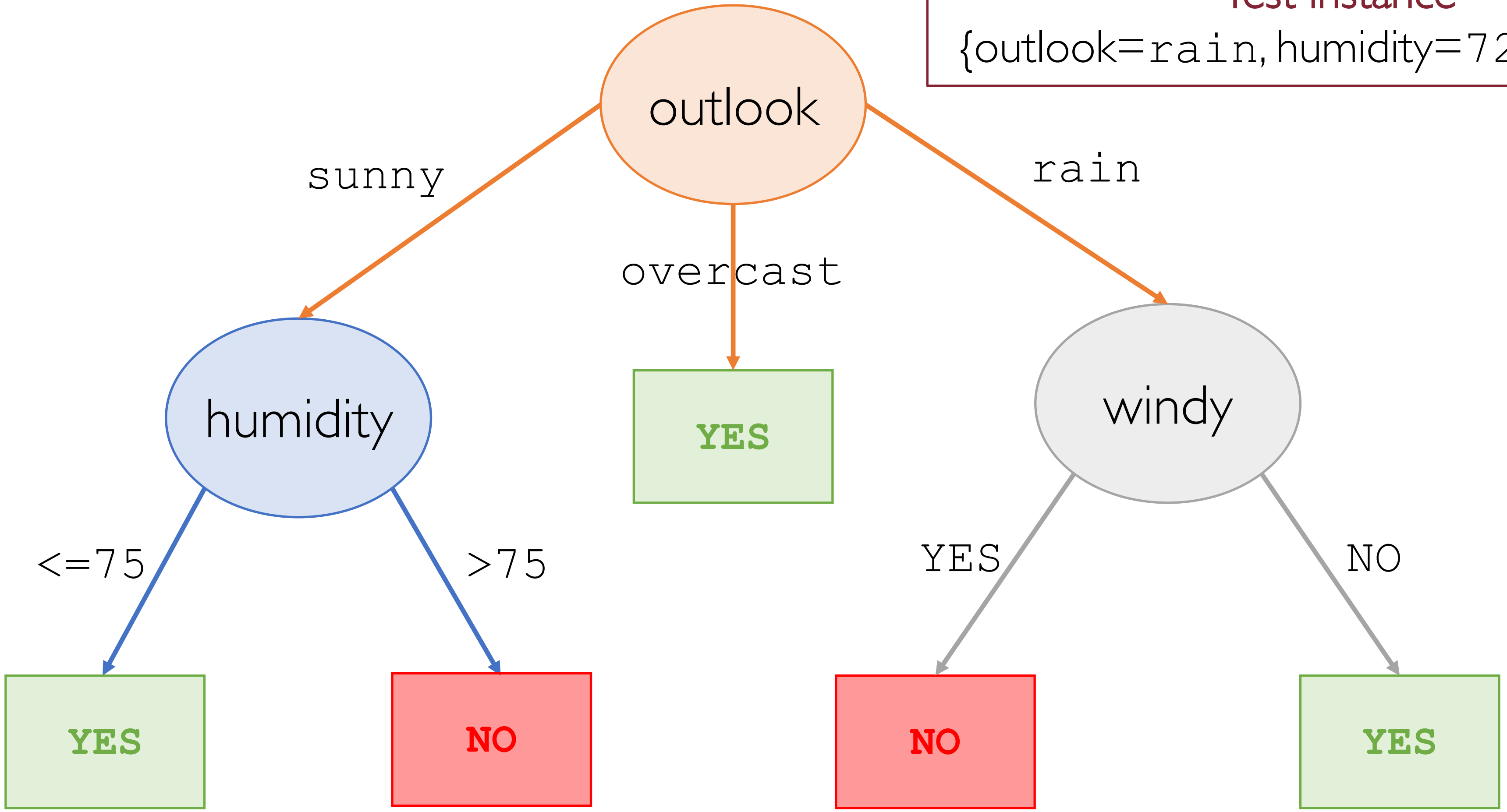
Decision Tree: An Example

Test instance
{outlook=sunny, humidity=**72**, windy=Y}



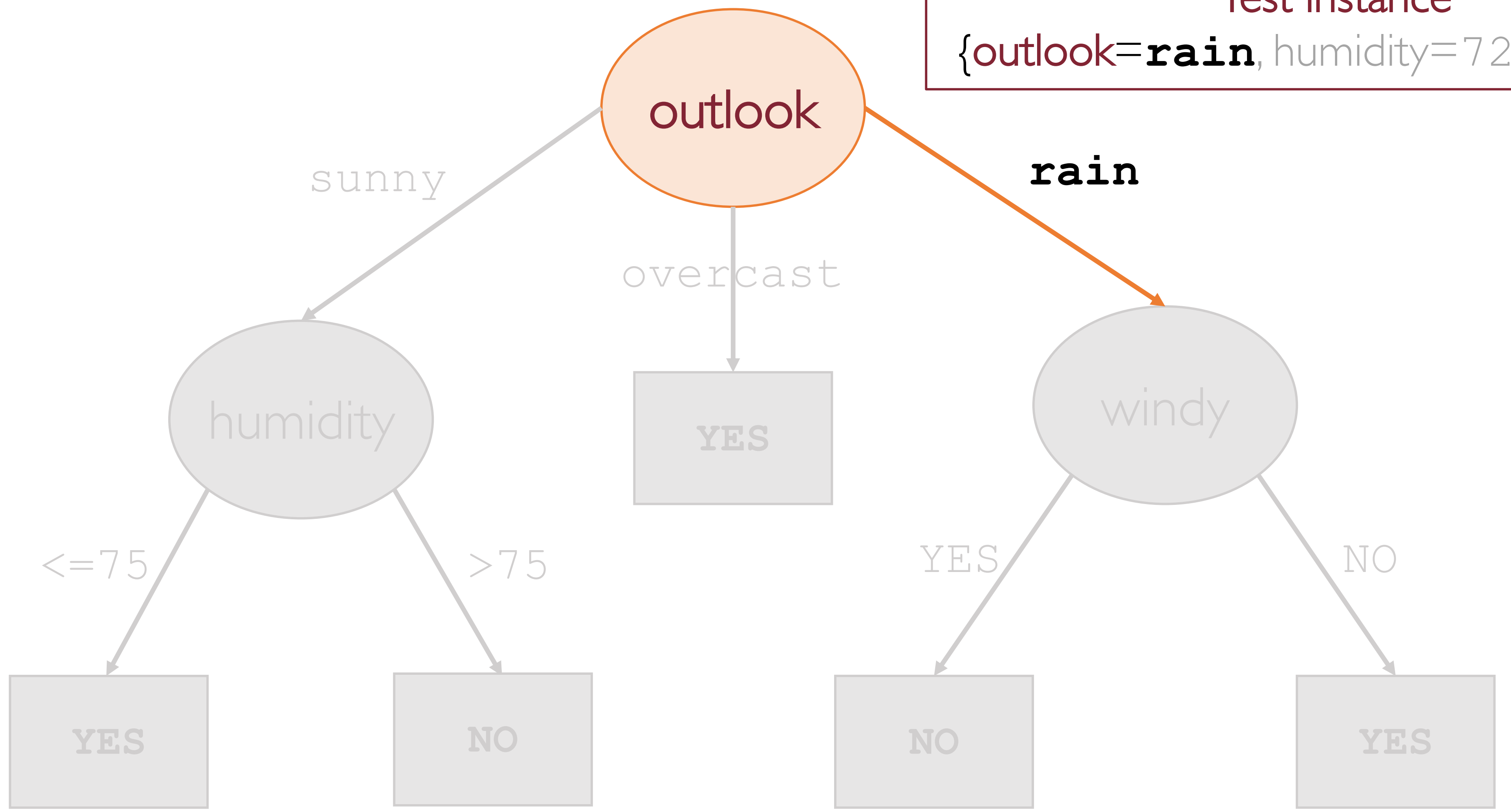
Decision Tree: An Example

Test instance
{outlook=`rain`, humidity=`72`, windy=`Y`}



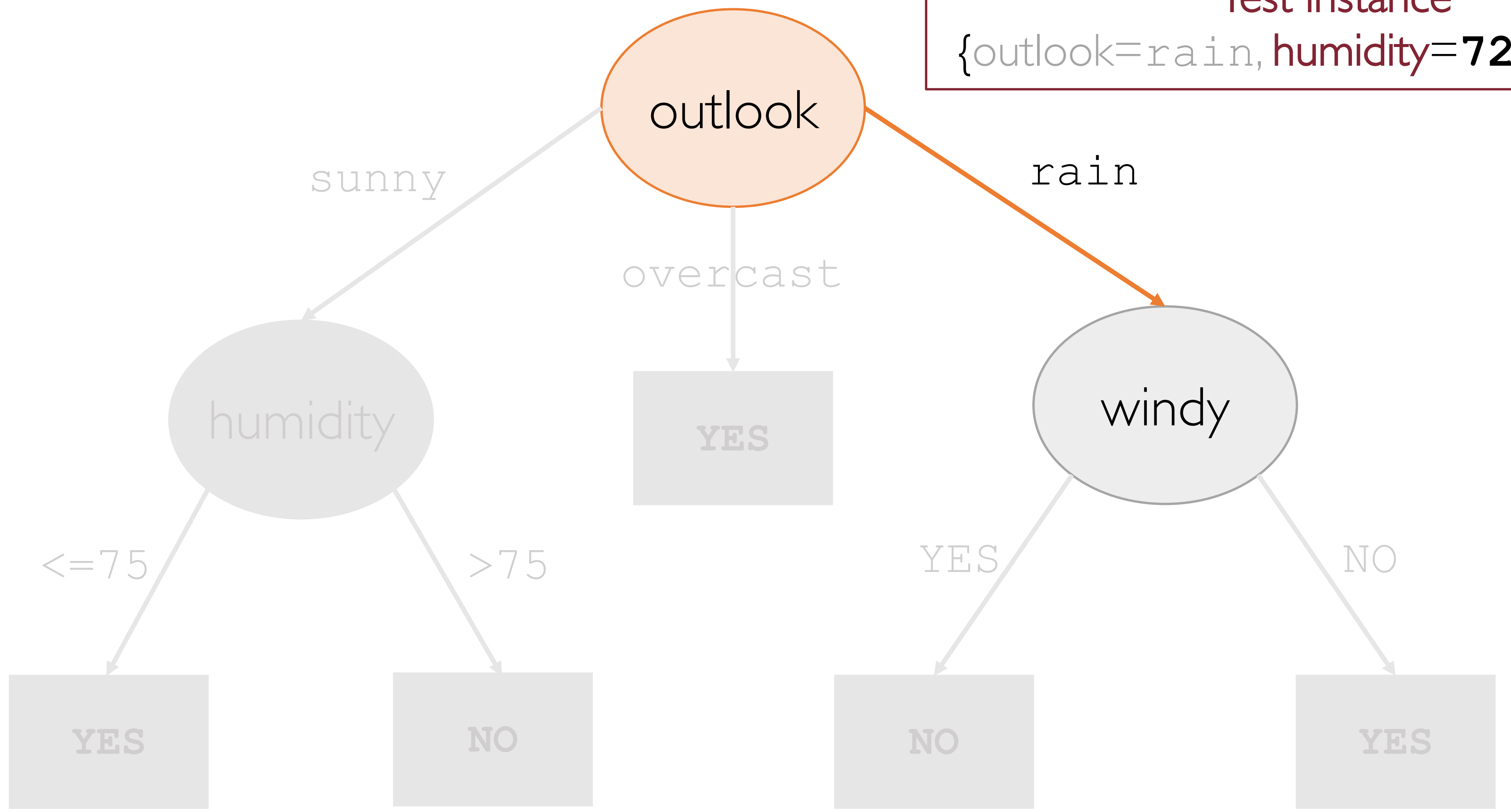
Decision Tree: An Example

Test instance
{outlook=**rain**, humidity=72, windy=Y}



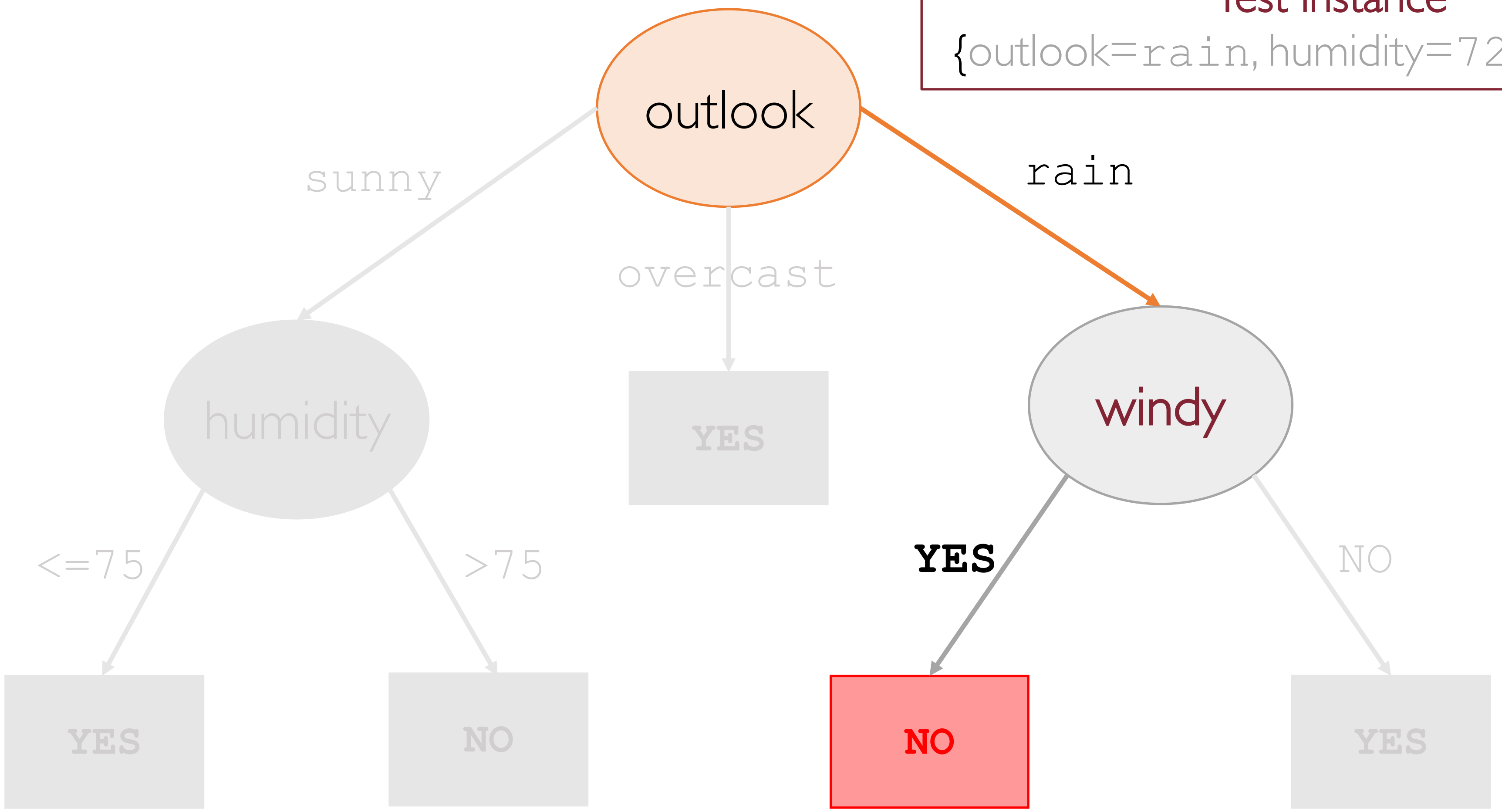
Decision Tree: An Example

Test instance
{outlook=rain, humidity=**72**, windy=Y}



Decision Tree: An Example

Test instance
{outlook=rain, humidity=72, windy=Y}



A Bit of Notation

$\mathcal{X} \subseteq \mathbb{R}^n$	input feature space
$\mathcal{Y}$	output space
$\quad \mathcal{Y} \subseteq \mathbb{R}$	real-value label (regression)
$\quad \mathcal{Y} = \{1, \ldots, k\}$	discrete-value label (k-ary classification)
$(\mathbf{x}_i, y_i)$	*i*-th labeled instance
$\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,n}) \in \mathcal{X}$	*n*-dimensional feature vector of the *i*-th instance
$y_i \in \mathcal{Y}$	label of the *i*-th instance
$\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$	dataset of *m* i.i.d. labeled instances

How Do We Build a Decision Tree?

- Split the input feature space (i.e., the set of possible values observed for each feature x_i) into a set of non-overlapping regions $R_1, R_2, \ldots, R_J$

How Do We Build a Decision Tree?

- Split the input feature space (i.e., the set of possible values observed for each feature x_i) into a set of non-overlapping regions R_1, R_2, . . ., R_J
- For every observation that falls into the region R_j, we make the same prediction, i.e., the mean of the response values in R_j

How Do We Build a Decision Tree?

- Split the input feature space (i.e., the set of possible values observed for each feature x_i) into a set of non-overlapping regions $R_1, R_2, \ldots, R_J$
- For every observation that falls into the region R_j, we make the same prediction, i.e., the mean of the response values in R_j
- Example:
  - Suppose we split the input feature space in 2 regions: R_1 and R_2 and the response mean as computed from $R_1 = 10$ and $R_2 = 20$
  - For any **x** belonging to R_1 (R_2) will be predicted 10 (20)

Partitioning the Input Space

- In theory, partitions could have *any* shape

Partitioning the Input Space

- In theory, partitions could have *any* shape
- We choose to divide the input feature space into hyper-rectangles (i.e., high-dimensional rectangles)

Partitioning the Input Space

- In theory, partitions could have *any* shape
- We choose to divide the input feature space into hyper-rectangles (i.e., high-dimensional rectangles)
- This way, the obtained model is easier to interpret

Partitioning the Input Space

- In theory, partitions could have *any* shape
- We choose to divide the input feature space into hyper-rectangles (i.e., high-dimensional rectangles)
- This way, the obtained model is easier to interpret

Minimize the Residual Sum of Squares

$$\text{RSS} = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

Partitioning the Input Space

- In theory, partitions could have *any* shape
- We choose to divide the input feature space into hyper-rectangles (i.e., high-dimensional rectangles)
- This way, the obtained model is easier to interpret

Minimize the Residual Sum of Squares

$$\mathrm{RSS} = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \boxed{\hat{y}_{R_j}})^2$$

The mean computed from observations in R_j

Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

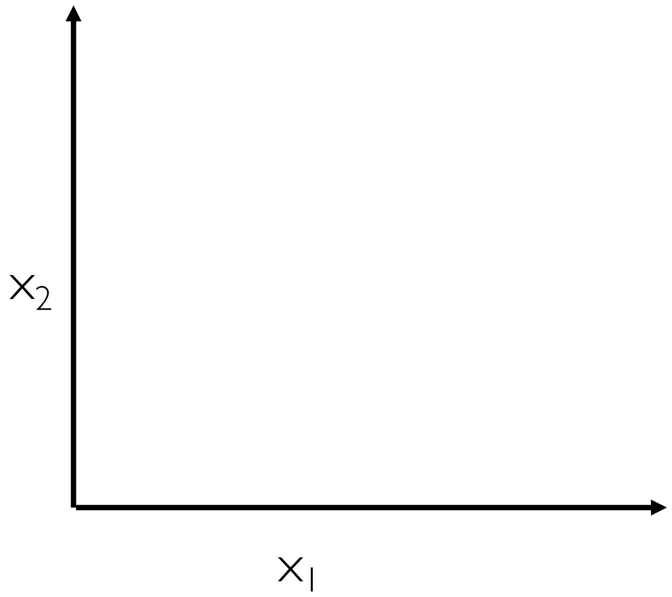
One branch for each value

Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

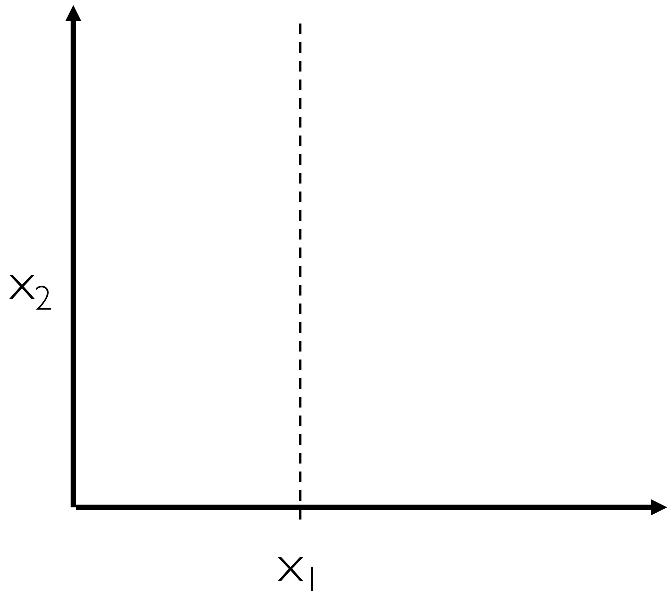


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

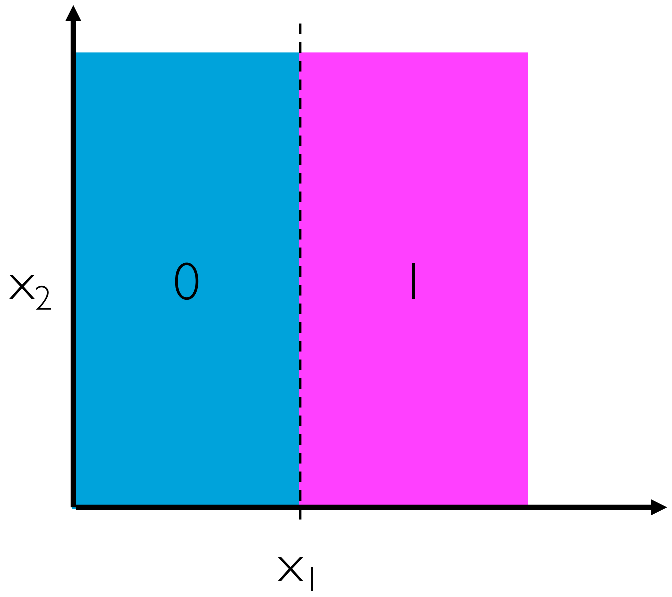
One branch for each value



Discrete vs. Continuous Inputs

Discrete Inputs
(e.g., boolean)

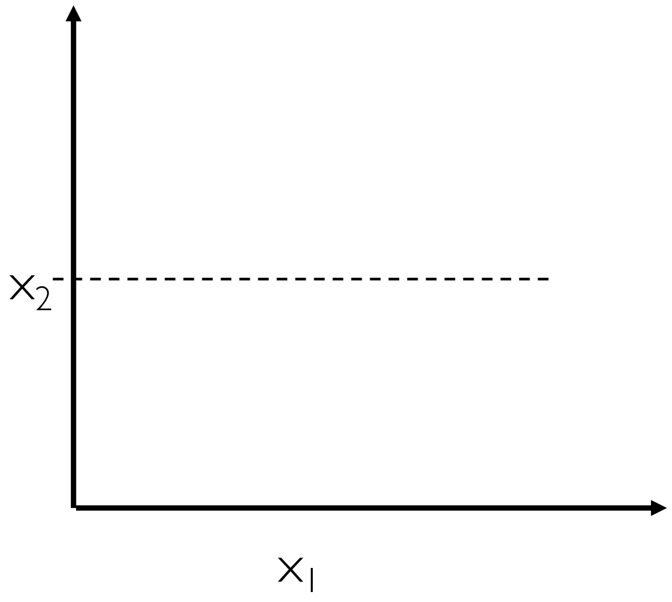
One branch for each value



Discrete vs. Continuous Inputs

Discrete Inputs
(e.g., boolean)

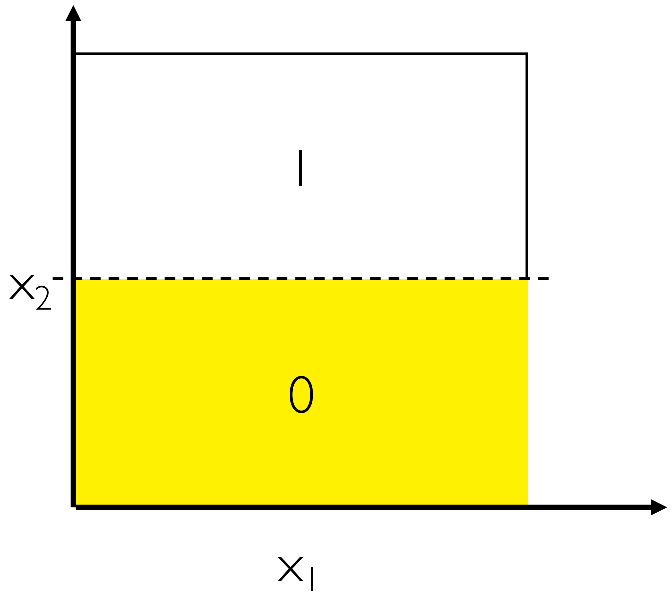
One branch for each value



Discrete vs. Continuous Inputs

Discrete Inputs
(e.g., boolean)

One branch for each value

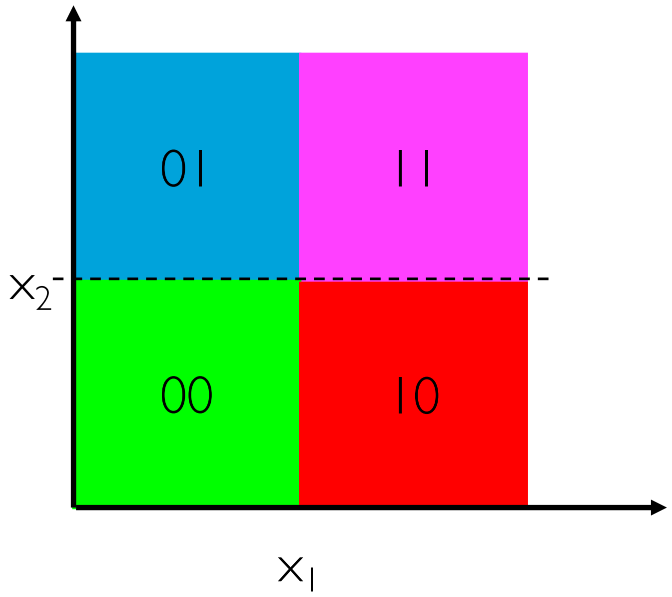


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

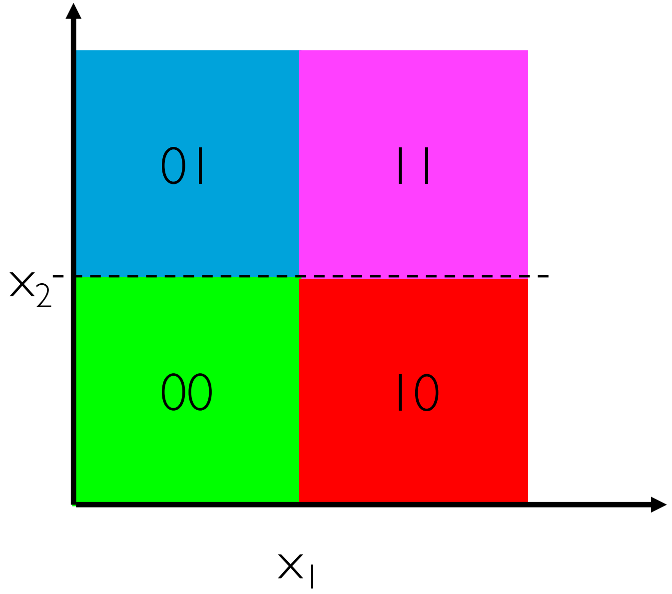


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

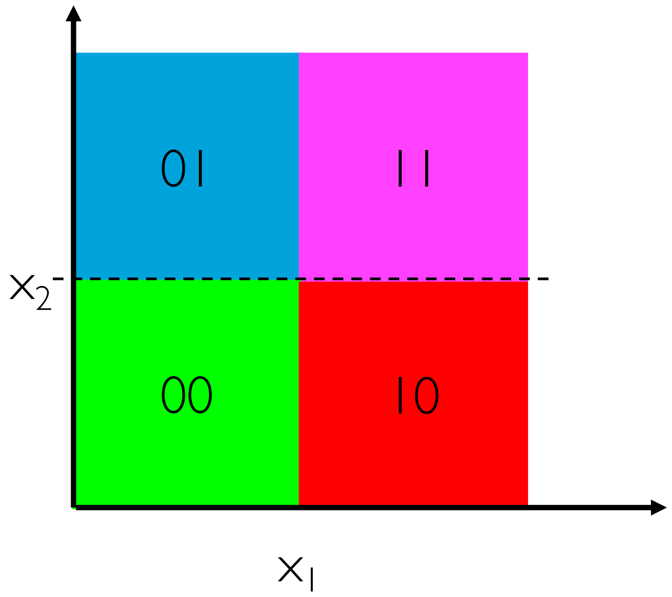


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

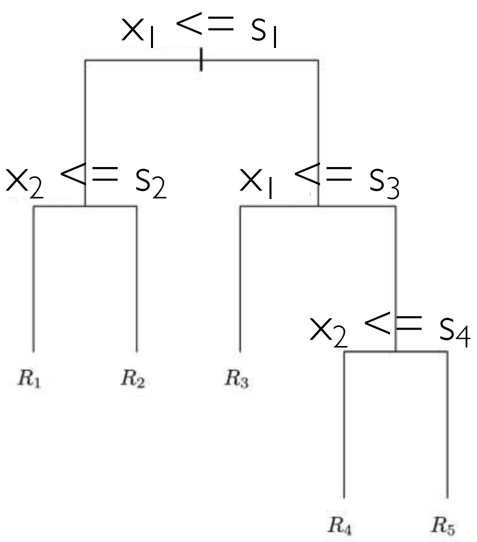


Continuous Inputs

For each attribute, find a split point s

Test x_j <= s and create 2 branches

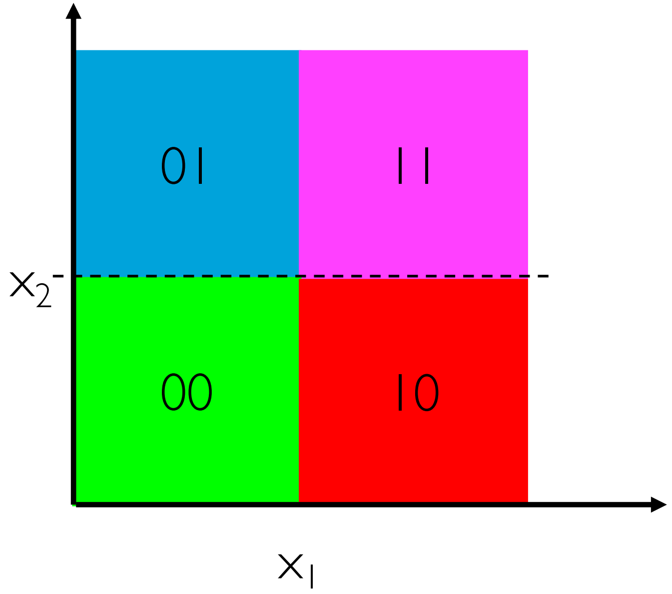
The same attribute may be further split in each subtree



Discrete vs. Continuous Inputs

Discrete Inputs
(e.g., boolean)

One branch for each value

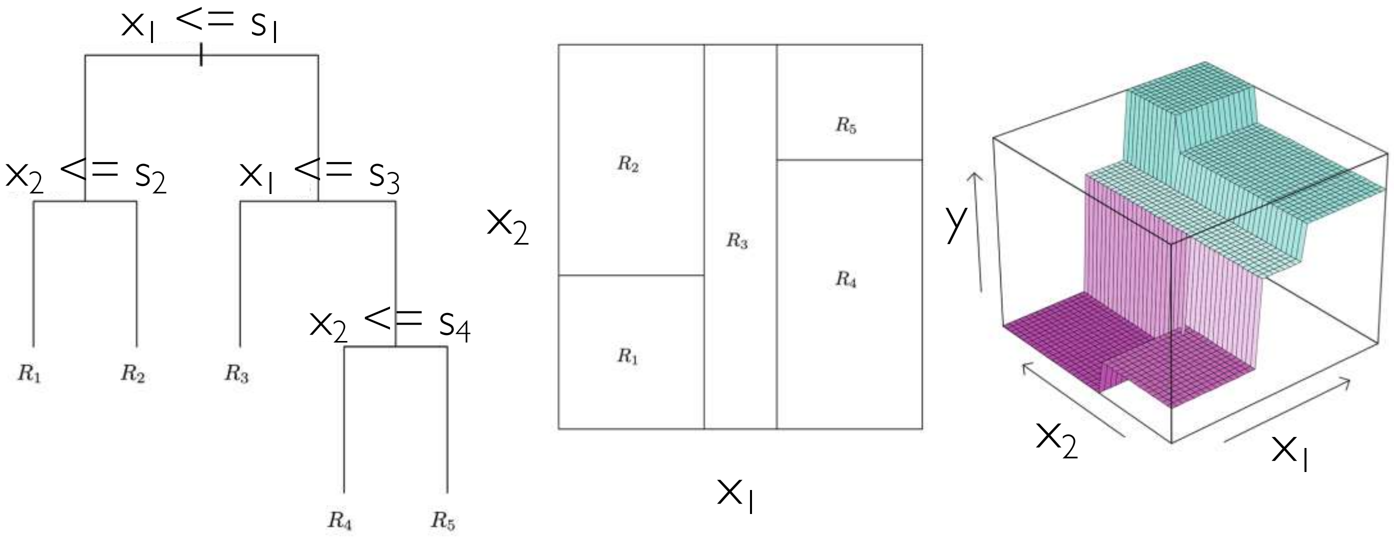


Continuous Inputs

For each attribute, find a split point s

Test x_j <= s and create 2 branches

The same attribute may be further split in each subtree

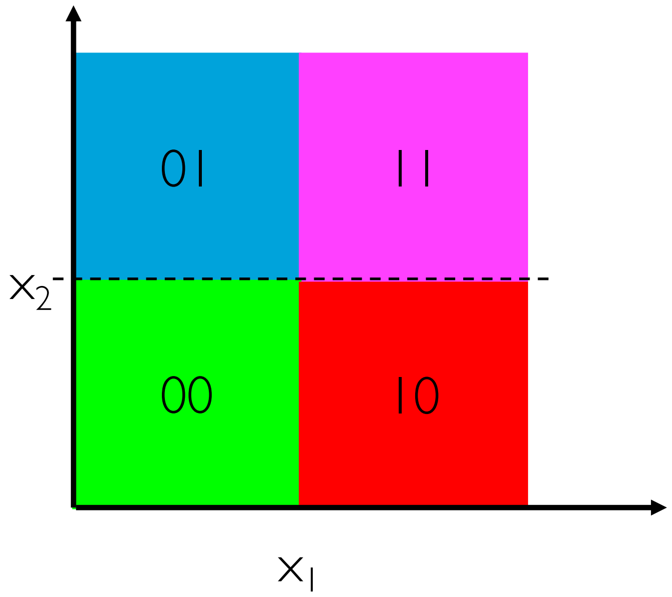


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

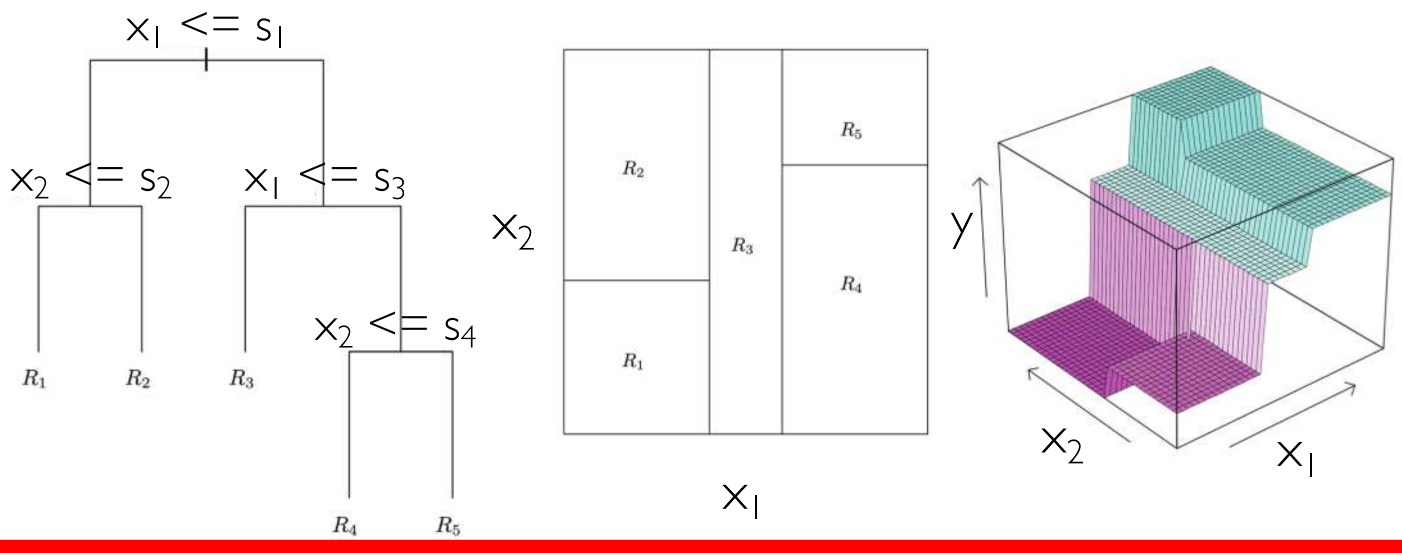


Continuous Inputs

For each attribute, find a split point s

Test x_j <= s and create 2 branches

The same attribute may be further split in each subtree



Expressiveness

Discrete Inputs/Discrete Outputs

Expressiveness

Discrete Inputs/Discrete Outputs

Decision Trees can express any function of the inputs

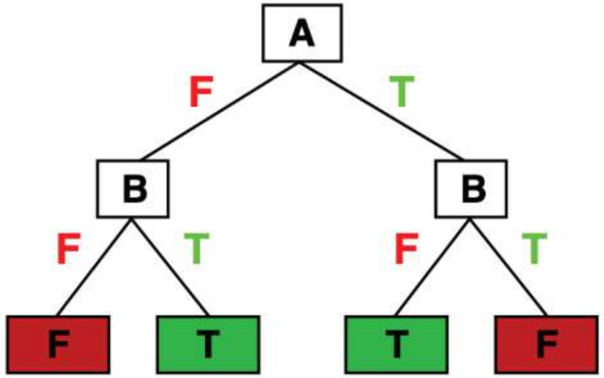
Expressiveness

Discrete Inputs/Discrete Outputs

Decision Trees can express any function of the inputs

Example: Boolean Functions

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Expressiveness

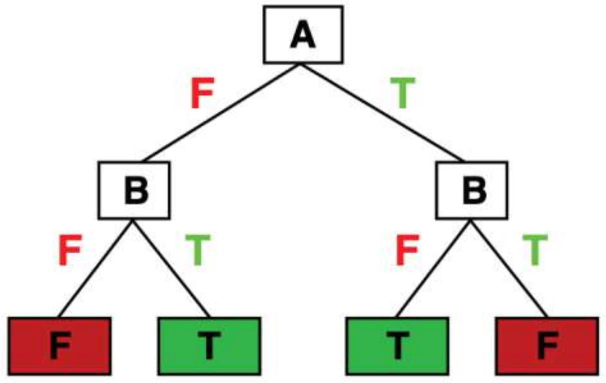
Discrete Inputs/Discrete Outputs

Decision Trees can express any function of the inputs

Example: Boolean Functions

Truth table

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



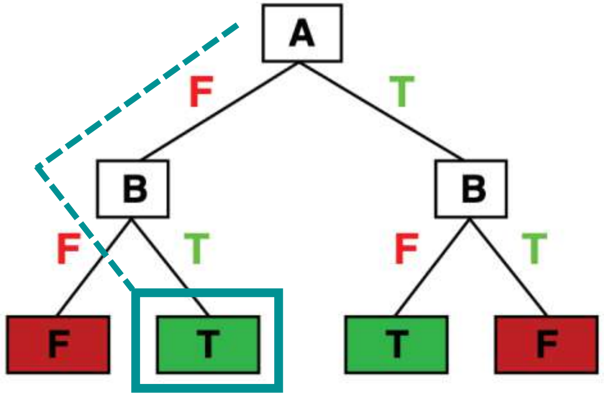
Expressiveness

Discrete Inputs/Discrete Outputs

Decision Trees can express any function of the inputs

<u>Example</u>: Boolean Functions

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Each row of the truth table maps to a root-to-leaf path on the tree

Expressiveness

Continuous Inputs/Continuous Outputs

Expressiveness

Continuous Inputs/Continuous Outputs

Decision Trees can approximate any function arbitrarily closely

Expressiveness

Continuous Inputs/Continuous Outputs

Decision Trees can approximate any function arbitrarily closely

Trivially, there exists a consistent decision tree for any training set

Expressiveness

Continuous Inputs/Continuous Outputs

Decision Trees can approximate any function arbitrarily closely

Trivially, there exists a consistent decision tree for any training set

Such a tree will have one dedicated root-to-leaf path for each training instance

Expressiveness

Continuous Inputs/Continuous Outputs

Decision Trees can approximate any function arbitrarily closely

Trivially, there exists a consistent decision tree for any training set

Such a tree will have one dedicated root-to-leaf path for each training instance

Of course, this tree clearly overfits the training data and it will not generalize to unseen examples (needs regularization)

Hypothesis Space

How many distinct decision trees can be built out of n boolean attributes?

Hypothesis Space

How many distinct decision trees can be built out of n boolean attributes?

This is equivalent to say how many boolean functions can be defined

Hypothesis Space

How many distinct decision trees can be built out of n boolean attributes?

This is equivalent to say how many boolean functions can be defined

Each boolean function of n boolean inputs is represented by a truth table

x_1	x_1	…	x_n	y
0	0	…	0	
1	0	…	0	
1	1	…	0	
…	…	…	…	
…	…	…	…	
…	…	…	…	
1	1	1	1	

Hypothesis Space

How many distinct decision trees can be built out of n boolean attributes?

This is equivalent to say how many boolean functions can be defined

Each boolean function of n boolean inputs is represented by a truth table

2^n rows

x_1	x_1	…	x_n	y
0	0	…	0	
1	0	…	0	
1	1	…	0	
…	…	…	…	
…	…	…	…	
…	…	…	…	
1	1	1	1	

Hypothesis Space

How many distinct decision trees can be built out of n boolean attributes?

This is equivalent to say how many boolean functions can be defined

Each boolean function of n boolean inputs is represented by a truth table

2^n rows

x_1	x_1	…	x_n	y
0	0	…	0	
1	0	…	0	
1	1	…	0	
…	…	…	…	
…	…	…	…	
…	…	…	…	
1	1	1	1	

For each input y = 0 or 1

Hypothesis Space

A possible boolean function is the one which will output all 0s

2^n rows

x_1	x_1	…	x_n	y
0	0	…	0	0
1	0	…	0	0
1	1	…	0	0
…	…	…	…	0
…	…	…	…	0
…	…	…	…	0
1	1	1	1	0

Hypothesis Space

Another possible boolean function is the one which will output all 1s

2^n rows

x_1	x_1	…	x_n	y
0	0	…	0	1
1	0	…	0	1
1	1	…	0	1
…	…	…	…	1
…	…	…	…	1
…	…	…	…	1
1	1	1	1	1

Hypothesis Space

Overall, there are $2^{(2^n)}$ possible boolean functions, therefore decision trees

Hypothesis Space

Overall, there are $2^{(2^n)}$ possible boolean functions, therefore decision trees

e.g., with just 6 boolean attributes, there are 18,446,744,073,709,551,616 trees

Hypothesis Space

Overall, there are $2^{(2^n)}$ possible boolean functions, therefore decision trees

e.g., with just 6 boolean attributes, there are 18,446,744,073,709,551,616 trees

The hypothesis space defined by decisions tree is very expressive because there are a lot of different functions it can represent

Hypothesis Space

Overall, there are $2^{(2^n)}$ possible boolean functions, therefore decision trees

e.g., with just 6 boolean attributes, there are 18,446,744,073,709,551,616 trees

The hypothesis space defined by decisions tree is very expressive because there are a lot of different functions it can represent

Larger hypothesis space means also it is generally harder for the learning algorithm to find the best hypothesis (larger space to explore)

Partitioning the Input Space: NP-Complete

- Unfortunately, minimizing RSS would require exploring all the possible partitions of the feature space into J boxes

Partitioning the Input Space: NP-Complete

- Unfortunately, minimizing RSS would require exploring all the possible partitions of the feature space into J boxes
- The problem is known to be NP-Complete, therefore computationally unfeasible

Partitioning the Input Space: NP-Complete

- Unfortunately, minimizing RSS would require exploring all the possible partitions of the feature space into J boxes
- The problem is known to be NP-Complete, therefore computationally unfeasible

⇩

Solution

Top-Down greedy heuristic Recursive Binary Splitting

Recursive Binary Splitting (RBS)

- Initially, all the observations belong to the root of the tree

Recursive Binary Splitting (RBS)

- Initially, all the observations belong to the root of the tree
- Split the input feature space into 2 subtrees (i.e., left and right)

Recursive Binary Splitting (RBS)

- Initially, all the observations belong to the root of the tree
- Split the input feature space into 2 subtrees (i.e., left and right)
- Recursively repeat the step above on both subtrees

Recursive Binary Splitting (RBS)

- Initially, all the observations belong to the root of the tree
- Split the input feature space into 2 subtrees (i.e., left and right)
- Recursively repeat the step above on both subtrees

top-down

Recursive Binary Splitting (RBS)

- Initially, all the observations belong to the root of the tree
- Split the input feature space into 2 subtrees (i.e., left and right)
- Recursively repeat the step above on both subtrees

top-down

- Greedy strategy:
  - At each step, the best "local" split is made
  - Looking ahead might result in a different split, which leads to a better tree

How to Choose the Split?

- Intuitively, we need to select the feature f and a splitting point s (of f)

How to Choose the Split?

- Intuitively, we need to select the feature *f* and a splitting point s (of *f*)
- Such a splitting will lead to the partitioning of the feature space into the following regions:

$$\boxed{R_{\text{left}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} \leq s\} \quad R_{\text{right}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} > s\}}$$

How to Choose the Split?

- Intuitively, we need to select the feature *f* and a splitting point s (of *f*)
- Such a splitting will lead to the partitioning of the feature space into the following regions:

$$\boxed{R_{\text{left}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} \leq s\} \quad R_{\text{right}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} > s\}}$$

- Among all the possible splittings, we select the one which reduces RSS **the most**

How to Choose the Split?

- Intuitively, we need to select the feature *f* and a splitting point s (of *f*)
- Such a splitting will lead to the partitioning of the feature space into the following regions:

$$\boxed{R_{\text{left}}(f,s) = \{\mathbf{x}_i \mid x_{i,f} \le s\} \quad R_{\text{right}}(f,s) = \{\mathbf{x}_i \mid x_{i,f} > s\}}$$

- Among all the possible splittings, we select the one which reduces RSS **the most**

$$\{x_{i,f} \le s\}$$

is the region of the feature space in which the *f*-th feature takes on values less than or equal to *s*

How to Choose the Split?

- For each feature $f = 1, \ldots, d$ we look at every possible splitting value s (i.e., all the values we observed in the training set for the feature f)

How to Choose the Split?

- For each feature $f = 1, \ldots, d$ we look at every possible splitting value s (i.e., all the values we observed in the training set for the feature f)
- For each pair of feature and splitting value (f, s), we obtain 2 regions:

$$R_{\text{left}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} \leq s\} \quad R_{\text{right}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} > s\}$$

How to Choose the Split?

- For each feature $f = 1, \ldots, d$ we look at every possible splitting value s (i.e., all the values we observed in the training set for the feature f)
- For each pair of feature and splitting value (f, s), we obtain 2 regions:

$$R_{\text{left}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} \leq s\} \quad R_{\text{right}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} > s\}$$

Goal: find the pair (f, s) which minimizes the following

$$\sum_{i:\ \mathbf{x}_i \in R_{\text{left}}(f,s)} (y_i - \hat{y}_{R_{\text{left}}})^2 + \sum_{i:\ \mathbf{x}_i \in R_{\text{right}}(f,s)} (y_i - \hat{y}_{R_{\text{right}}})^2$$

Growing the Tree

- Finding the pair (f, s) which minimizes the quantity below can be done "easily", especially when the number of features d is not too large

$$\sum_{i:\ \mathbf{x}_i \in R_{\text{left}}(f,s)} (y_i - \hat{y}_{R_{\text{left}}})^2 + \sum_{i:\ \mathbf{x}_i \in R_{\text{right}}(f,s)} (y_i - \hat{y}_{R_{\text{right}}})^2$$

Growing the Tree

- Finding the pair (*f*, *s*) which minimizes the quantity below can be done "easily", especially when the number of features d is not too large

$$\sum_{i:\ \mathbf{x}_i \in R_{\text{left}}(f,s)} (y_i - \hat{y}_{R_{\text{left}}})^2 + \sum_{i:\ \mathbf{x}_i \in R_{\text{right}}(f,s)} (y_i - \hat{y}_{R_{\text{right}}})^2$$

- At the next step, the same splitting strategy applies yet considering only the data falling into the previously identified regions

Growing the Tree

- Finding the pair (*f*, *s*) which minimizes the quantity below can be done "easily", especially when the number of features d is not too large

$$\sum_{i:\ \mathbf{x}_i \in R_{\text{left}}(f,s)} (y_i - \hat{y}_{R_{\text{left}}})^2 + \sum_{i:\ \mathbf{x}_i \in R_{\text{right}}(f,s)} (y_i - \hat{y}_{R_{\text{right}}})^2$$

- At the next step, the same splitting strategy applies yet considering only the data falling into the previously identified regions
- Each time, we reduce the RSS

When Do We Stop?

- We need a stopping criterion otherwise the tree will grow until each training instance falls into a leaf node

When Do We Stop?

- We need a stopping criterion otherwise the tree will grow until each training instance falls into a leaf node
- Clearly, when that happens the RSS is minimum (in fact, it is 0!)
  - That would correspond to an overfitted tree

When Do We Stop?

- We need a stopping criterion otherwise the tree will grow until each training instance falls into a leaf node
- Clearly, when that happens the RSS is minimum (in fact, it is 0!)
  - That would correspond to an overfitted tree
- Possible stopping criteria (tree grows until):
  - no region contains more than N observations
  - max depth of the tree is D
  - RSS is reduced by at least a threshold value t

Test Time Predictions

- Once the tree is built using training instances, we will have J regions

Test Time Predictions

- Once the tree is built using training instances, we will have J regions
- In each of them, we have a fraction of the initial training examples

Test Time Predictions

- Once the tree is built using training instances, we will have J regions
- In each of them, we have a fraction of the initial training examples
- We can compute the mean of each region as the mean of the responses of the instances falling in that region

Test Time Predictions

- Once the tree is built using training instances, we will have J regions
- In each of them, we have a fraction of the initial training examples
- We can compute the **mean** of each region as the mean of the responses of the instances falling in that region
- At test time, an unseen instance follows a root-to-leaf path on the tree and ends up into a region R_j

Test Time Predictions

- Once the tree is built using training instances, we will have J regions
- In each of them, we have a fraction of the initial training examples
- We can compute the mean of each region as the mean of the responses of the instances falling in that region
- At test time, an unseen instance follows a root-to-leaf path on the tree and ends up into a region R_j
- The prediction for that test instance will be the mean of the region R_j

Classification Trees

- Very similar to a regression tree

Classification Trees

- Very similar to a regression tree
- Used to predict a categorical response rather than a numerical one

Classification Trees

- Very similar to a regression tree
- Used to predict a categorical response rather than a numerical one
- Tree building is still based on Recursive Binary Splitting algorithm but RSS cannot be used as a criterion for splitting nodes

Classification Trees

- Very similar to a regression tree
- Used to predict a categorical response rather than a numerical one
- Tree building is still based on Recursive Binary Splitting algorithm but RSS cannot be used as a criterion for splitting nodes
- A natural alternative to RSS minimization is to minimize the "impurity"

Classification Trees

- Very similar to a regression tree
- Used to predict a categorical response rather than a numerical one
- Tree building is still based on Recursive Binary Splitting algorithm but RSS cannot be used as a criterion for splitting nodes
- A natural alternative to RSS minimization is to minimize the "impurity"
- The predicted label of a test instance is the most frequent label (mode) of the instances belonging to the region where it falls

The "Impurity" of a Node

- A node containing instances all with the same label is perfectly pure

The "Impurity" of a Node

- A node containing instances all with the same label is perfectly pure
- We would like to grow a tree whose nodes are as purest as possible

The "Impurity" of a Node

- A node containing instances all with the same label is perfectly pure
- We would like to grow a tree whose nodes are as purest as possible
- Several different measures to represent this notion of node "impurity":
  - Classification Error Rate
  - Gini Index
  - Entropy

The "Impurity" of a Node

- A node containing instances all with the same label is perfectly pure
- We would like to grow a tree whose nodes are as purest as possible
- Several different measures to represent this notion of node "impurity":
  - Classification Error Rate
  - Gini Index
  - Entropy
- It is often convenient to refer to the information gain of a split

Classification Trees: Notation

Subset of training instances falling into region R

$$\mathcal{D}_R = \mathcal{D} \cap R = \{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid \mathbf{x}_i \in R\}$$

Classification Trees: Notation

Subset of training instances falling into region R

$$\mathcal{D}_R = \mathcal{D} \cap R = \{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid \mathbf{x}_i \in R\}$$

(*f*, *s*)-split

$$R_{\text{left}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} \leq s\} \quad R_{\text{right}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} > s\}$$

Classification Trees: Notation

Subset of training instances falling into region R

$$\mathcal{D}_R = \mathcal{D} \cap R = \{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid \mathbf{x}_i \in R\}$$

(*f*, *s*)-split

$$R_{\text{left}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} \leq s\} \quad R_{\text{right}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} > s\}$$

impurity of a region

$$I(\mathcal{D}_R)$$

Classification Trees: Notation

Subset of training instances falling into region R

$$\mathcal{D}_R = \mathcal{D} \cap R = \{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid \mathbf{x}_i \in R\}$$

(*f*, *s*)-split

$$R_{\text{left}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} \leq s\} \quad R_{\text{right}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} > s\}$$

impurity of a region

$$I(\mathcal{D}_R)$$

information gain obtained with an (*f*, *s*)-split

$$IG(\mathcal{D}_R, f, s)$$

Classification Trees: Notation

Subset of training instances falling into region R

$$\mathcal{D}_R = \mathcal{D} \cap R = \{(\mathbf{x}_i, y_i) \in \mathcal{D} \mid \mathbf{x}_i \in R\}$$

(f, s)-split

$$R_{\text{left}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} \leq s\} \quad R_{\text{right}}(f, s) = \{\mathbf{x}_i \mid x_{i,f} > s\}$$

impurity of a region

$$I(\mathcal{D}_R)$$

information gain obtained with an (f, s)-split

$$IG(\mathcal{D}_R, f, s)$$

Maximize Information Gain

At each step, the best (f, s)-split is found so as to minimize the children nodes' impurity

Maximize Information Gain

At each step, the best *(f, s)*-split is found so as to minimize the children nodes' impurity

This corresponds to select the *(f, s)*-split that maximizes the information gain

Maximize Information Gain

At each step, the best (*f*, *s*)-split is found so as to minimize the children nodes' impurity

This corresponds to select the (*f*, *s*)-split that maximizes the information gain

$$IG(\mathcal{D}_R, f, s) = \underbrace{I(\mathcal{D}_R)}_{\text{parent node's impurity}} - \underbrace{\left[\frac{|\mathcal{D}_{R_{\text{left}}(f,s)}|}{|\mathcal{D}_R|} I(\mathcal{D}_{R_{\text{left}}(f,s)}) + \frac{|\mathcal{D}_{R_{\text{right}}(f,s)}|}{|\mathcal{D}_R|} I(\mathcal{D}_{R_{\text{right}}(f,s)})\right]}_{\text{children nodes' impurity}}$$

Node Impurity: Classification Error Rate

- Classification error rate is the fraction of the training observations in that region that are not labeled with the most common class

Node Impurity: Classification Error Rate

- Classification error rate is the fraction of the training observations in that region that are not labeled with the most common class

$$I_E(\mathcal{D}_{Rj}) = 1 - \max_k\{\hat{p}_{jk}\}$$

Node Impurity: Classification Error Rate

- Classification error rate is the fraction of the training observations in that region that are not labeled with the most common class

$$I_E(\mathcal{D}_{Rj}) = 1 - \max_k\{\boxed{\hat{p}_{jk}}\}$$

proportion of training observations in the *j*-th region that are from the *k*-th class

Node Impurity: Classification Error Rate

- Classification error rate is the fraction of the training observations in that region that are not labeled with the most common class

$$I_E(\mathcal{D}_{Rj}) = 1 - \max_k\{\boxed{\hat{p}_{jk}}\}$$

proportion of training observations in the *j*-th region that are from the *k*-th class

- Although this is the most intuitive notion of impurity we will see how this may not be a great choice

Node Impurity: Classification Error Rate

- Classification error rate is the fraction of the training observations in that region that are not labeled with the most common class

$$I_E(\mathcal{D}_{Rj}) = 1 - \max_k\{\boxed{\hat{p}_{jk}}\}$$

proportion of training observations in the *j*-th region that are from the *k*-th class

- Although this is the most intuitive notion of impurity we will see how this may not be a great choice

- Other 2 measures are preferable: Gini Index and Entropy

Node Impurity: Gini Index

- Measures the variance across the K classes

$$I_G(\mathcal{D}_{Rj}) = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$$

Node Impurity: Gini Index

- Measures the variance across the K classes

$$I_G(\mathcal{D}_{Rj}) = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$$

- A small value of Gini is obtained whenever all the proportions are either close to 1 or to 0

Node Impurity: Gini Index

- Measures the variance across the K classes

$$I_G(\mathcal{D}_{Rj}) = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$$

- A small value of Gini is obtained whenever all the proportions are either close to 1 or to 0
- A small value indicates that a node contains predominantly observations from a single class

Node Impurity: Entropy

- Alternative, yet similar to Gini

$$I_H(\mathcal{D}_{Rj}) = -\sum_{k=1}^{K} \hat{p}_{jk} \log_2(\hat{p}_{jk})$$

Node Impurity: Entropy

- Alternative, yet similar to Gini

$$I_H(\mathcal{D}_{Rj}) = -\sum_{k=1}^{K} \hat{p}_{jk} \log_2(\hat{p}_{jk})$$

- It ranges between [0, +infinity]

Node Impurity: Entropy

- Alternative, yet similar to Gini

$$I_H(\mathcal{D}_{Rj}) = -\sum_{k=1}^{K} \hat{p}_{jk} \log_2(\hat{p}_{jk})$$

- It ranges between [0, +infinity]
- Like Gini, a small value of entropy is obtained whenever all the proportions are either close to 1 or to 0

Node Impurity: Entropy

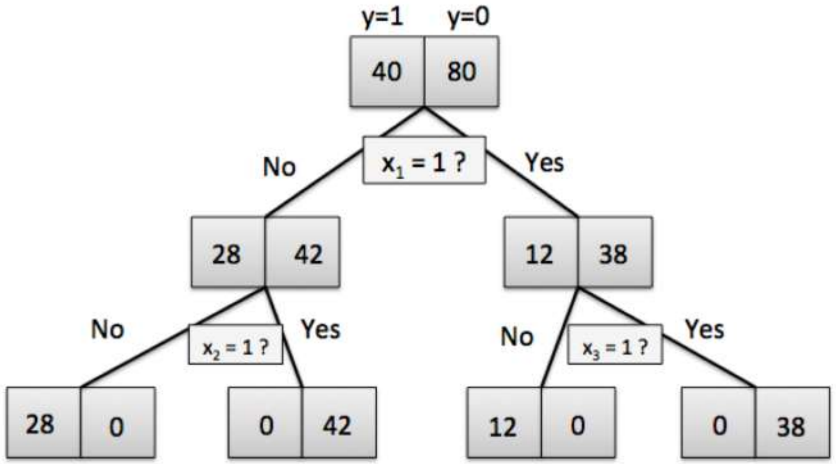
- Alternative, yet similar to Gini

$$I_H(\mathcal{D}_{Rj}) = -\sum_{k=1}^{K} \hat{p}_{jk} \log_2(\hat{p}_{jk})$$

- It ranges between [0, +infinity]
- Like Gini, a small value of entropy is obtained whenever all the proportions are either close to 1 or to 0
- In practice, both entropy and Gini can be used to grow a tree

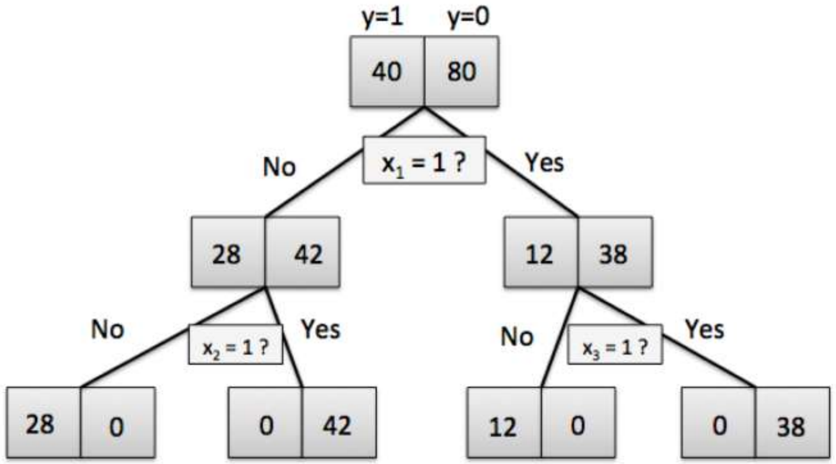
Classification Error Rate vs. Entropy (Gini)

Consider this decision tree which perfectly separates positive (y=1) from negative (y=0) samples using 3 splits on 3 binary features x_1, x_2, x_3



Classification Error Rate vs. Entropy (Gini)

Consider this decision tree which perfectly separates positive ($y=1$) from negative ($y=0$) samples using 3 splits on 3 binary features x_1, x_2, x_3

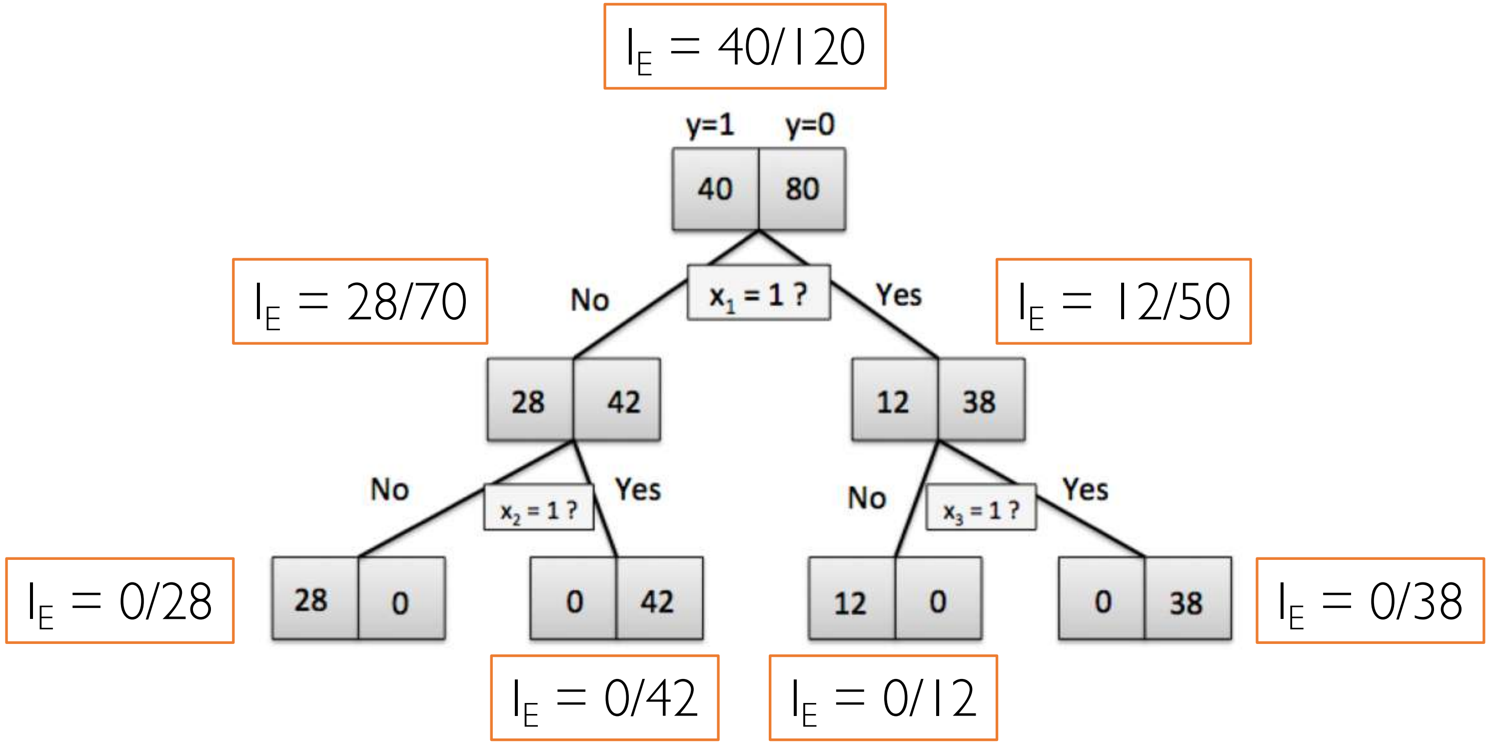


Question

Would we be able to learn the tree above using classification error rate as splitting criterion?

Classification Error Rate vs. Entropy (Gini)

Node impurity using I_E

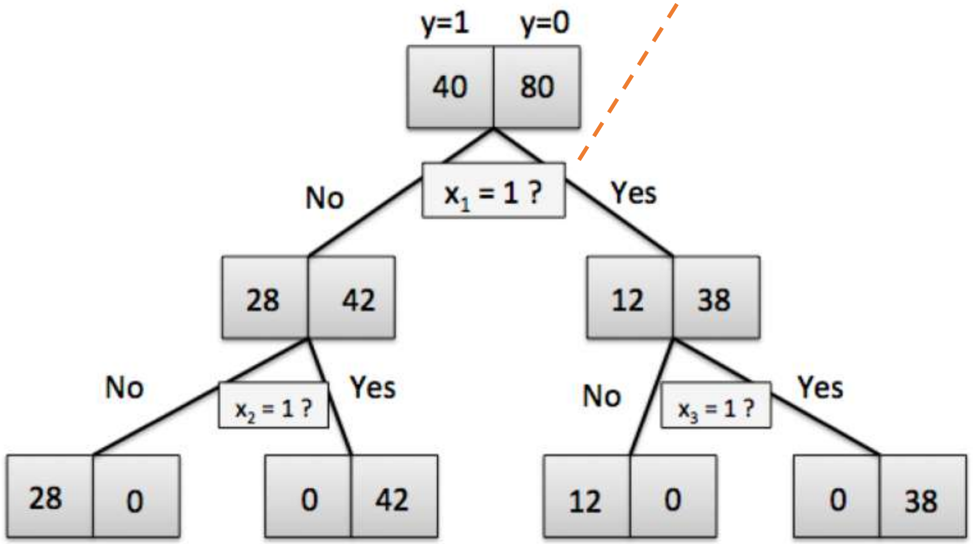


source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)

Compute IG of the first split using I_E

$I_E = 40/120$



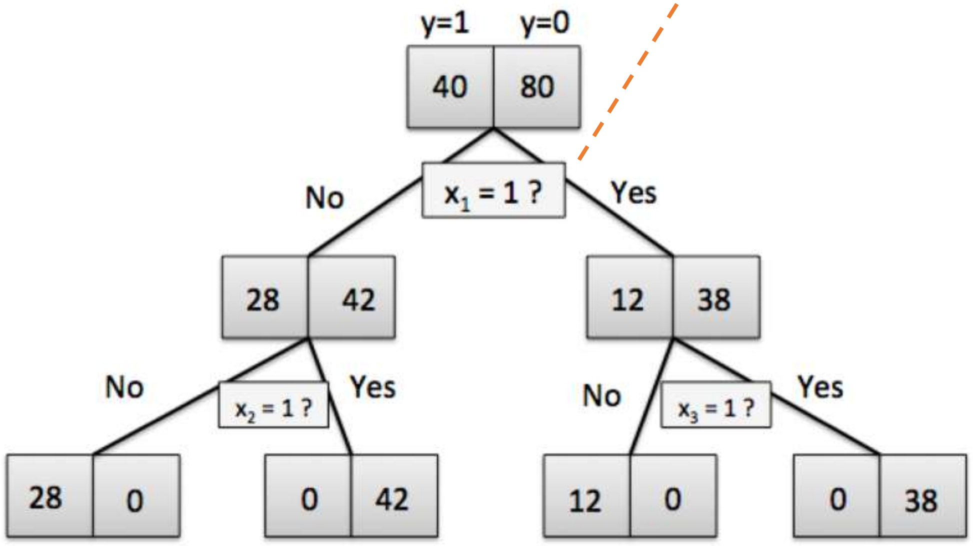
source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)

Compute IG of the first split using I_E

$40/120 - [(70/120 * 28/70) + (50/120 * 12/50)]$

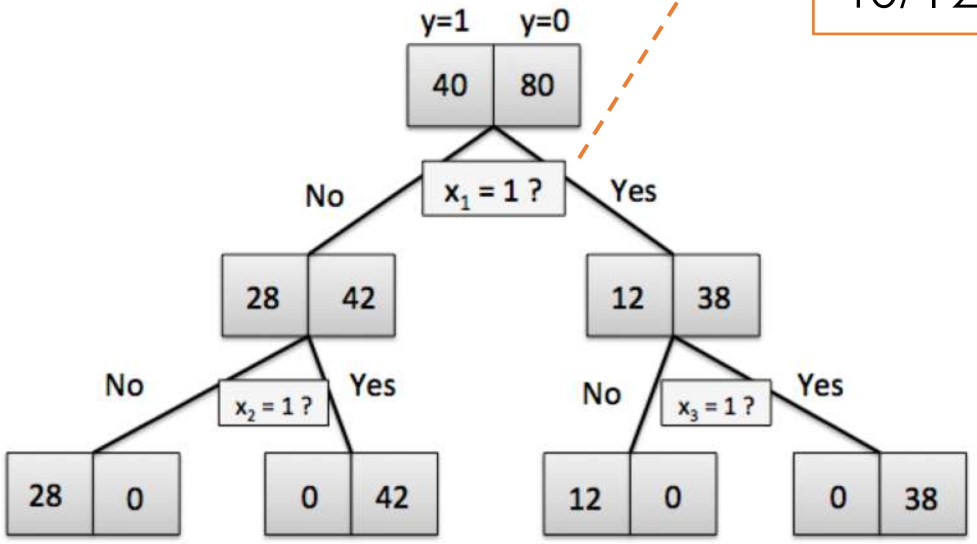
$I_E = 40/120$



source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)

$I_E = 40/120$



Compute IG of the first split using I_E

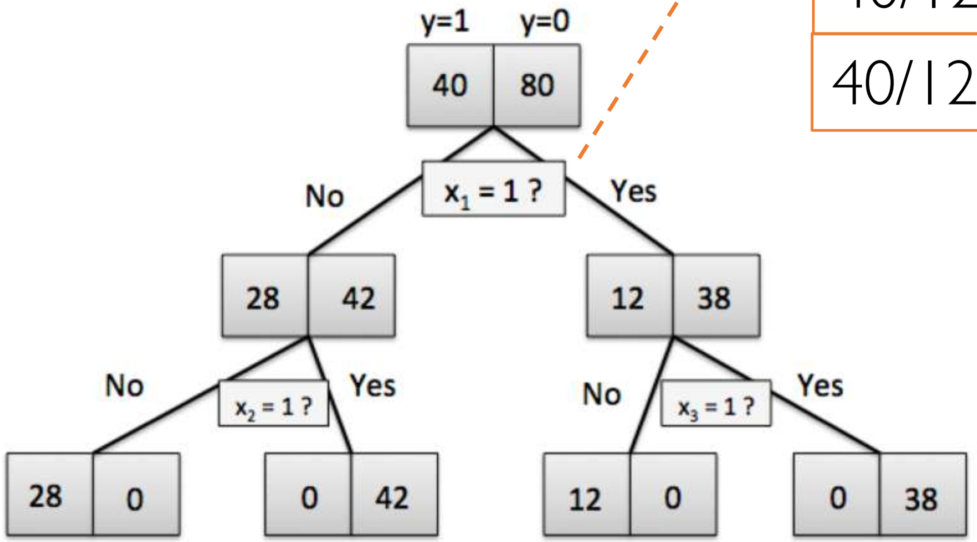
$40/120 - [(70/120 * 28/70) + (50/120 * 12/50)]$

$40/120 - [28/10 + 12/120]$

source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)

$I_E = 40/120$



Compute IG of the first split using I_E

$40/120 - [(70/120 * 28/70) + (50/120 * 12/50)]$

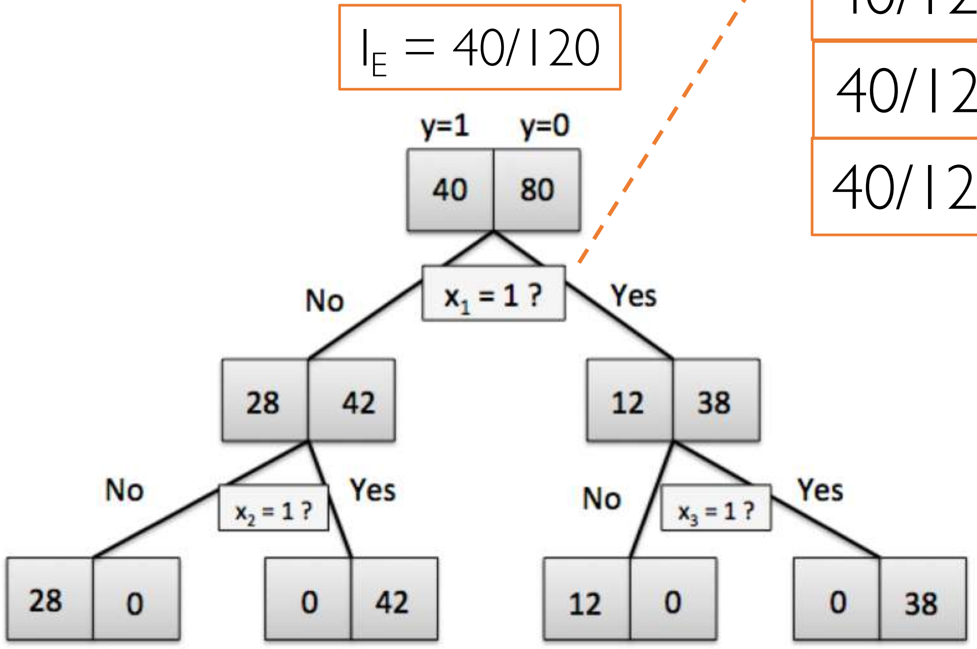
$40/120 - [28/10 + 12/120]$

$40/120 - 40/120 = 0$

IG after the first split is exactly 0

source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)



Compute IG of the first split using I_E

$40/120 - [(70/120 * 28/70) + (50/120 * 12/50)]$

$40/120 - [28/10 + 12/120]$

$40/120 - 40/120 = 0$

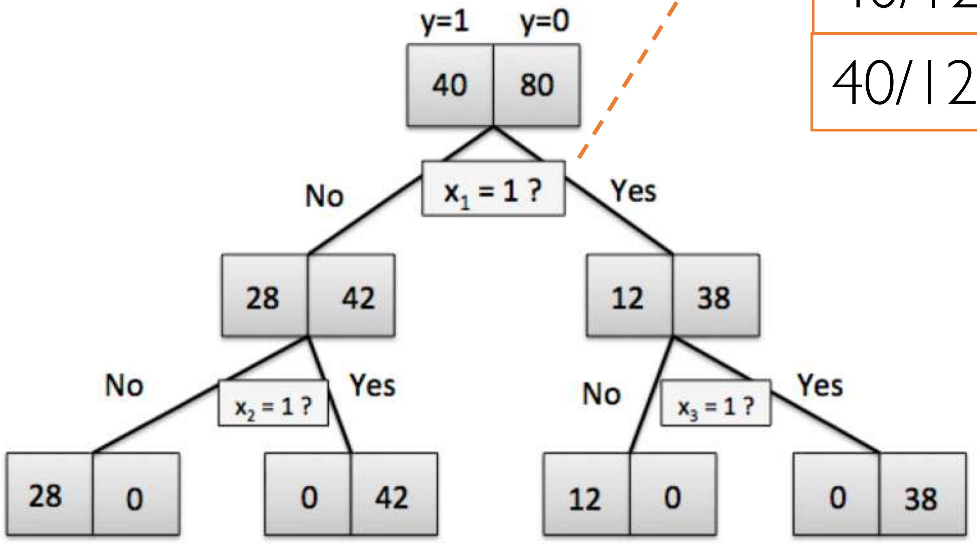
IG after the first split is exactly 0

the sum of I_E of the 2 child nodes is equal to that of the parent (40/120)

source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)

I_E = 40/120



Compute IG of the first split using I_E

40/120 - [(70/120 * 28/70) + (50/120 * 12/50)]

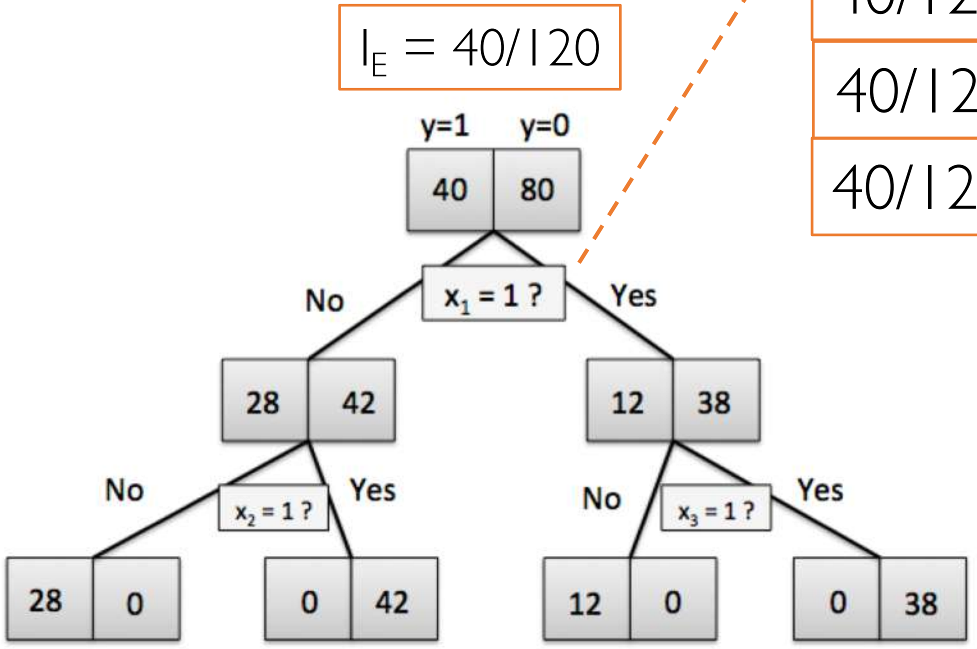
40/120 - [28/10 + 12/120]

40/120 - 40/120 = 0

Splitting root gives no IG improvement in terms of IE

source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)



Compute IG of the first split using I_E

$40/120 - [(70/120 * 28/70) + (50/120 * 12/50)]$

$40/120 - [28/10 + 12/120]$

$40/120 - 40/120 = 0$

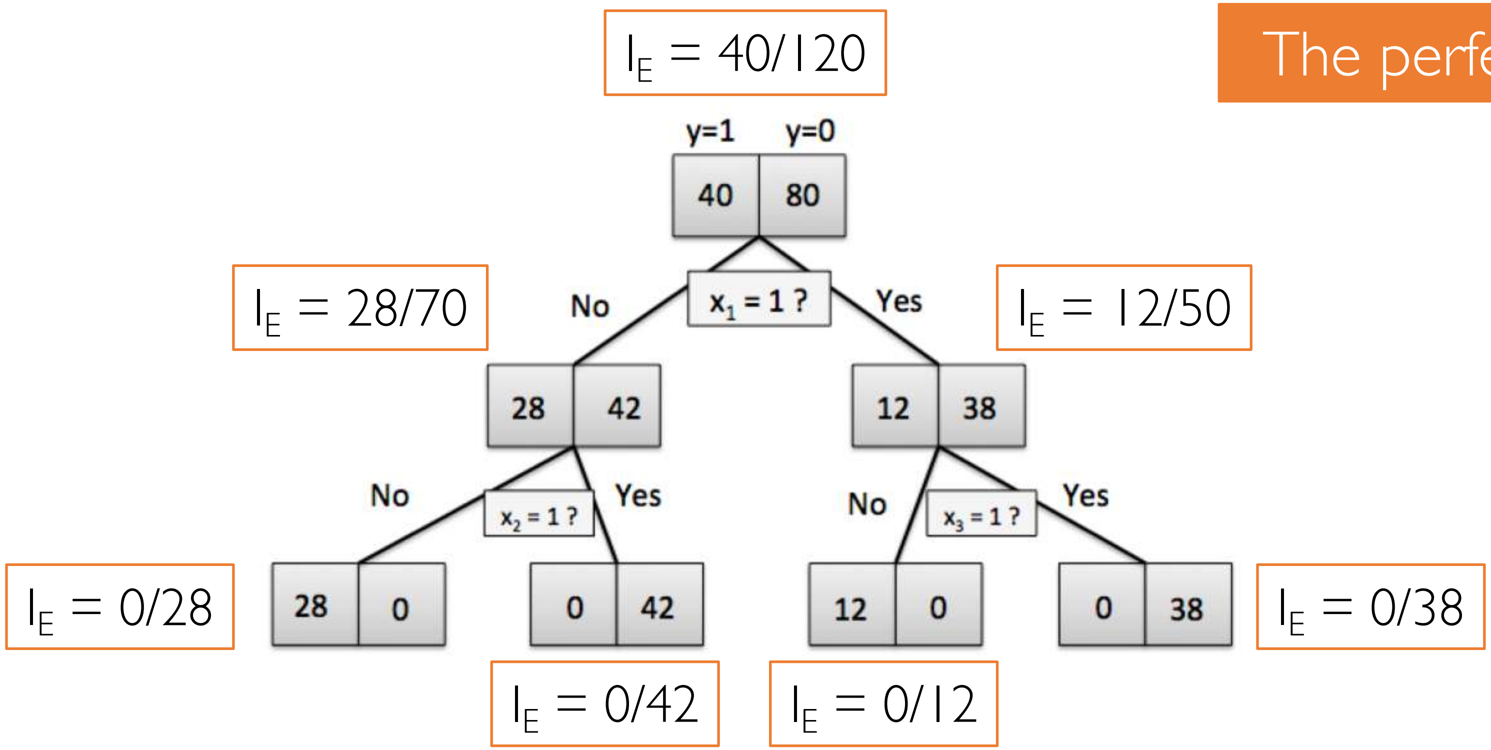
Splitting root gives no IG improvement in terms of IE

The tree learning algorithm would stop at this point

source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)

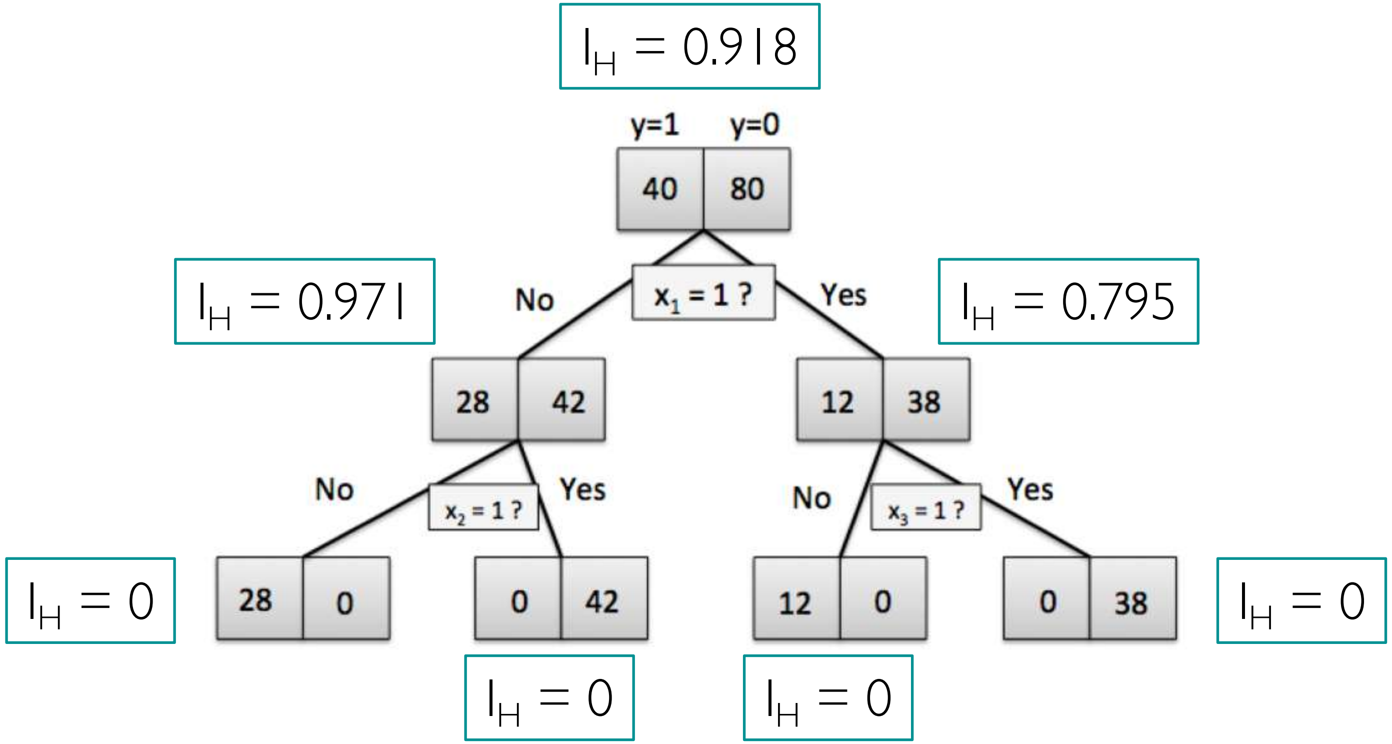
The perfect tree cannot be built using I_E



source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)

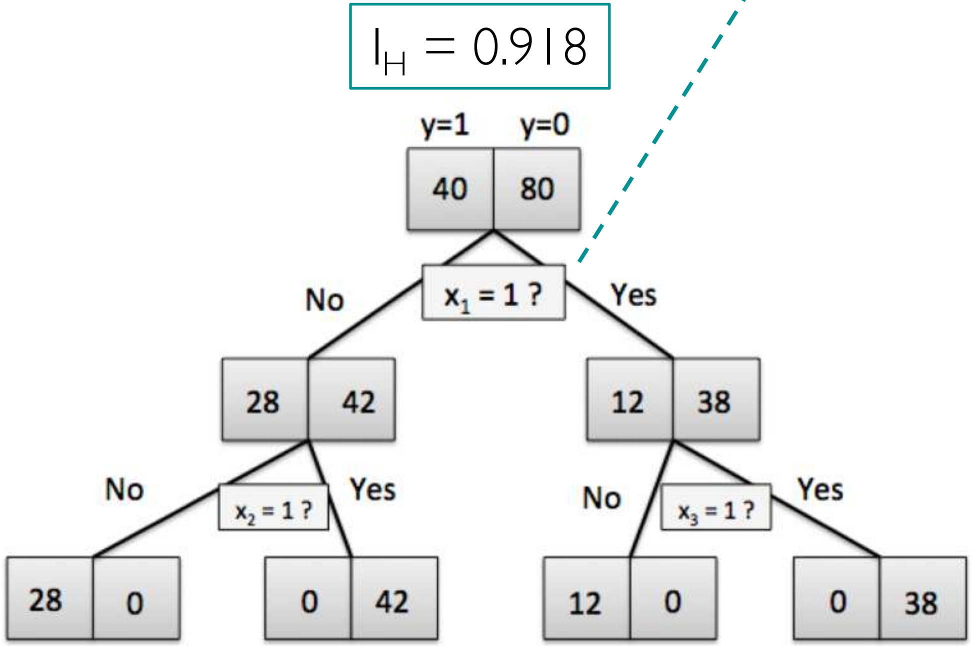
Node impurity using I_H



source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

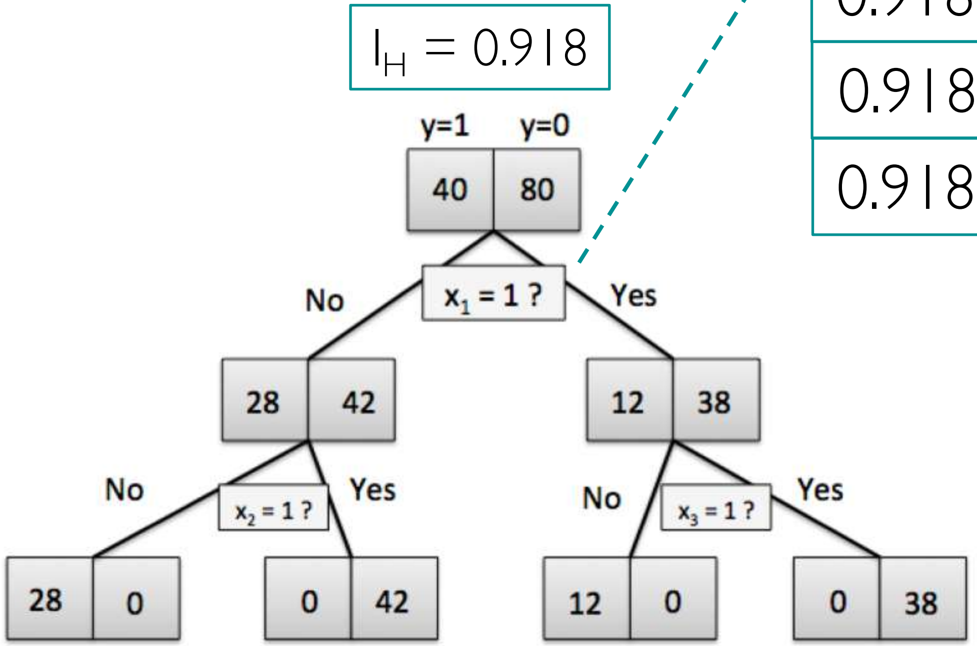
Classification Error Rate vs. Entropy (Gini)

Compute IG of the first split using I_H



source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)



Compute IG of the first split using I_H

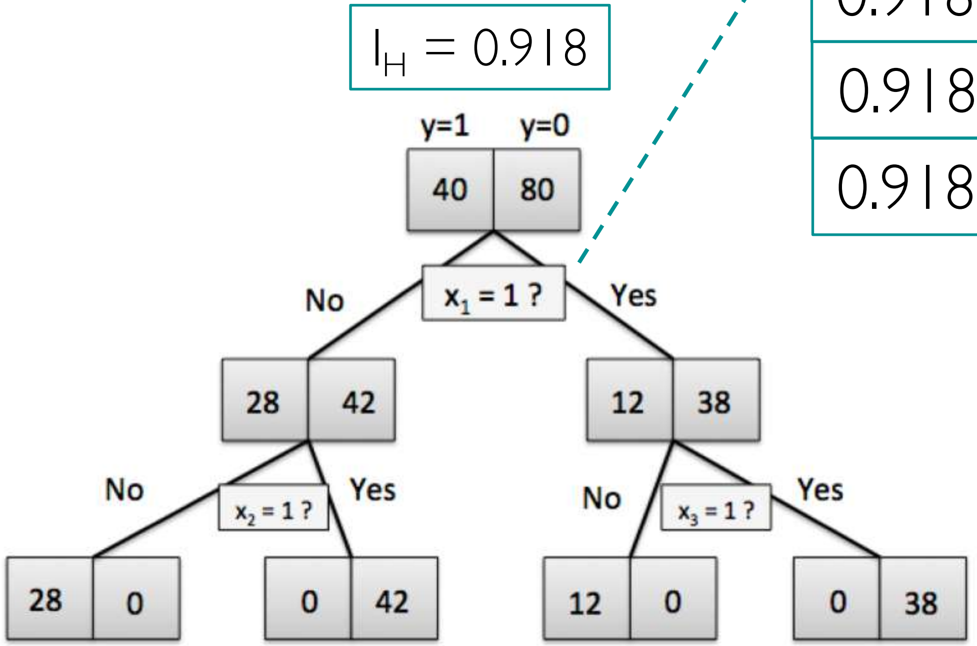
$0.918 - [(70/120 * 0.971) + (50/120 * 0.795)]$

$0.918 - [0.566 + 0.331]$

$0.918 - 0.897 = 0.021$

source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)



Compute IG of the first split using I_H

$0.918 - [(70/120 * 0.971) + (50/120 * 0.795)]$

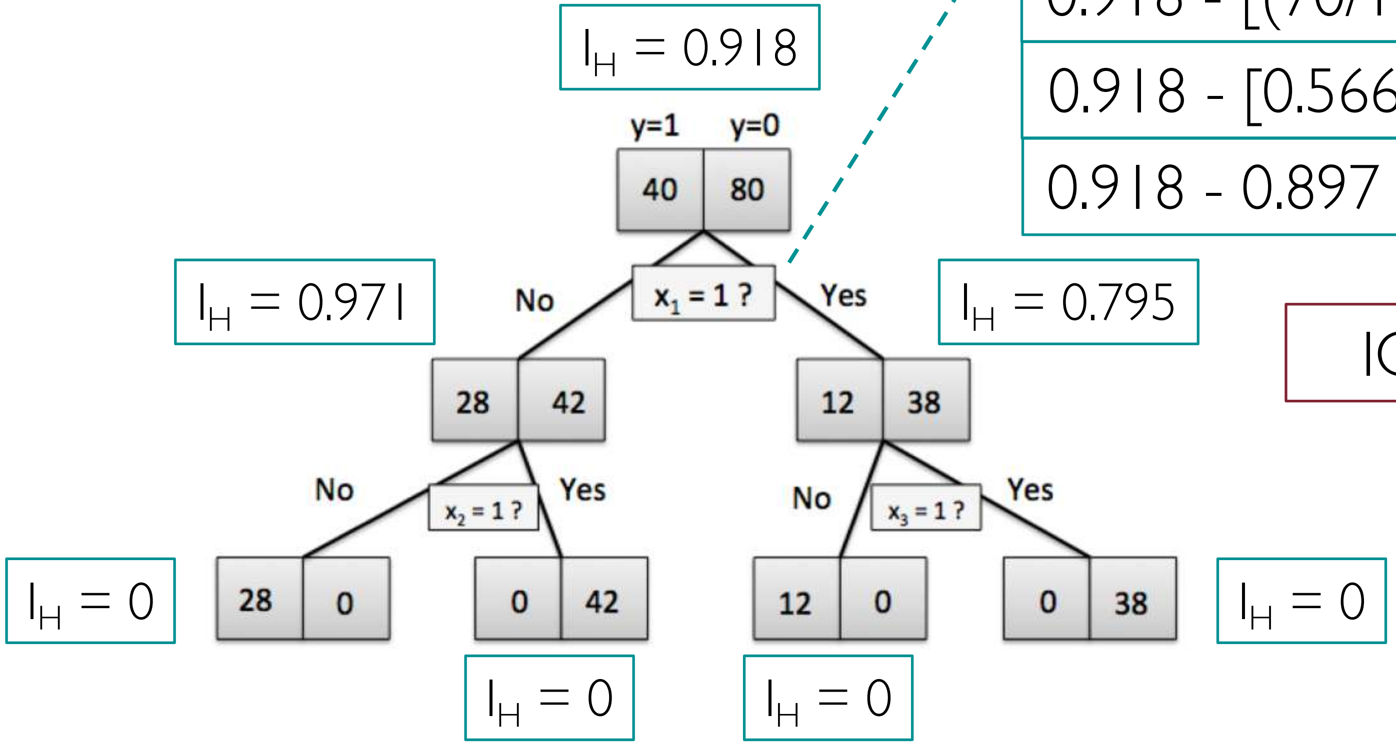
$0.918 - [0.566 + 0.331]$

$0.918 - 0.897 = 0.021$

IG after the first split is greater than 0

source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)



Compute IG of the first split using I_H

$0.918 - [(70/120 * 0.971) + (50/120 * 0.795)]$

$0.918 - [0.566 + 0.331]$

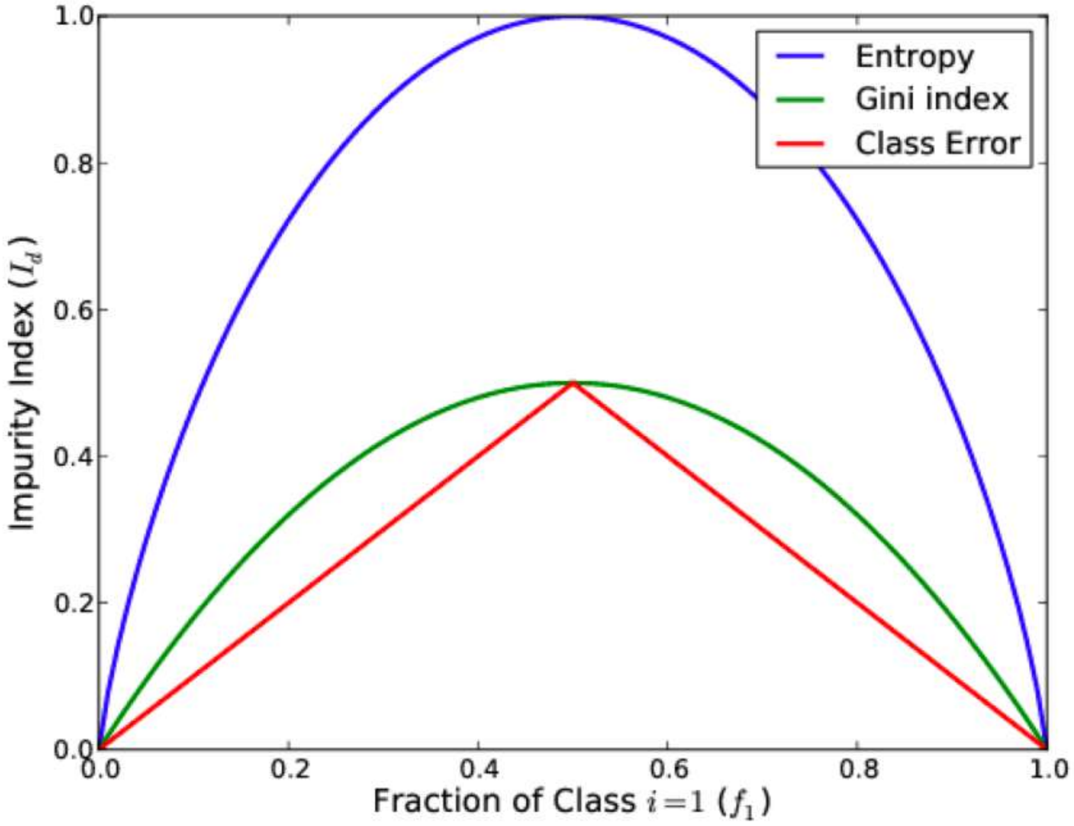
$0.918 - 0.897 = 0.021$

IG after the first split is greater than 0

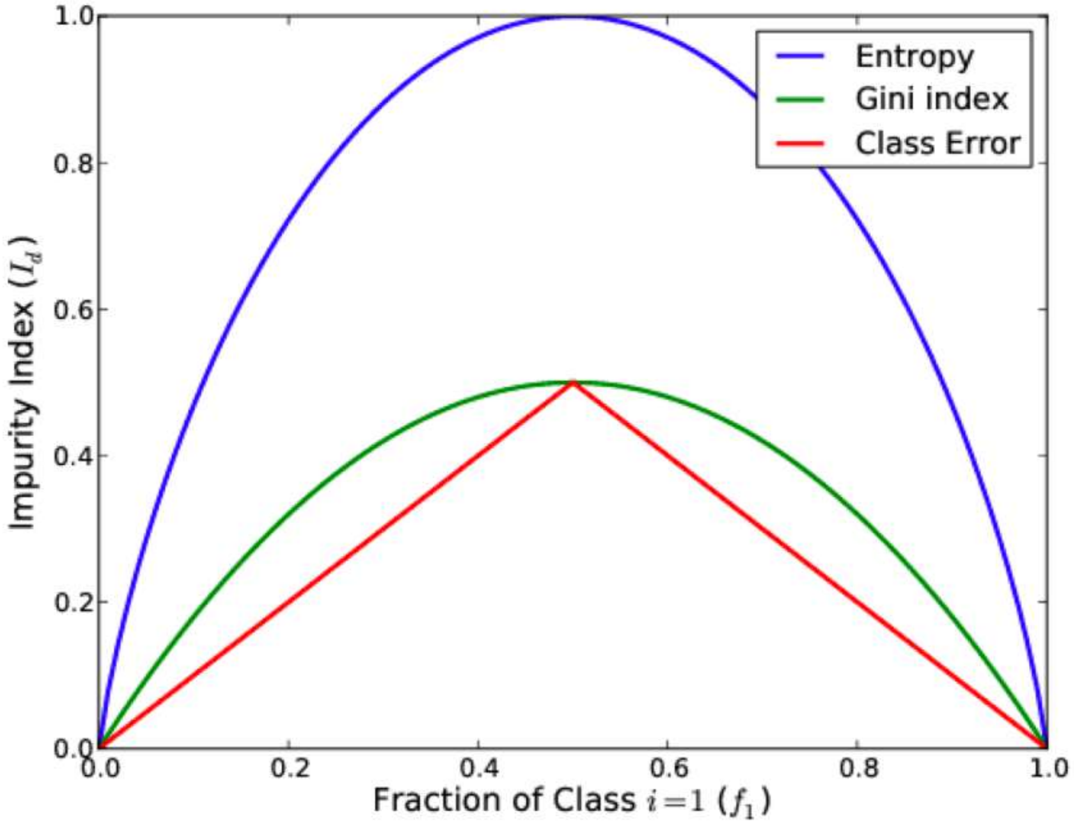
The splitting rule would continue until the child nodes are pure

source: https://sebastianraschka.com/faq/docs/decisiontree-error-vs-entropy.html

Classification Error Rate vs. Entropy (Gini)

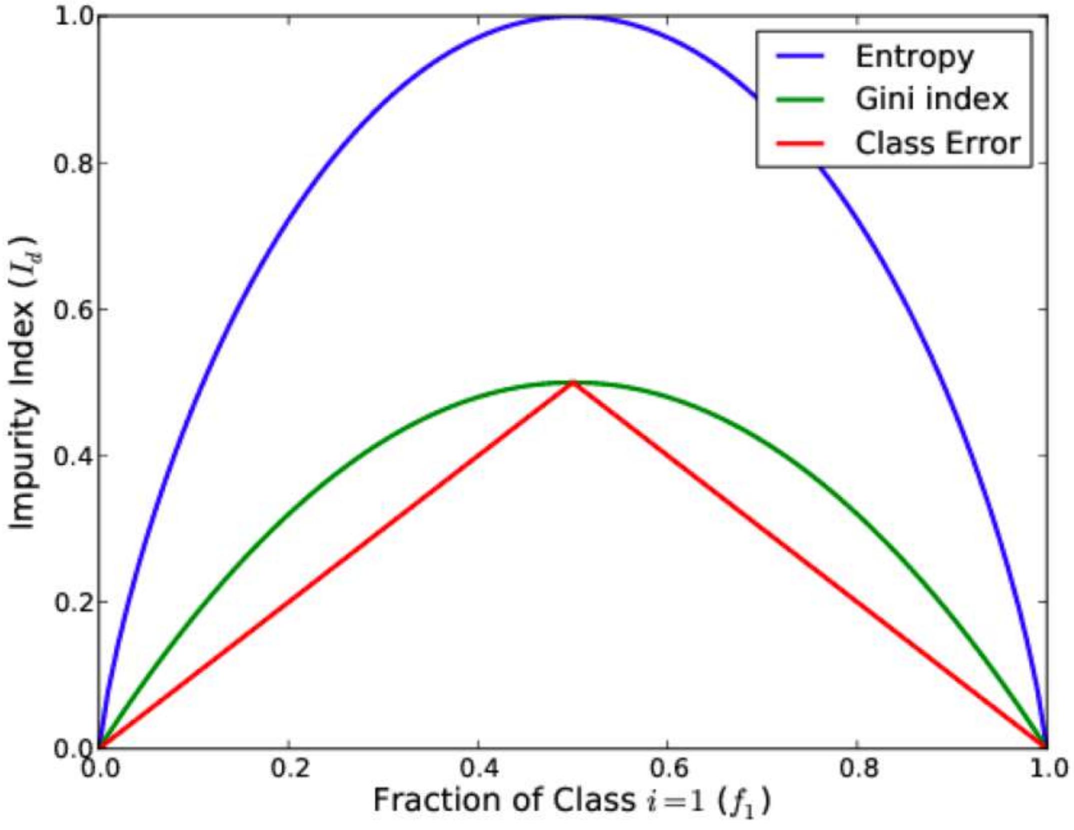


Classification Error Rate vs. Entropy (Gini)

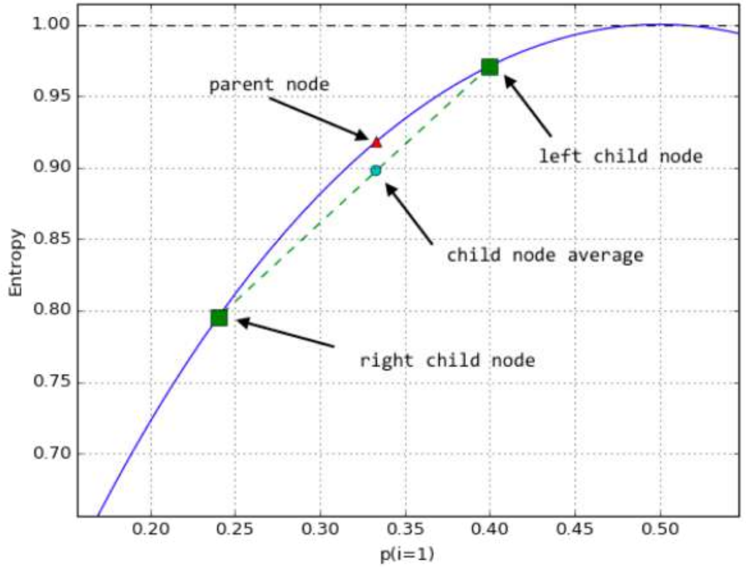


Gini and Entropy are "smoother" than classification error

Classification Error Rate vs. Entropy (Gini)



Gini and Entropy are "smoother" than classification error



Entropy is always larger than the weighted averaged entropy due to its "bell shape"

Evolution of Entropy

- Suppose we have $x_1, \ldots, x_N$ N items

Evolution of Entropy

- Suppose we have $x_1, \ldots, x_N$ N items
- Each item is labelled either as positive ($y=1$) or negative ($y=0$)

Evolution of Entropy

- Suppose we have $x_1, \ldots, x_N$ N items
- Each item is labelled either as positive ($y=1$) or negative ($y=0$)
- Let N^+ (N^-) be the number of positive (negative) elements

Evolution of Entropy

- Suppose we have $x_1, \ldots, x_N$ N items
- Each item is labelled either as positive ($y=1$) or negative ($y=0$)
- Let N^+ (N^-) be the number of positive (negative) elements
- We define the ratio $p = N^+/N$ and $q = N^-/N$

Evolution of Entropy

- Suppose we have $x_1, \ldots, x_N$ N items
- Each item is labelled either as positive (y=1) or negative (y=0)
- Let N^+ (N^-) be the number of positive (negative) elements
- We define the ratio $p = N^+/N$ and $q = N^-/N$
- The entropy is defined as:

$$H = -\Big[p\log_2(p) + q\log_2(q)\Big] = -\frac{N^+}{N}\log_2\left(\frac{N^+}{N}\right) - \frac{N^-}{N}\log_2\left(\frac{N^-}{N}\right)$$

Evolution of Entropy

$$H = -\Big[p\log_2(p) + q\log_2(q)\Big] = -\frac{N^+}{N}\log_2\left(\frac{N^+}{N}\right) - \frac{N^-}{N}\log_2\left(\frac{N^-}{N}\right)$$

Evolution of Entropy

$$H = -\left[p \log_2(p) + q \log_2(q)\right] = -\frac{N^+}{N} \log_2\left(\frac{N^+}{N}\right) - \frac{N^-}{N} \log_2\left(\frac{N^-}{N}\right)$$

- H is minimum (i.e., H=0) when either N^+ or N^- is equal to N
  - best case of a pure set (all the elements have the same label)

Evolution of Entropy

$$H = -\left[p\log_2(p) + q\log_2(q)\right] = -\frac{N^+}{N}\log_2\left(\frac{N^+}{N}\right) - \frac{N^-}{N}\log_2\left(\frac{N^-}{N}\right)$$

- H is minimum (i.e., H=0) when either N^+ or N^- is equal to N
  - best case of a pure set (all the elements have the same label)
- H is maximum (i.e., H=1) when $N^+ = N^- = N/2$
  - worst case (half of the elements labelled positive, half negative)

Evolution of Entropy

$$H = -\Big[p\log_2(p) + q\log_2(q)\Big] = -\frac{N^+}{N}\log_2\left(\frac{N^+}{N}\right) - \frac{N^-}{N}\log_2\left(\frac{N^-}{N}\right)$$

- H is minimum (i.e., H=0) when either N^+ or N^- is equal to N
  - best case of a pure set (all the elements have the same label)
- H is maximum (i.e., H=1) when $N^+ = N^- = N/2$
  - worst case (half of the elements labelled positive, half negative)
- H ranges in [0, 1] independently of the size of the set

Evolution of Entropy

$$H = -\left[p\log_2(p) + q\log_2(q)\right] = -\frac{N^+}{N}\log_2\left(\frac{N^+}{N}\right) - \frac{N^-}{N}\log_2\left(\frac{N^-}{N}\right)$$

- H is minimum (i.e., H=0) when either N^+ or N^- is equal to N
  - best case of a pure set (all the elements have the same label)
- H is maximum (i.e., H=1) when $N^+ = N^- = N/2$
  - worst case (half of the elements labelled positive, half negative)
- H ranges in [0, 1] independently of the size of the set

Only because each x_i takes on a binary value, in general H ranges in [0, +infinity]

Evolution of Entropy in Tree Splitting

- Suppose we start with a node whose entropy is 1 (half +/half -)

Evolution of Entropy in Tree Splitting

- Suppose we start with a node whose entropy is 1 (half +/half -)
- The worst split will occur if the two resulting children nodes will both contain half +/half - examples
  - In such a case, each child node will still have entropy = 1

Evolution of Entropy in Tree Splitting

- Suppose we start with a node whose entropy is 1 (half +/half -)
- The worst split will occur if the two resulting children nodes will both contain half +/half - examples
  - In such a case, each child node will still have entropy = 1
- To account for the number of elements, the weighted average of children entropies are computed

$$H_{\text{split}} = \frac{N_{\text{left}}}{N} H_{\text{left}} + \frac{N_{\text{right}}}{N} H_{\text{right}}$$

Evolution of Entropy in Tree Splitting

- Suppose we start with a node whose entropy is 1 (half +/half -)
- The worst split will occur if the two resulting children nodes will both contain half +/half - examples
  - In such a case, each child node will still have entropy = 1
- To account for the number of elements, the weighted average of children entropies are computed

splitting can't do worst!

$$H_{\text{split}} = \frac{N_{\text{left}}}{N} H_{\text{left}} + \frac{N_{\text{right}}}{N} H_{\text{right}}$$

Linear Models vs. Decision Trees

Linear Models

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$$

Decision Trees

$$h(\mathbf{x}) = \sum_{j=1}^{J} c_j \cdot \mathbf{1}_{R_j}(\mathbf{x})$$

Learned hypothesis is constant within a region

Linear Models vs. Decision Trees

Linear Models

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$$

Decision Trees

$$h(\mathbf{x}) = \sum_{j=1}^{J} c_j \cdot \mathbf{1}_{R_j}(\mathbf{x})$$

Learned hypothesis is constant within a region

Which one is better?

Linear Models vs. Decision Trees

Linear Models

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$$

Decision Trees

$$h(\mathbf{x}) = \sum_{j=1}^{J} c_j \cdot \mathbf{1}_{R_j}(\mathbf{x})$$

Learned hypothesis is constant within a region

Which one is better?

If there is a strong linear relationship between input and output

Linear Models vs. Decision Trees

Linear Models

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$$

Decision Trees

$$h(\mathbf{x}) = \sum_{j=1}^{J} c_j \cdot \mathbf{1}_{R_j}(\mathbf{x})$$

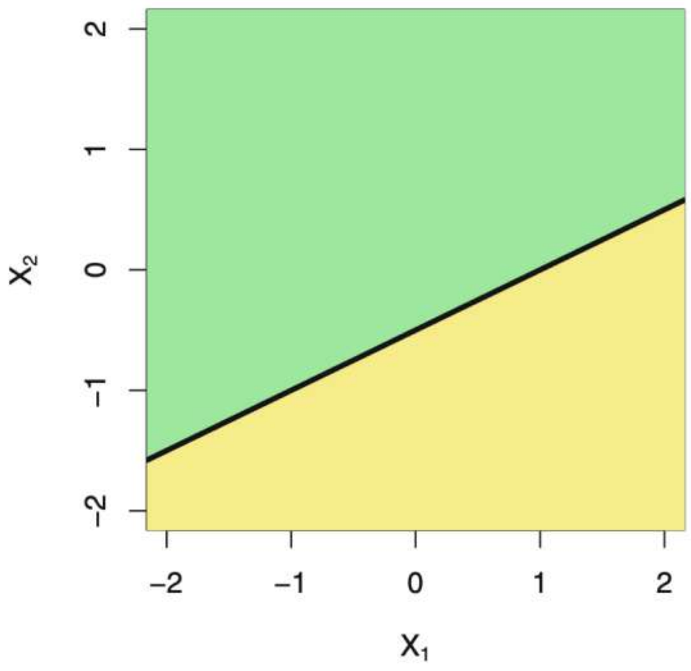
Learned hypothesis is constant within a region

Which one is better?

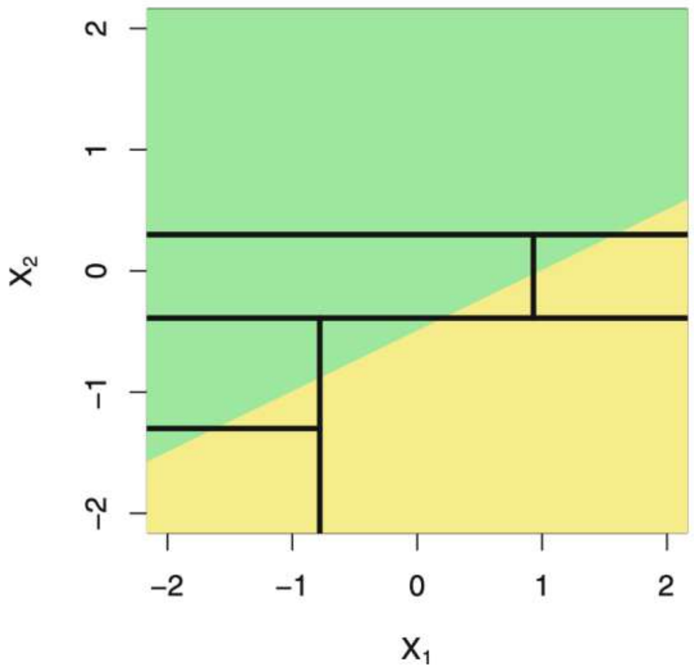
If there is a highly non-linear relationship between input and output

Linear Models vs. Decision Trees: Example

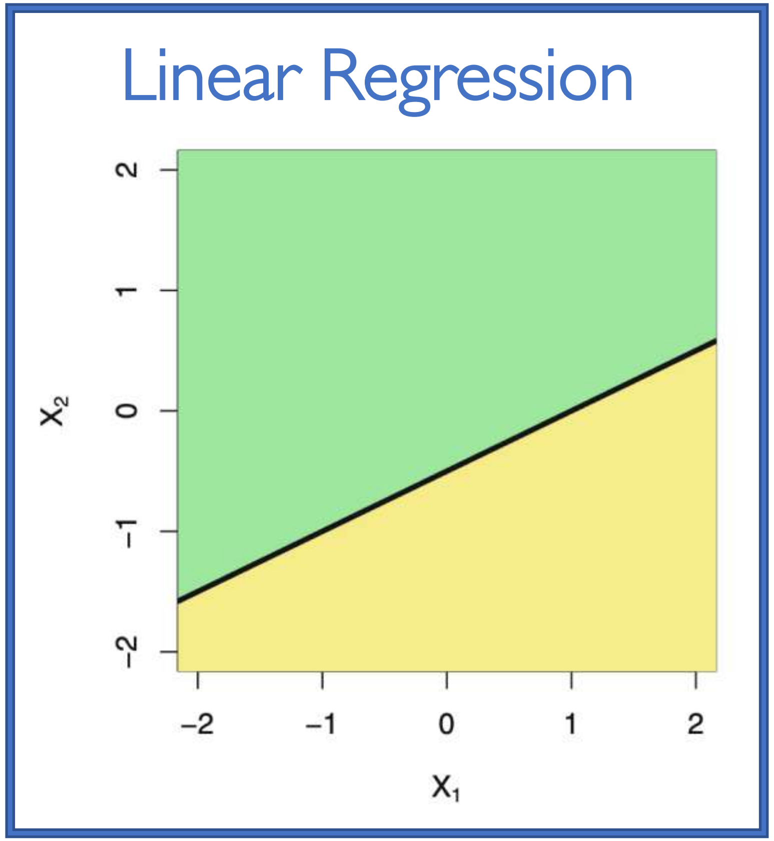
Linear Regression

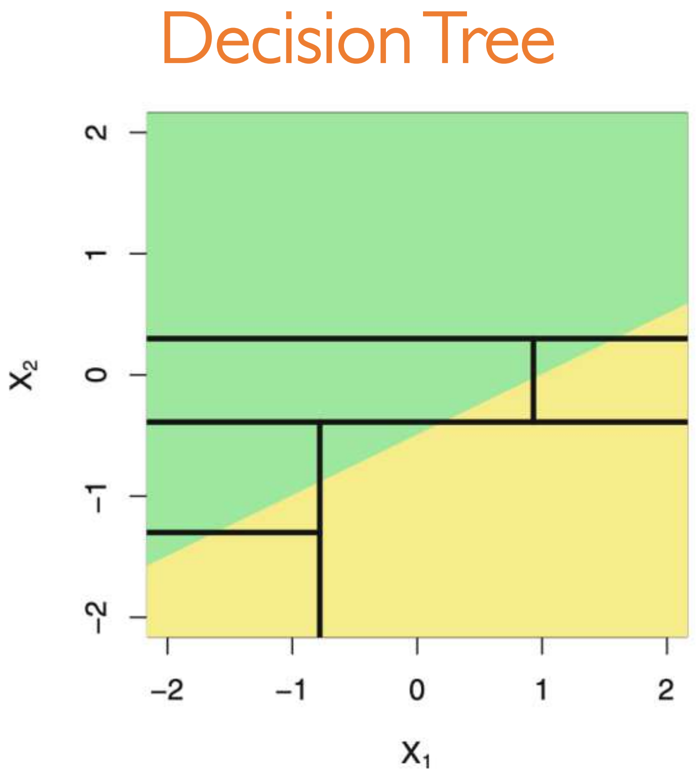


Decision Tree



Linear Models vs. Decision Trees: Example

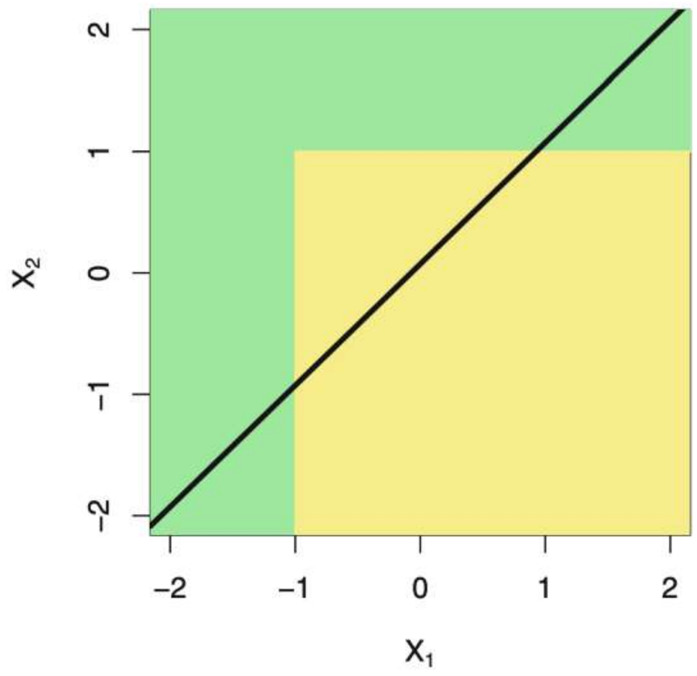




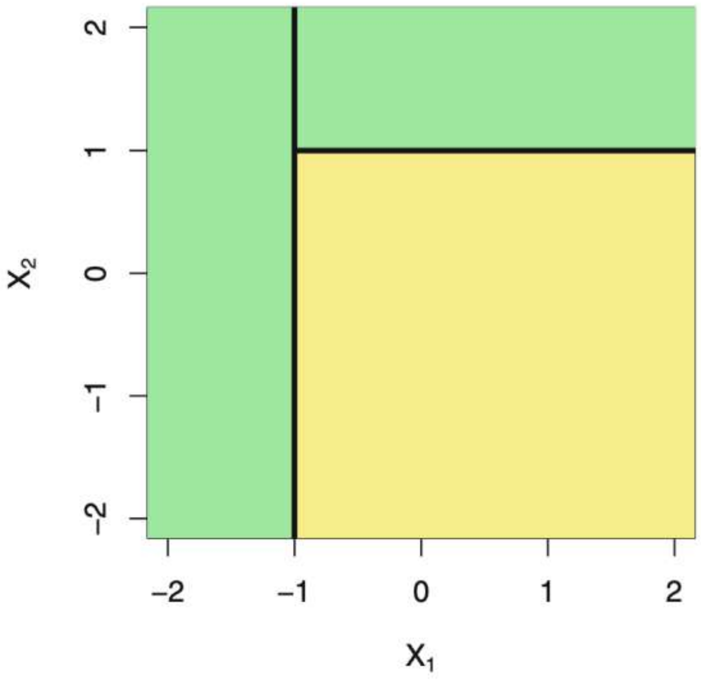
Nice linear decision boundary

Linear Models vs. Decision Trees: Example

Linear Regression

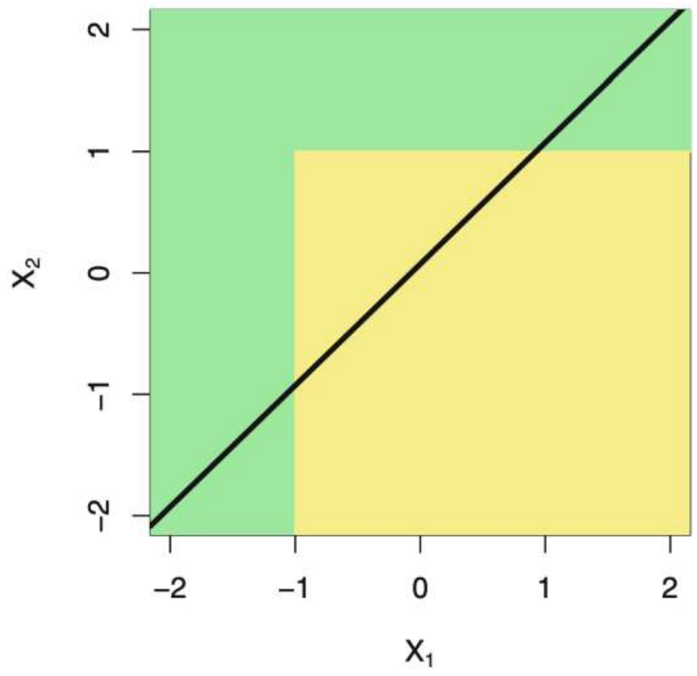


Decision Tree

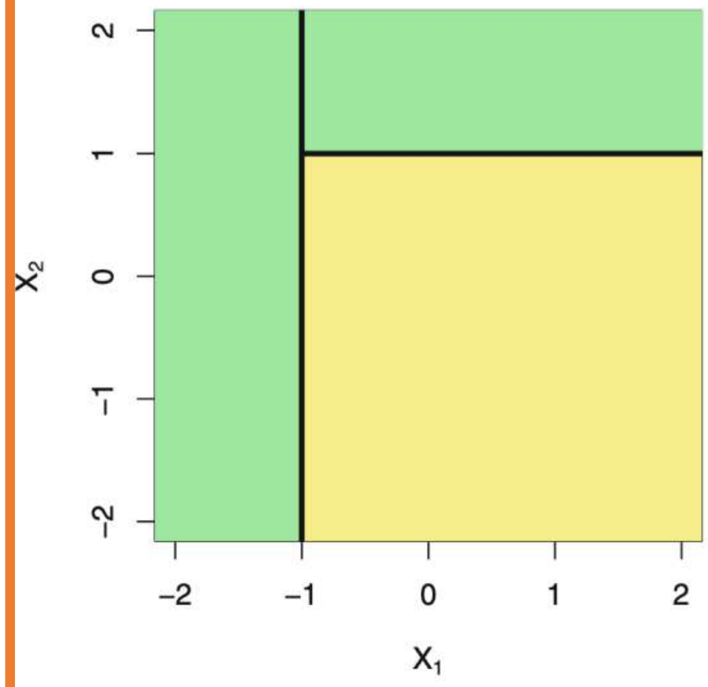


Linear Models vs. Decision Trees: Example

Linear Regression



Decision Tree



Non-linear decision boundary

Decision Trees: Pruning

- Greedy tree growing strategy may lead to:
  - Overfitting → if we keep splitting as long as there is an information gain

Decision Trees: Pruning

- Greedy tree growing strategy may lead to:
  - Overfitting → if we keep splitting as long as there is an information gain
  - Underfitting → if we early-stop the splitting process

Decision Trees: Pruning

- Greedy tree growing strategy may lead to:
  - Overfitting → if we keep splitting as long as there is an information gain
  - Underfitting → if we early-stop the splitting process
- A better strategy is to grow a very large tree T_0, and then **prune** it back in order to obtain a subtree

Decision Trees: Pruning

- Greedy tree growing strategy may lead to:
  - Overfitting → if we keep splitting as long as there is an information gain
  - Underfitting → if we early-stop the splitting process
- A better strategy is to grow a very large tree T_0, and then **prune** it back in order to obtain a subtree

How do we determine such a subtree?

Decision Trees: Pruning

- Greedy tree growing strategy may lead to:
  - Overfitting → if we keep splitting as long as there is an information gain
  - Underfitting → if we early-stop the splitting process
- A better strategy is to grow a very large tree T_0, and then **prune** it back in order to obtain a subtree

How do we determine such a subtree?

Intuitively, by selecting the subtree with the smalles test error!

Decision Trees: Pruning

- Given a subtree, we can estimate its test error using cross-validation or the validation set approach

Decision Trees: Pruning

- Given a subtree, we can estimate its test error using cross-validation or the validation set approach
- <u>Problem:</u> too many subtrees to consider (unfeasible!)

Decision Trees: Pruning

- Given a subtree, we can estimate its test error using cross-validation or the validation set approach
- Problem: too many subtrees to consider (unfeasible!)
- We need a way to select a small set of subtrees for consideration

Decision Trees: Pruning

- Given a subtree, we can estimate its test error using cross-validation or the validation set approach
- Problem: too many subtrees to consider (unfeasible!)
- We need a way to select a small set of subtrees for consideration

↓

Cost Complexity Pruning (a.k.a. Weakest Link Pruning)

Cost Complexity Pruning

- Rather than considering every possible subtree, we consider a sequence of trees indexed by a non-negative tuning parameter α

Cost Complexity Pruning

- Rather than considering every possible subtree, we consider a sequence of trees indexed by a non-negative tuning parameter α
- For each value of α there is a subtree $T \subset T_0$ so as to minimize:

$$\sum_{j=1}^{|T|} \sum_{i:\ \mathbf{x}_i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha|T|$$

Cost Complexity Pruning

- Rather than considering every possible subtree, we consider a sequence of trees indexed by a non-negative tuning parameter α
- For each value of α there is a subtree $T \subset T_0$ so as to minimize:

$$\sum_{j=1}^{|T|} \sum_{i:\ \mathbf{x}_i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha |T|$$

- $|T|$ indicates the number of terminal nodes of the tree T

Cost Complexity Pruning

- Rather than considering every possible subtree, we consider a sequence of trees indexed by a non-negative tuning parameter α
- For each value of α there is a subtree $T \subset T_0$ so as to minimize:

$$\sum_{j=1}^{|T|} \sum_{i:\ \mathbf{x}_i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha |T|$$

- $|T|$ indicates the number of terminal nodes of the tree T
- R_j is the subset of feature space corresponding to the *j*-th leaf

Cost Complexity Pruning

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data

Cost Complexity Pruning

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data
  - When $\alpha = 0$ T = T_0 (just minimize the training error)

Cost Complexity Pruning

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data
  - When $\alpha = 0$ T $= T_0$ (just minimize the training error)
  - As α increases there is a price to pay for the extra complexity of the tree

Cost Complexity Pruning

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data
  - When $\alpha = 0$ T $= T_0$ (just minimize the training error)
  - As α increases there is a price to pay for the extra complexity of the tree
- The cost complexity pruning is similar to LASSO regularization of linear models

Cost Complexity Pruning

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data
  - When $\alpha = 0$ $T = T_0$ (just minimize the training error)
  - As α increases there is a price to pay for the extra complexity of the tree
- The cost complexity pruning is similar to LASSO regularization of linear models
- We can select a value of α using a validation set or cross-validation

Decision Trees: PROs

- Trees are very easy to explain (even easier than linear regression!)

Decision Trees: PROs

- Trees are very easy to explain (even easier than linear regression!)
- Some people think decision trees mimic human decision-making

Decision Trees: PROs

- Trees are very easy to explain (even easier than linear regression!)
- Some people think decision trees mimic human decision-making
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small)

Decision Trees: PROs

- Trees are very easy to explain (even easier than linear regression!)
- Some people think decision trees mimic human decision-making
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small)
- Trees can easily handle categorical features without the need to create dummy variables (i.e., one-hot encoding)

Decision Trees: CONs

- Trees generally have lower predictive accuracy than regression and classification approaches

Decision Trees: CONs

- Trees generally have lower predictive accuracy than regression and classification approaches
- Trees may be not very robust: a small change in the data can cause a large change in the final estimated tree

Decision Trees: CONs

- Trees generally have lower predictive accuracy than regression and classification approaches
- Trees may be not very robust: a small change in the data can cause a large change in the final estimated tree

Improvement: Ensembles of Decision Trees

Decision Trees: CONs

- Trees generally have lower predictive accuracy than regression and classification approaches
- Trees may be not very robust: a small change in the data can cause a large change in the final estimated tree

Improvement: Ensembles of Decision Trees

Bagging

Decision Trees: CONs

- Trees generally have lower predictive accuracy than regression and classification approaches
- Trees may be not very robust: a small change in the data can cause a large change in the final estimated tree

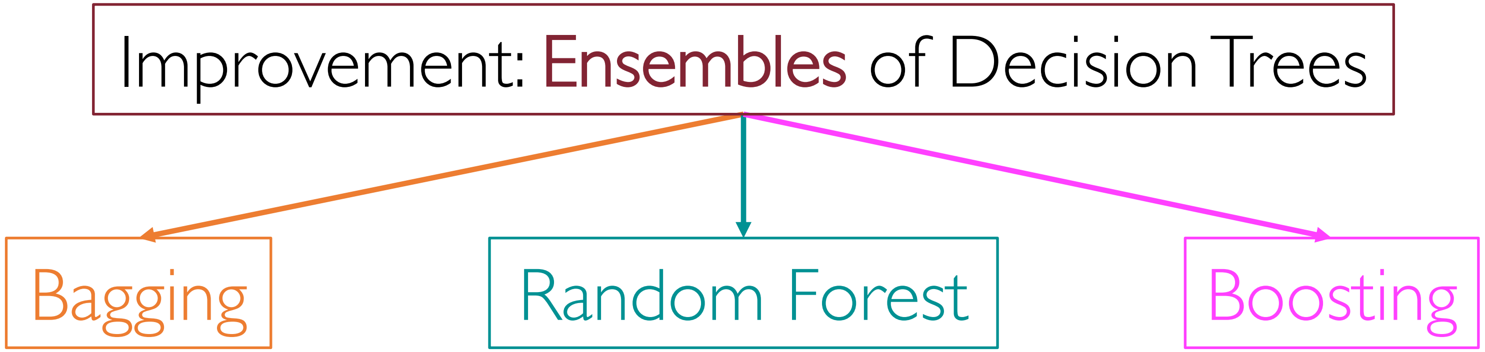
Improvement: Ensembles of Decision Trees

Bagging

Random Forest

Decision Trees: CONs

- Trees generally have lower predictive accuracy than regression and classification approaches
- Trees may be not very robust: a small change in the data can cause a large change in the final estimated tree



Tree Ensembles: Bagging

- Standard decision trees suffer from high variance (overfitting)

Tree Ensembles: Bagging

- Standard decision trees suffer from high variance (overfitting)
- If we randomly split a training set in two halves and fit a decision tree on each, chances are we end up with 2 very different trees

Tree Ensembles: Bagging

- Standard decision trees suffer from high variance (overfitting)
- If we randomly split a training set in two halves and fit a decision tree on each, chances are we end up with 2 very different trees
- Low-variance approaches, instead, are less sensitive to different training sets

Tree Ensembles: Bagging

- Standard decision trees suffer from high variance (overfitting)
- If we randomly split a training set in two halves and fit a decision tree on each, chances are we end up with 2 very different trees
- Low-variance approaches, instead, are less sensitive to different training sets
- Bootstrap aggregation (Bagging) is a general-purpose method to lower the variance of a statistical learning method

Bagging: Intuition

- Given a set of n i.i.d. observations $Z_1, \ldots, Z_n$, each with variance σ^2

Bagging: Intuition

- Given a set of n i.i.d. observations $Z_1, \ldots, Z_n$, each with variance σ^2
- The variance of the empirical mean is σ^2/n

Bagging: Intuition

- Given a set of n i.i.d. observations $Z_1, \ldots, Z_n$, each with variance σ^2
- The variance of the empirical mean is σ^2/n

$$\bar{Z} = \frac{1}{n}\left(Z_1 + \ldots + Z_n\right)$$

$$\mathrm{Var}(\bar{Z}) = \mathrm{Var}\left[\frac{1}{n}\left(Z_1 + \ldots + Z_n\right)\right] =$$

$$\frac{1}{n^2}\left[\mathrm{Var}(Z_1) + \ldots + \mathrm{Var}(Z_n)\right] =$$

$$\frac{1}{n^2}\left(\underbrace{\sigma^2 + \ldots + \sigma^2}_{n}\right) = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}$$

Bagging: Intuition

- Given a set of n i.i.d. observations $Z_1, \ldots, Z_n$, each with variance σ^2
- The variance of the empirical mean is σ^2/n

$$\bar{Z} = \frac{1}{n}\left(Z_1 + \ldots + Z_n\right)$$

$$\mathrm{Var}(\bar{Z}) = \mathrm{Var}\left[\frac{1}{n}\left(Z_1 + \ldots + Z_n\right)\right] =$$

$$\frac{1}{n^2}\left[\mathrm{Var}(Z_1) + \ldots + \mathrm{Var}(Z_n)\right] =$$

$$\frac{1}{n^2}\left(\underbrace{\sigma^2 + \ldots + \sigma^2}_{n}\right) = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}$$

$$\boxed{\begin{aligned} \mathrm{Var}(\alpha X) &= \alpha^2 \mathrm{Var}(X) \\ \mathrm{Var}\left(\sum_{i=1}^{n} X_i\right) &= \sum_{i=1}^{n} \mathrm{Var}(X_i) \quad \text{if } X_i \text{ are independent} \end{aligned}}$$

Bagging: Intuition

- Given a set of n i.i.d. observations $Z_1, \ldots, Z_n$, each with variance σ^2
- The variance of the empirical mean is σ^2/n

$$\bar{Z} = \frac{1}{n}\left(Z_1 + \ldots + Z_n\right)$$

$$\mathrm{Var}(\bar{Z}) = \mathrm{Var}\left[\frac{1}{n}\left(Z_1 + \ldots + Z_n\right)\right] =$$

$$\frac{1}{n^2}\left[\mathrm{Var}(Z_1) + \ldots + \mathrm{Var}(Z_n)\right] =$$

$$\frac{1}{n^2}\Big(\underbrace{\sigma^2 + \ldots + \sigma^2}_{n}\Big) = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}$$

$$\begin{aligned}\mathrm{Var}(\alpha X) = E[(\alpha X - E[\alpha X])^2] = E[(\alpha X - \alpha E[X])^2] &= \\ E[\alpha^2 (X - E[X])^2] &= \\ \alpha^2 E[(X - E[X])^2] &= \alpha^2 \mathrm{Var}(X)\end{aligned}$$

$$\boxed{\mathrm{Var}(\alpha X) = \alpha^2 \mathrm{Var}(X)}$$

$$\mathrm{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \mathrm{Var}(X_i) \quad \text{if } X_i \text{ are independent}$$

Bagging: Intuition

- Given a set of n i.i.d. observations $Z_1, \ldots, Z_n$, each with variance σ^2
- The variance of the empirical mean is σ^2/n

$$\bar{Z} = \frac{1}{n}\left(Z_1 + \ldots + Z_n\right)$$

$$\mathrm{Var}(\bar{Z}) = \mathrm{Var}\left[\frac{1}{n}\left(Z_1 + \ldots + Z_n\right)\right] =$$

$$\frac{1}{n^2}\left[\mathrm{Var}(Z_1) + \ldots + \mathrm{Var}(Z_n)\right] =$$

$$\frac{1}{n^2}\left(\underbrace{\sigma^2 + \ldots + \sigma^2}_{n}\right) = \frac{n\sigma^2}{n^2} = \boxed{\frac{\sigma^2}{n}}$$

$$\mathrm{Var}(\alpha X) = E[(\alpha X - E[\alpha X])^2] = E[(\alpha X - \alpha E[X])^2] = E[\alpha^2 (X - E[X])^2] = \alpha^2 E[(X - E[X])^2] = \alpha^2 \mathrm{Var}(X)$$

$$\boxed{\mathrm{Var}(\alpha X) = \alpha^2 \mathrm{Var}(X)}$$

$$\mathrm{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \mathrm{Var}(X_i) \quad \text{if } X_i \text{ are independent}$$

Averaging a set of observations reduces variance!

Bagging: Intuition

Take many training sets, build a separate prediction model on each, and average the resulting predictions

Bagging: Intuition

Take many training sets, build a separate prediction model on each, and average the resulting predictions

$h^1(\mathbf{x}), \ldots, h^B(\mathbf{x})$ B predictive models learned from B training sets

Bagging: Intuition

Take many training sets, build a separate prediction model on each, and average the resulting predictions

$h^1(\mathbf{x}), \ldots, h^B(\mathbf{x})$

B predictive models learned from B training sets

$$h_{\text{avg}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} h^b(\mathbf{x})$$

Final model obtained averaging the B predictions

Bagging: Intuition

Take many training sets, build a separate prediction model on each, and average the resulting predictions

$h^1(\mathbf{x}), \ldots, h^B(\mathbf{x})$ B predictive models learned from B training sets

$$h_{\text{avg}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} h^b(\mathbf{x})$$

Final model obtained averaging the B predictions

Unfortunately, we generally do not have access to multiple training sets

Bagging: Intuition

Take many training sets, build a separate prediction model on each, and average the resulting predictions

$h^1(\mathbf{x}), \ldots, h^B(\mathbf{x})$ — B predictive models learned from B training sets

$$h_{\text{avg}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} h^b(\mathbf{x})$$

Final model obtained averaging the B predictions

Unfortunately, we generally do not have access to multiple training sets

$\Downarrow$

Bootstrap

Taking repeated samples from the same training set

Bagging: Intuition

Generate B different bootstrapped samples from the original training set

Bagging: Intuition

Generate B different bootstrapped samples from the original training set

Train a model on each bootstrapped sample

Bagging: Intuition

Generate B different bootstrapped samples from the original training set

Train a model on each bootstrapped sample

Average the B predictions

$$\boxed{h_{\text{bagging}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} h^{b*}(\mathbf{x})}$$

Bagging

- Note that bagging is a general-purpose framework

Bagging

- Note that bagging is a general-purpose framework
- It can be used in combination with *any* model

Bagging

- Note that bagging is a general-purpose framework
- It can be used in combination with *any* model
- When used with classification trees the final prediction is typically obtained via majority voting
  - The overall prediction is just the most common across the B models

Bagging: Variable Importance

- The improved prediction accuracy of bagging trees comes at the expense of the interpretability of a single tree

Bagging: Variable Importance

- The improved prediction accuracy of bagging trees comes at the expense of the interpretability of a single tree
- Still, one can obtain an overall summary of the importance of each feature using RSS (regression) or Gini index/Entropy (classification)

Bagging: Variable Importance

- The improved prediction accuracy of bagging trees comes at the expense of the interpretability of a single tree
- Still, one can obtain an overall summary of the importance of each feature using RSS (regression) or Gini index/Entropy (classification)
- Add up the total RSS/Gini index reduction obtained splitting on a certain feature and take the average over all the B trees

Tree Ensemble: Random Forests

- Improve bagging trees through decorrelating individual trees

Tree Ensemble: Random Forests

- Improve bagging trees through decorrelating individual trees
- As in bagging, there will be B decision trees learned on bootstrapped samples of the original training set

Tree Ensemble: Random Forests

- Improve bagging trees through decorrelating individual trees
- As in bagging, there will be B decision trees learned on bootstrapped samples of the original training set
- But for each individual tree, every time it comes to splitting a node only a random sample of $k < n$ features is considered

Tree Ensemble: Random Forests

- Improve bagging trees through decorrelating individual trees
- As in bagging, there will be B decision trees learned on bootstrapped samples of the original training set
- But for each individual tree, every time it comes to splitting a node only a random sample of $k < n$ features is considered
- Each split is allowed to use only one of those k features

Random Forests: Sample Feature Split

- A fresh sample of k features is taken at each split

Random Forests: Sample Feature Split

- A fresh sample of k features is taken at each split
- A typical value of k is ~sqrt(n)

Random Forests: Sample Feature Split

- A fresh sample of k features is taken at each split
- A typical value of k is ~sqrt(n)
- At each split in the tree, the algorithm is not even allowed to consider a majority of the available features!

Random Forests: Sample Feature Split

- A fresh sample of *k* features is taken at each split
- A typical value of *k* is ~sqrt(*n*)
- At each split in the tree, the algorithm is not even allowed to consider a majority of the available features!
- This may sound crazy, but it has a clever rationale

Random Forests: Rationale

- Suppose a dataset contains a very strongly predictive feature along with a set of other moderately predictive variables

Random Forests: Rationale

- Suppose a dataset contains a very strongly predictive feature along with a set of other moderately predictive variables
- If we use "vanilla" bagging, most of the bagged trees will use that highly predictive feature as the top split

Random Forests: Rationale

- Suppose a dataset contains a very strongly predictive feature along with a set of other moderately predictive variables
- If we use "vanilla" bagging, most of the bagged trees will use that highly predictive feature as the top split
- As a result, most (if not all) the bagged trees will look very similar

Random Forests: Rationale

- Suppose a dataset contains a very strongly predictive feature along with a set of other moderately predictive variables
- If we use "vanilla" bagging, most of the bagged trees will use that highly predictive feature as the top split
- As a result, most (if not all) the bagged trees will look very similar
- Predictions from the individual bagged trees will be highly correlated

Random Forests: Rationale

- Suppose a dataset contains a very strongly predictive feature along with a set of other moderately predictive variables
- If we use "vanilla" bagging, most of the bagged trees will use that highly predictive feature as the top split
- As a result, most (if not all) the bagged trees will look very similar
- Predictions from the individual bagged trees will be highly correlated

Lower variance reduction

Random Forests: Rationale

- Enforce random sampling of *k* out of *n* features at each split

Random Forests: Rationale

- Enforce random sampling of k out of n features at each split
- The probability a split considers the strongest predictor is k/n
  - Hypergeometric distribution (1 success, k draws with replacement, pop. size = n)
  - For k = sqrt(n) this means 1/sqrt(n)

Random Forests: Rationale

- Enforce random sampling of k out of n features at each split
- The probability a split considers the strongest predictor is k/n
  - Hypergeometric distribution (1 success, k draws with replacement, pop. size = n)
  - For $k = \text{sqrt}(n)$ this means $1/\text{sqrt}(n)$
- We can think of this process as **decorrelating** the trees

Random Forests: Rationale

- Enforce random sampling of k out of n features at each split
- The probability a split considers the strongest predictor is k/n
  - Hypergeometric distribution (1 success, k draws with replacement, pop. size $= n$)
  - For $k = \text{sqrt}(n)$ this means $1/\text{sqrt}(n)$
- We can think of this process as **decorrelating** the trees
- Note that when $k = n$ this simply resembles to bagging

Random Forests: Rationale

- Enforce random sampling of k out of n features at each split
- The probability a split considers the strongest predictor is k/n
  - Hypergeometric distribution (1 success, k draws with replacement, pop. size = n)
  - For k = sqrt(n) this means 1/sqrt(n)
- We can think of this process as **decorrelating** the trees
- Note that when $k = n$ this simply resembles to bagging
- As with bagging, random forests will not overfit if we increase B

Tree Ensemble: Boosting

- Again, general approach that can be applied to many statistical learning methods for regression or classification

Tree Ensemble: Boosting

- Again, general approach that can be applied to many statistical learning methods for regression or classification
- In bagging, each tree is built on a bootstrap data set, independent of the other trees

Tree Ensemble: Boosting

- Again, general approach that can be applied to many statistical learning methods for regression or classification
- In bagging, each tree is built on a bootstrap data set, independent of the other trees
- Boosting works in a similar way, except that the trees are grown sequentially using information from previously grown trees

Tree Ensemble: Boosting

- Again, general approach that can be applied to many statistical learning methods for regression or classification
- In bagging, each tree is built on a bootstrap data set, independent of the other trees
- Boosting works in a similar way, except that the trees are grown sequentially using information from previously grown trees
- Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set

Boosting

- Unlike fitting a single large decision tree to the data, potentially leading to overfitting, the boosting approach instead learns slowly

Boosting

- Unlike fitting a single large decision tree to the data, potentially leading to overfitting, the boosting approach instead learns slowly
- Consider boosting regression trees:
  1. Fit the tree to the current residuals rather than the actual response Y
  2. Add this new decision tree into the fitted function so as to update the residuals
  3. Each of these trees can be rather small, with just a few terminal nodes, determined by a model's hyperparameter (d)
  4. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals

Boosting: Algorithm

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Boosting: Hyperparameters

- Boosting has 3 tuning hyperparameters:
  - The number of trees B (boosting can overfit if B is too large)

Boosting: Hyperparameters

- Boosting has 3 tuning hyperparameters:
  - The number of trees B (boosting can overfit if B is too large)
  - The shrinkage parameter λ (a small positive number) controls the rate at which boosting learns [0.01 ÷ 0.001]

Boosting: Hyperparameters

- Boosting has 3 tuning hyperparameters:
  - The number of trees B (boosting can overfit if B is too large)
  - The shrinkage parameter λ (a small positive number) controls the rate at which boosting learns $[0.01 \div 0.001]$
  - The number d of splits in each tree, which controls the complexity of the boosted ensemble (often $d = 1$ works well, in which case each tree is a stump)

Boosting: Hyperparameters

- Boosting has 3 tuning hyperparameters:
  - The number of trees B (boosting can overfit if B is too large)
  - The shrinkage parameter λ (a small positive number) controls the rate at which boosting learns $[0.01 \div 0.001]$
  - The number d of splits in each tree, which controls the complexity of the boosted ensemble (often $d = 1$ works well, in which case each tree is a stump)
- Tuning done via validation or cross-validation

Take-Home Message of Today

- Decision Trees (DTs) highly expressive yet interpretable models both for regression and classification

Take-Home Message of Today

- Decision Trees (DTs) highly expressive yet interpretable models both for regression and classification
- Learning the optimal DT is NP-Complete: Recursive Binary Splitting algorithm is an effective greedy heuristic

Take-Home Message of Today

- Decision Trees (DTs) highly expressive yet interpretable models both for regression and classification
- Learning the optimal DT is NP-Complete: Recursive Binary Splitting algorithm is an effective greedy heuristic
- DTs tend to overfit and have a low prediction accuracy

Take-Home Message of Today

- Decision Trees (DTs) highly expressive yet interpretable models both for regression and classification
- Learning the optimal DT is NP-Complete: Recursive Binary Splitting algorithm is an effective greedy heuristic
- DTs tend to overfit and have a low prediction accuracy
- Pruning and Ensembling techniques overcome both issues