Big Data Computing

Master's Degree in Computer Science

2022-2023

Gabriele Tolomei

Department of Computer Science

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

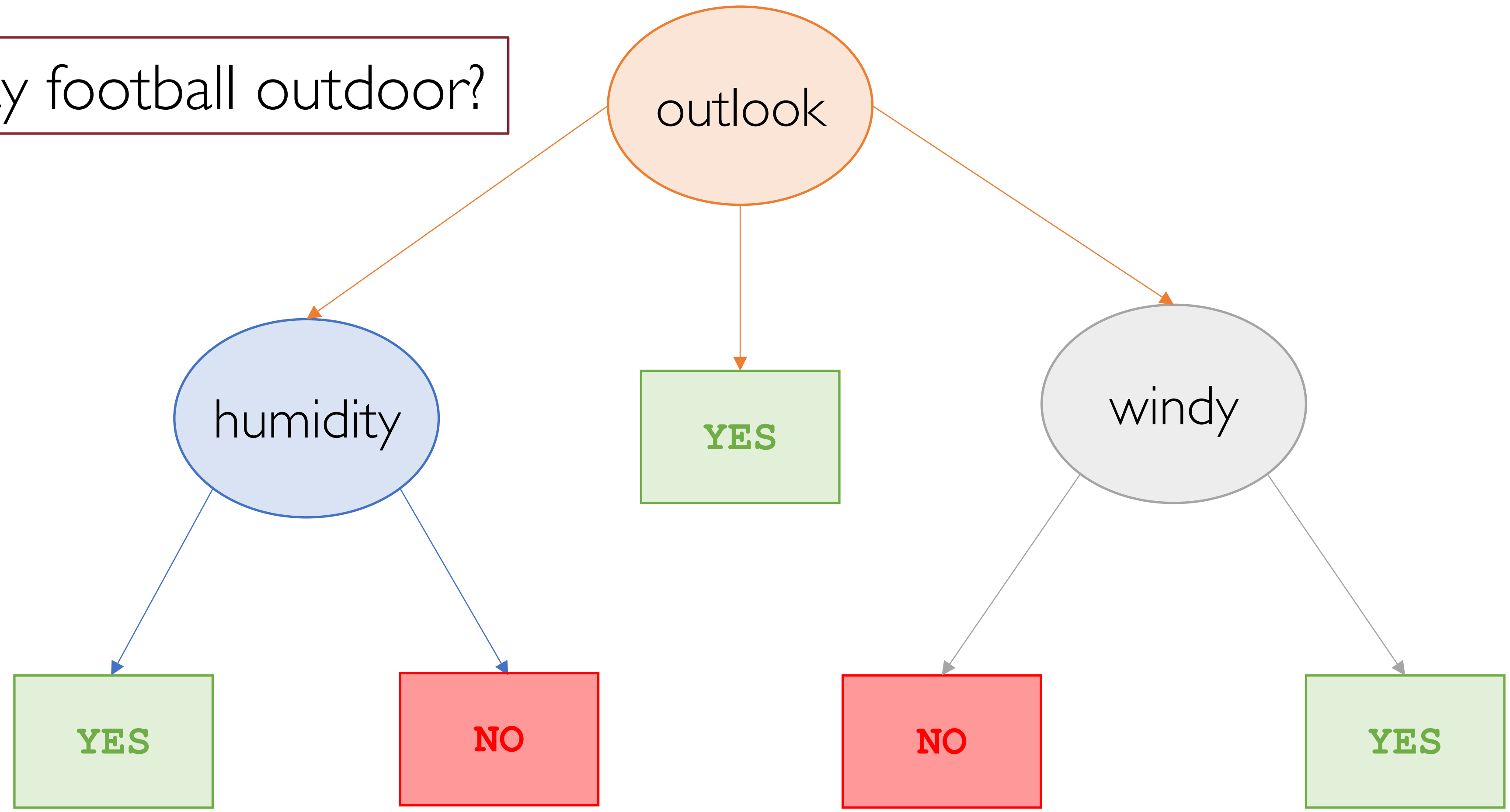
- Suitable for both regression and classification tasks
- Work by repeatedly splitting the input feature space into a number of regions
- A prediction is computed by taking the mean (regression) or the mode (classification) of the points in the region which the observation belongs
- The set of splitting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

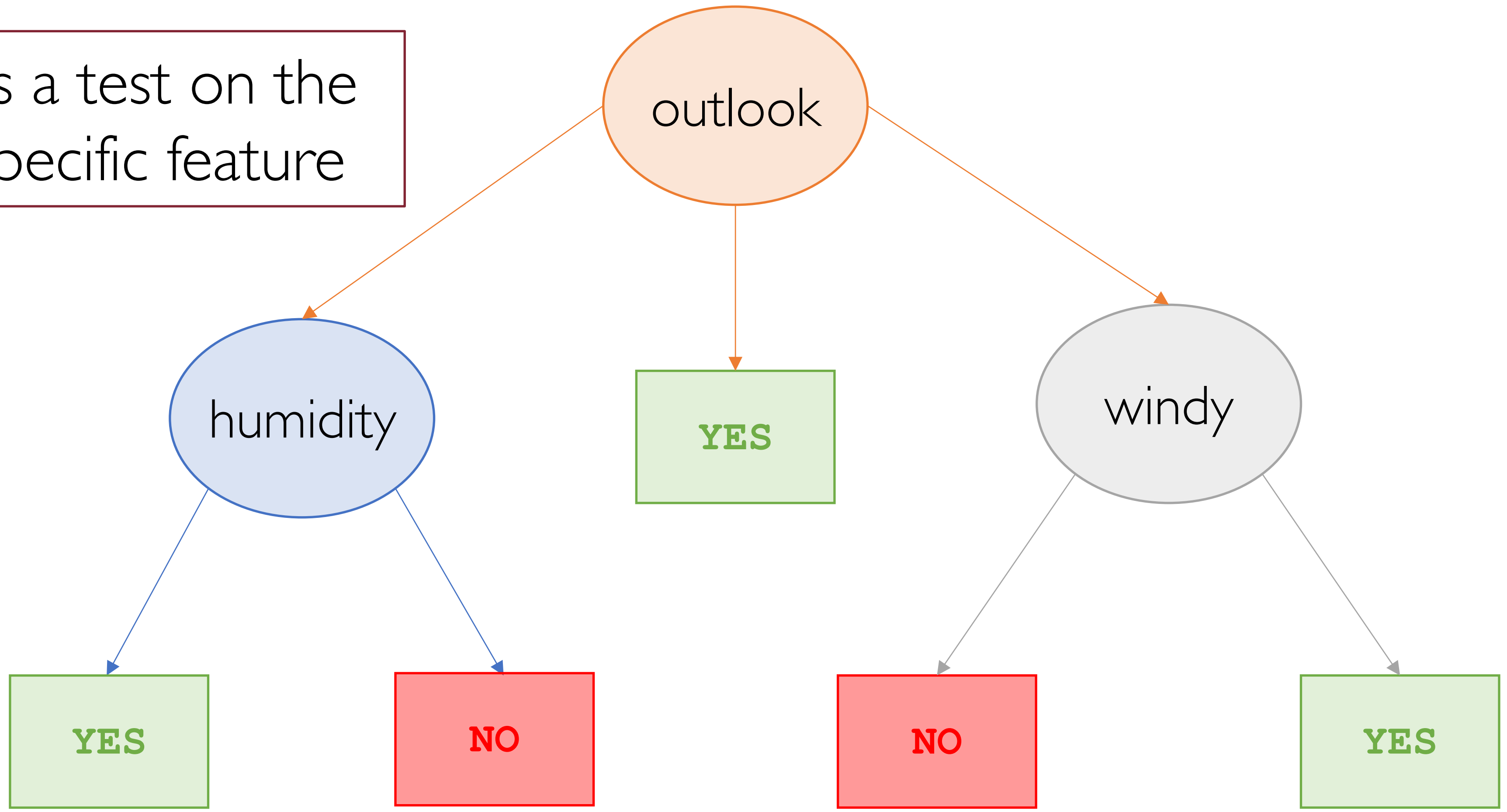
Decision Tree: An Example

Should we play football outdoor?

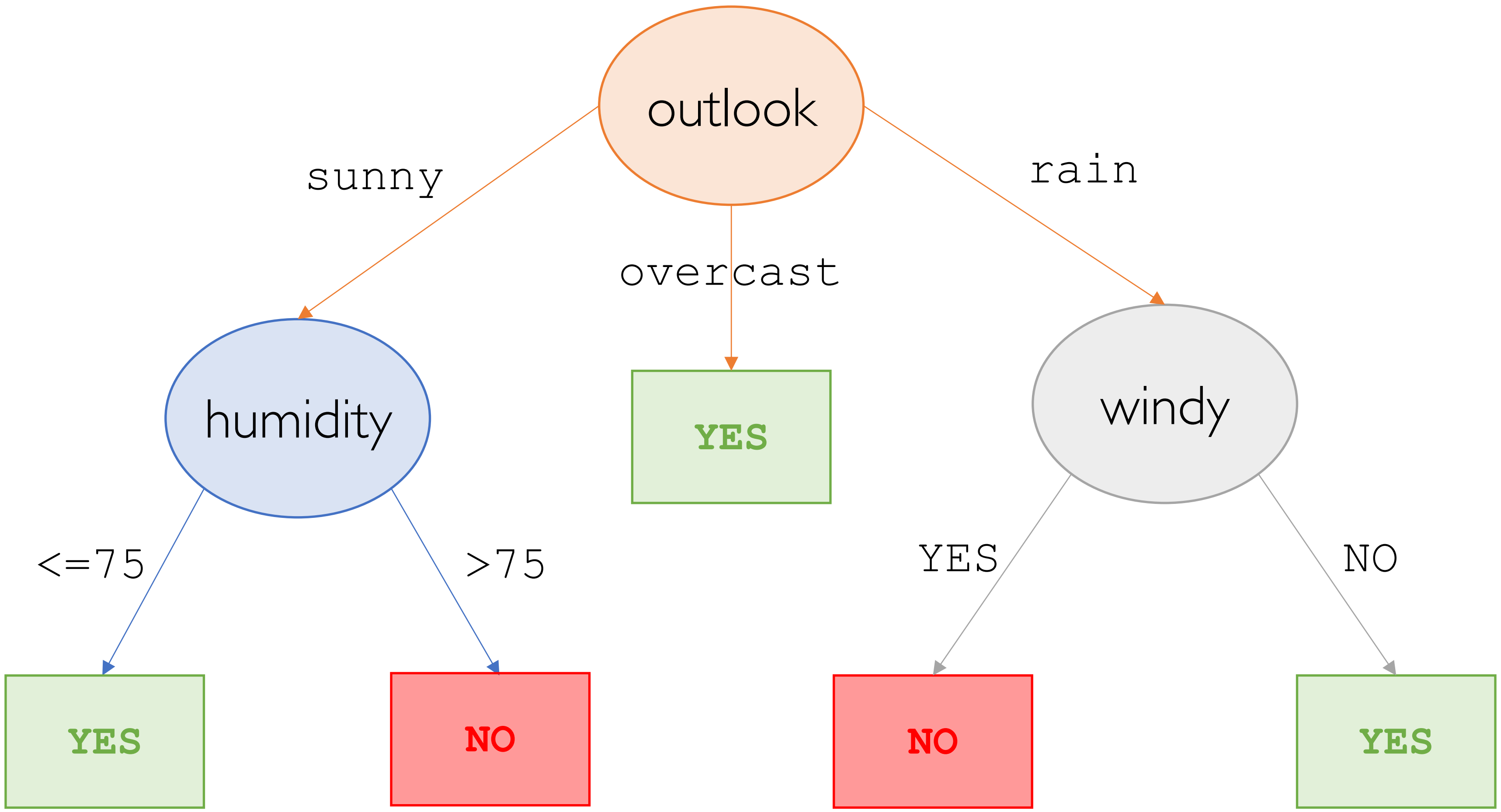


Decision Tree: An Example

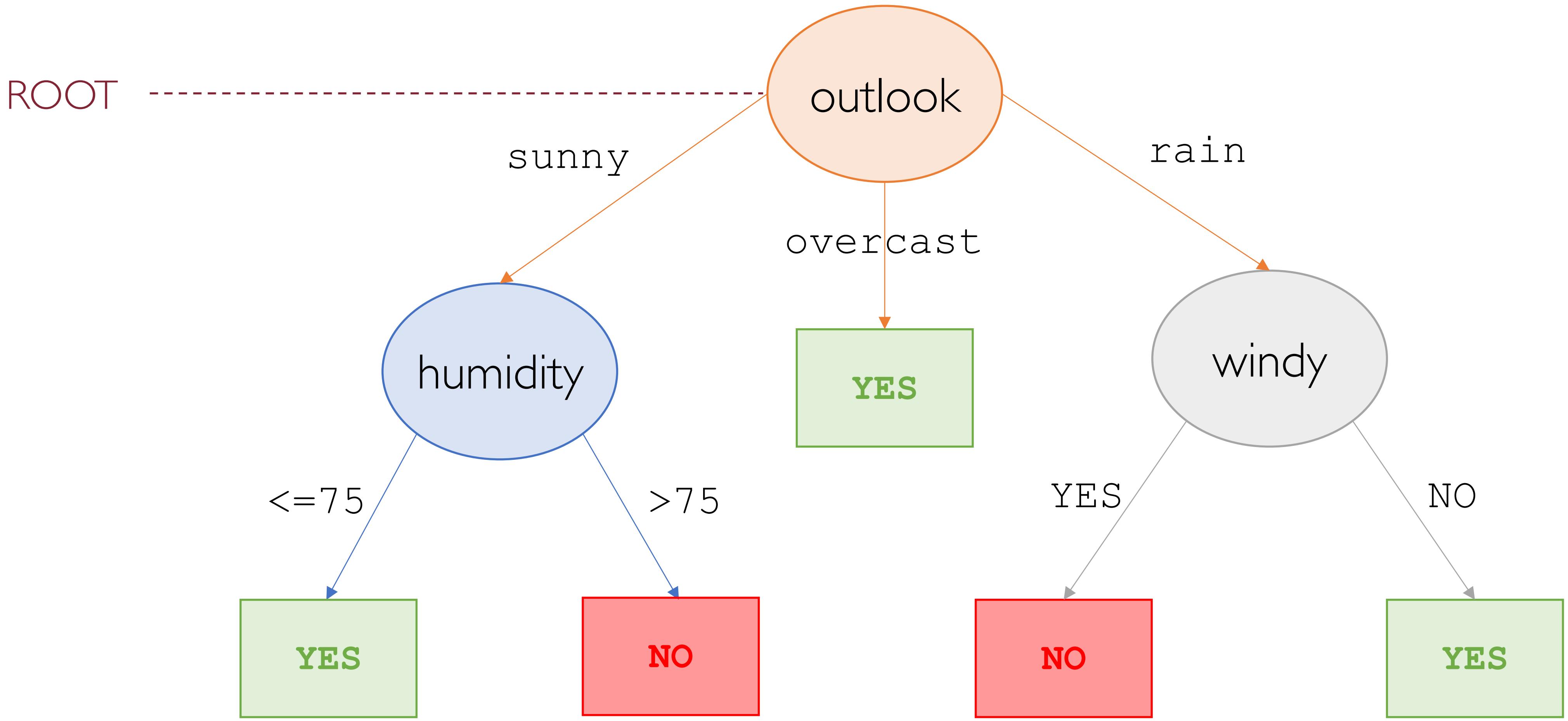
Each node is a test on the value of a specific feature



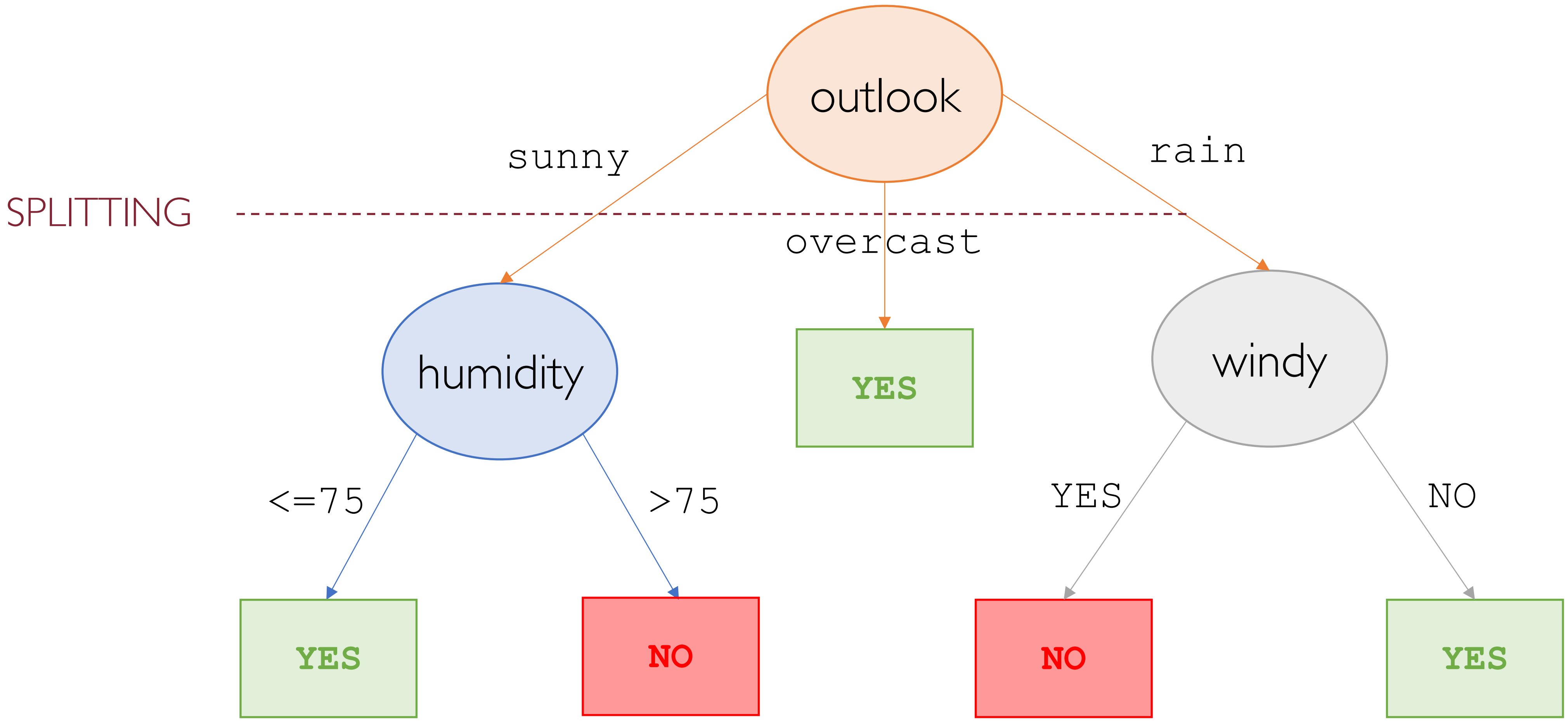
Decision Tree: An Example



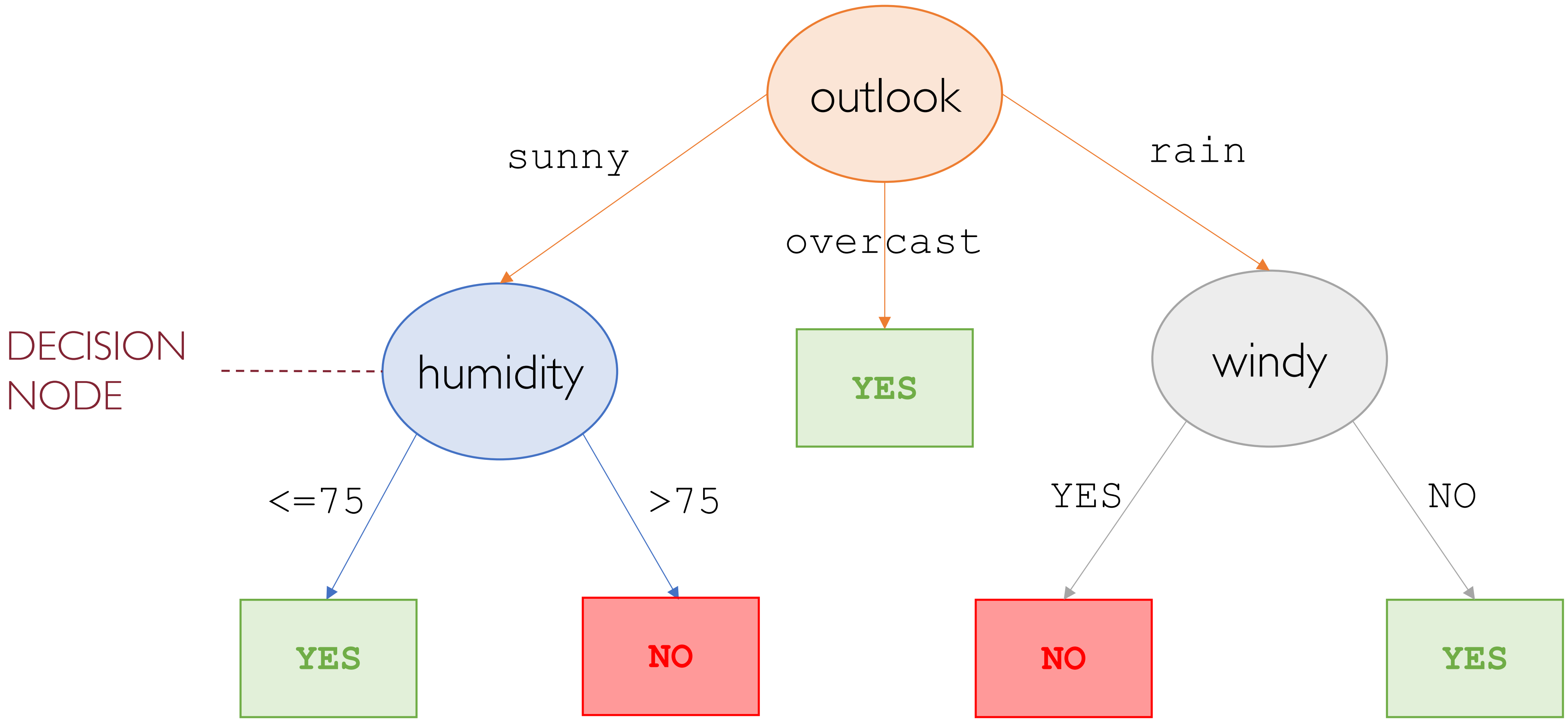
Decision Tree: An Example



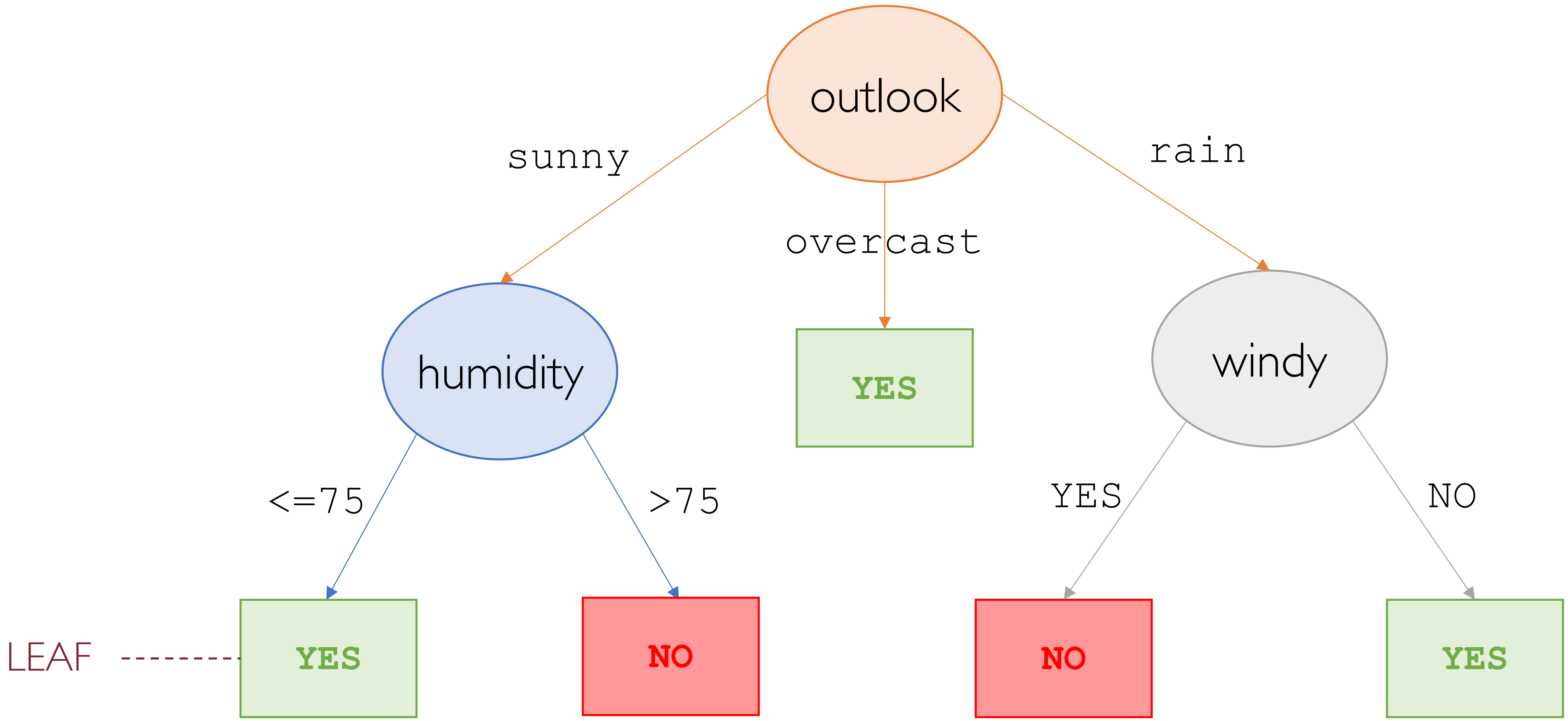
Decision Tree: An Example



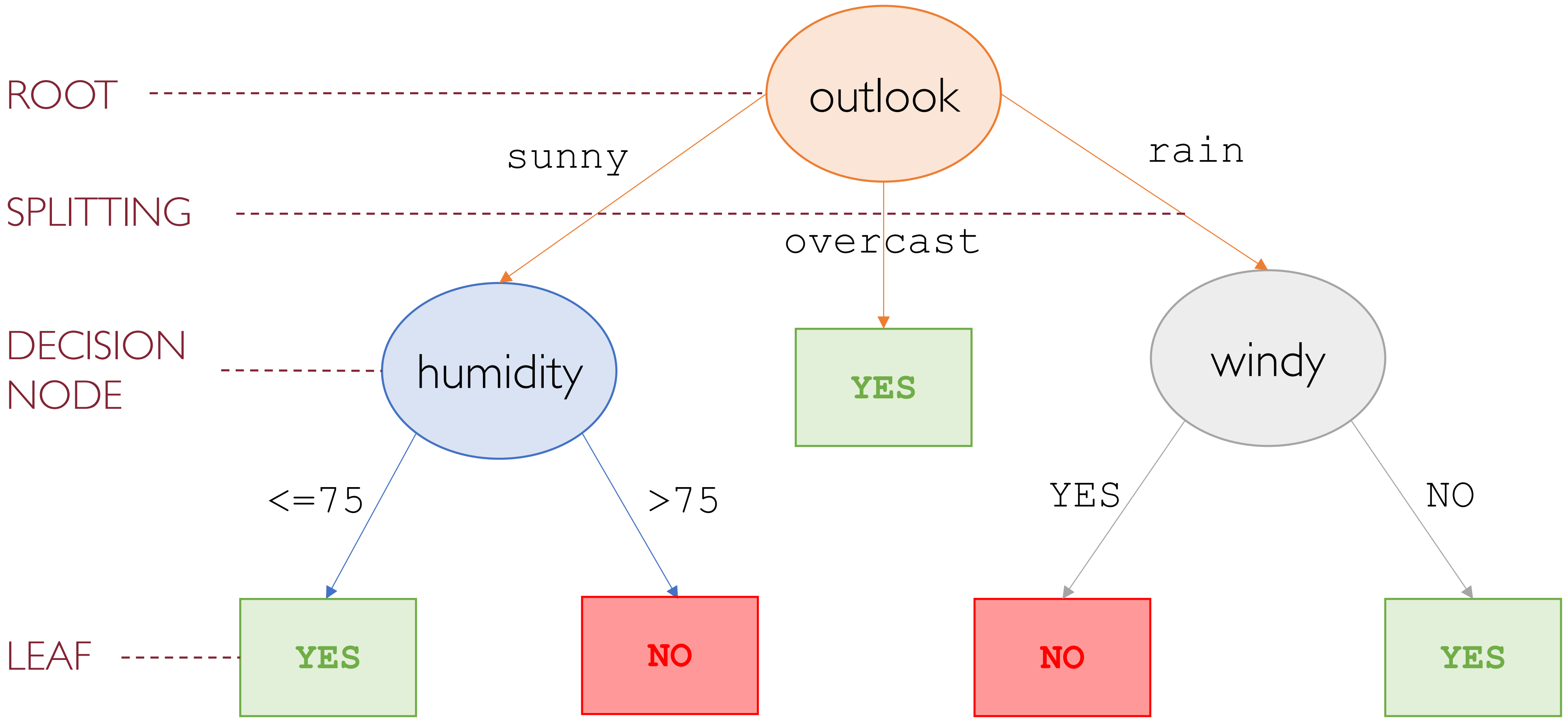
Decision Tree: An Example



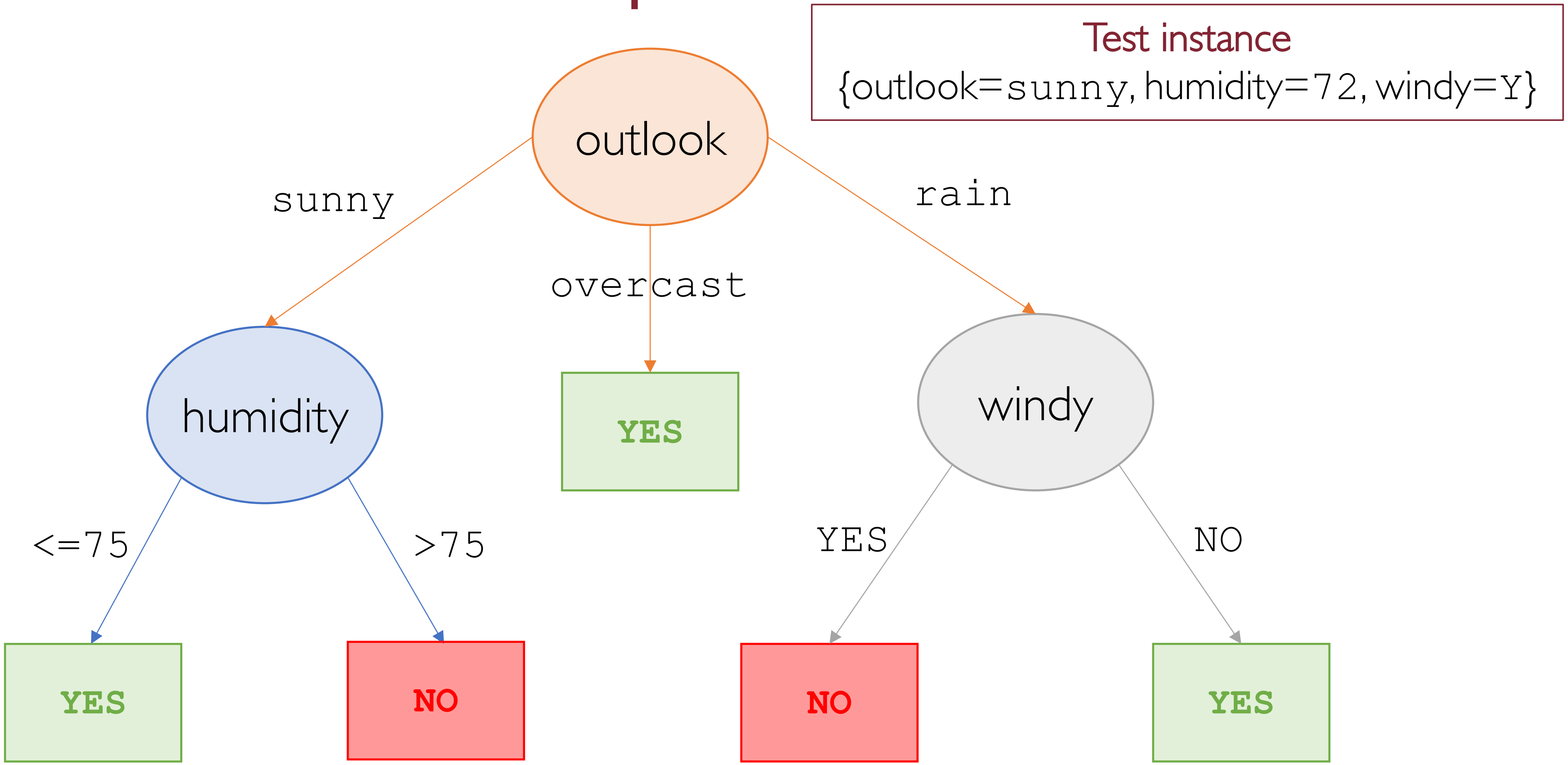
Decision Tree: An Example



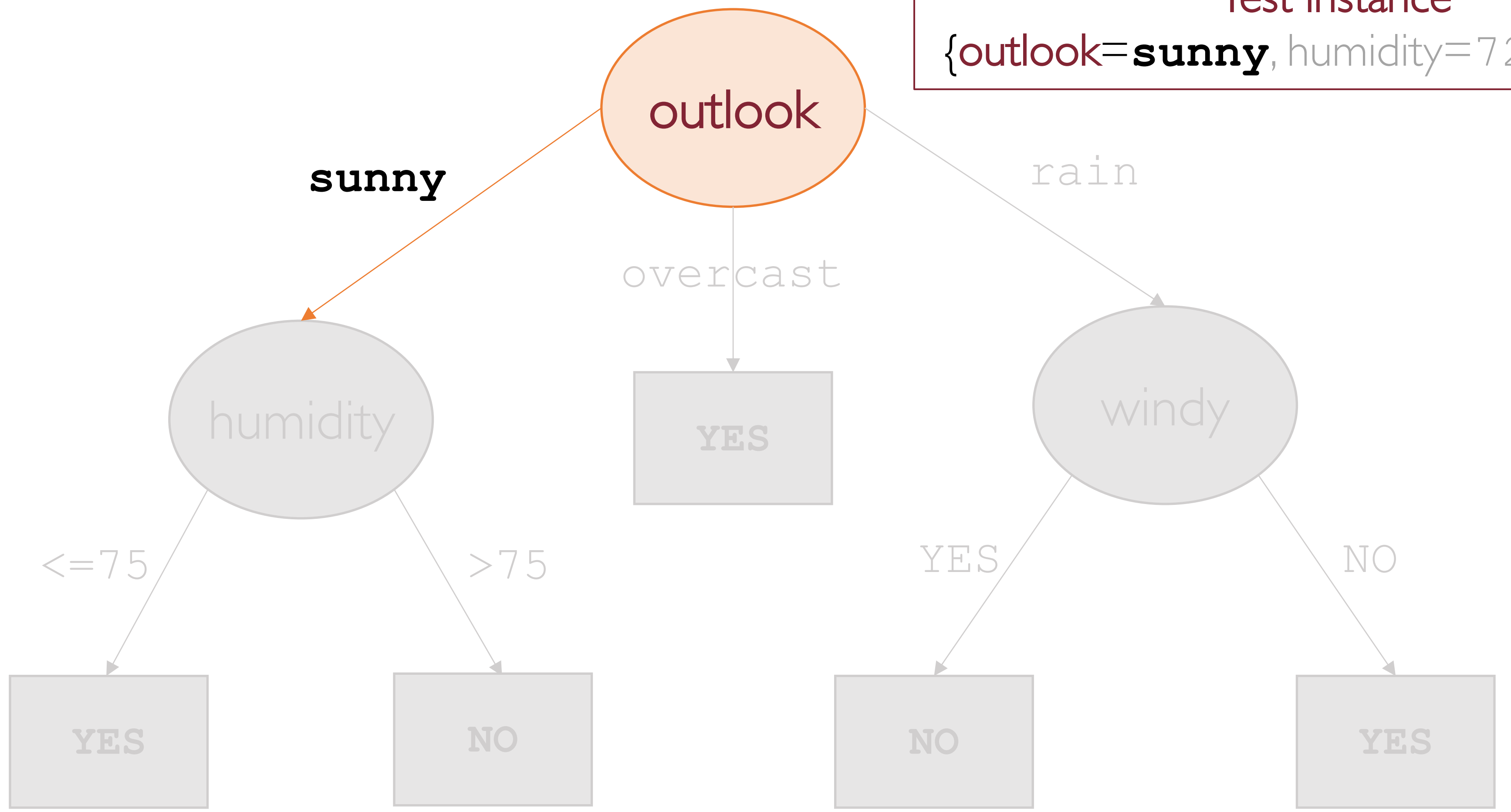
Decision Tree: An Example



Decision Tree: An Example



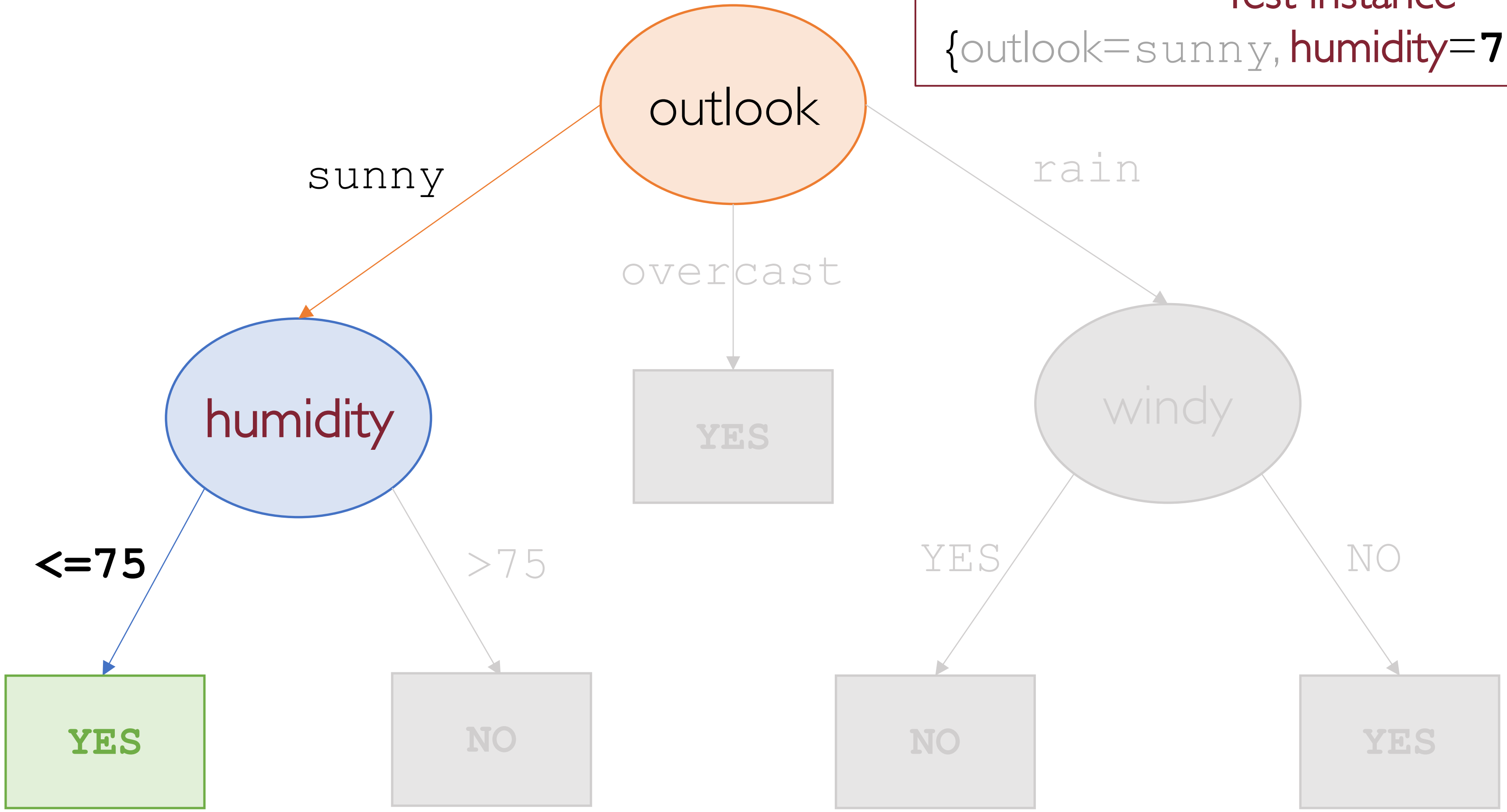
Decision Tree: An Example



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

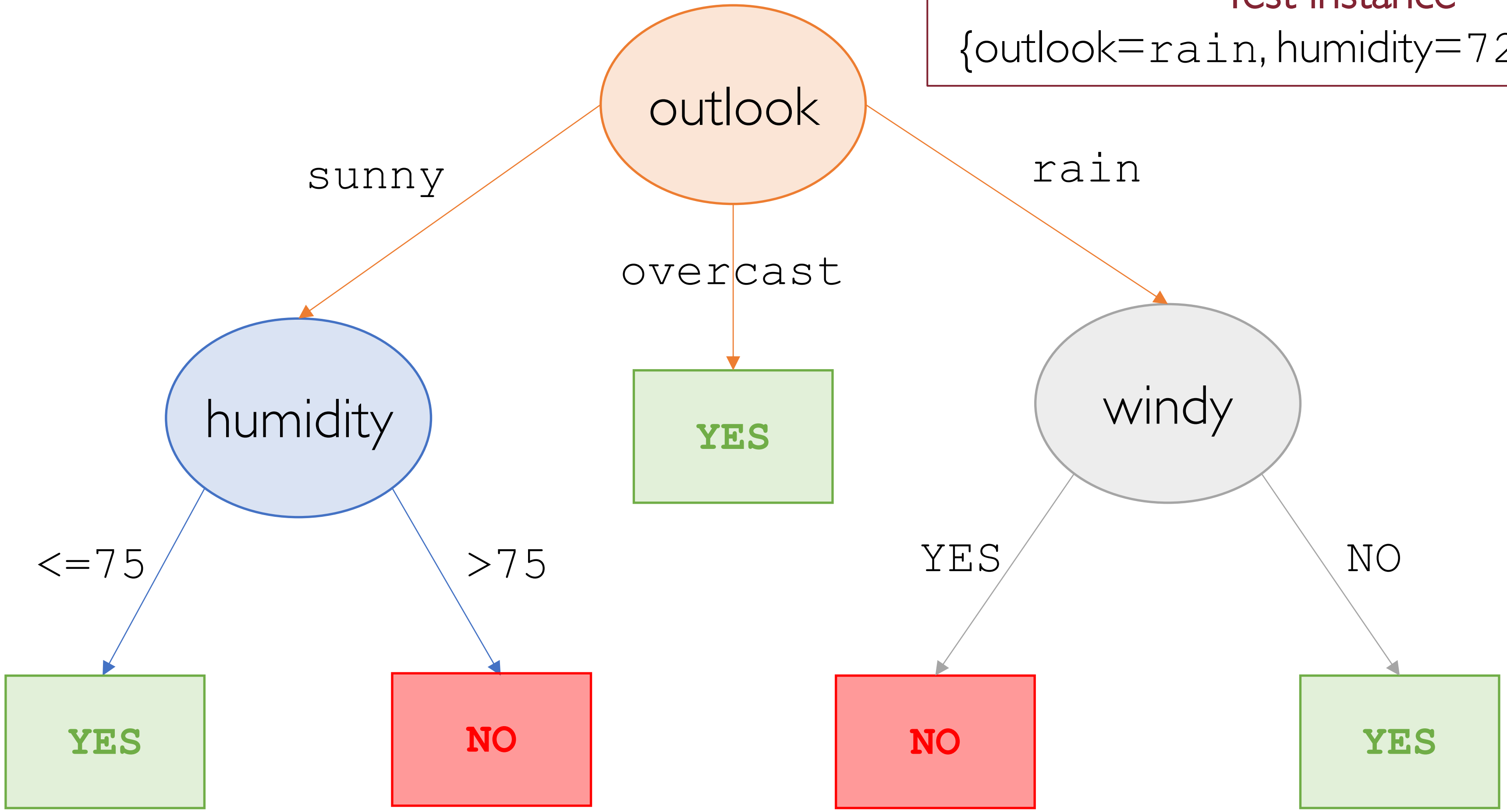
Decision Tree: An Example

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



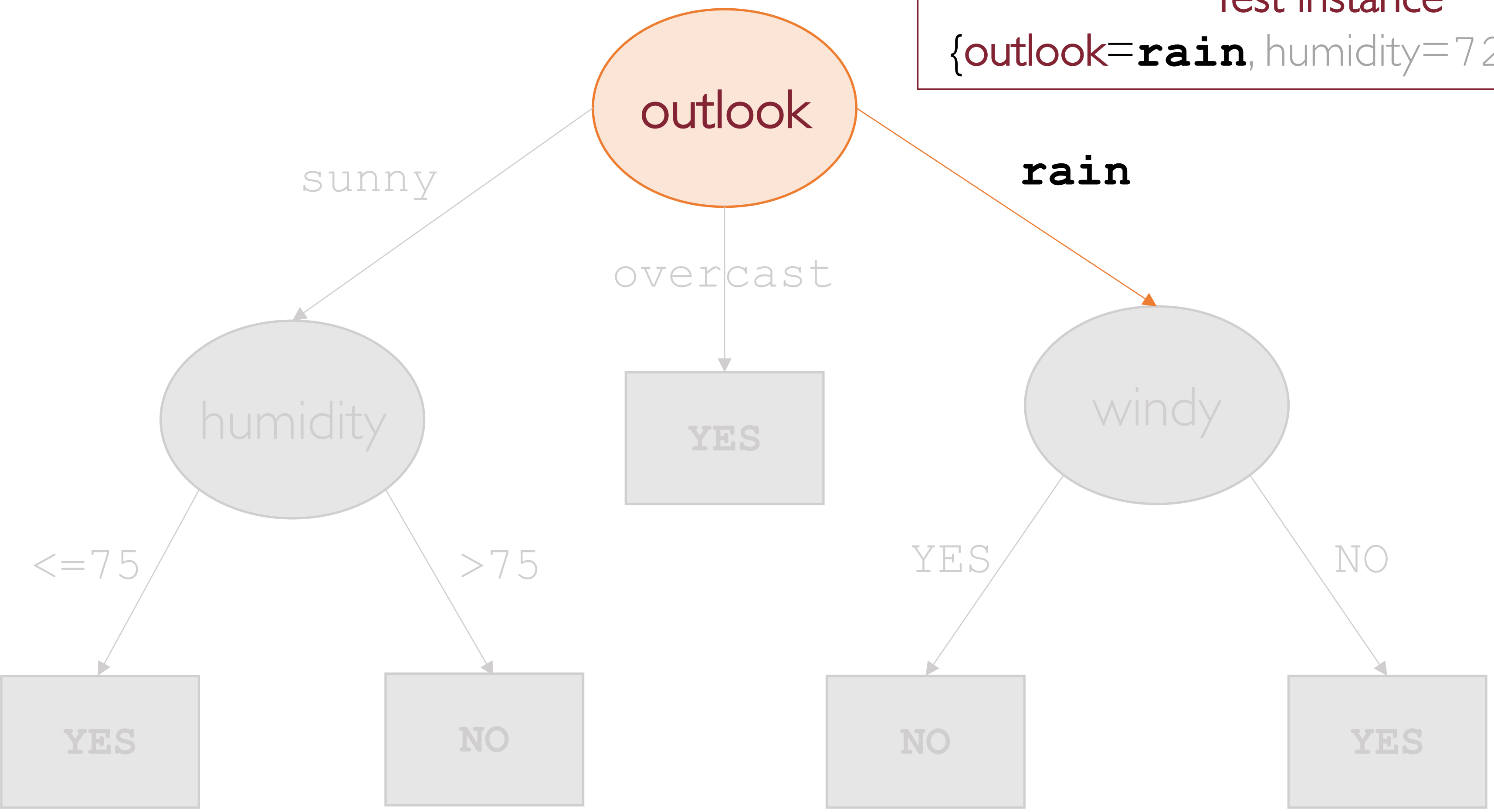
Decision Tree: An Example

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



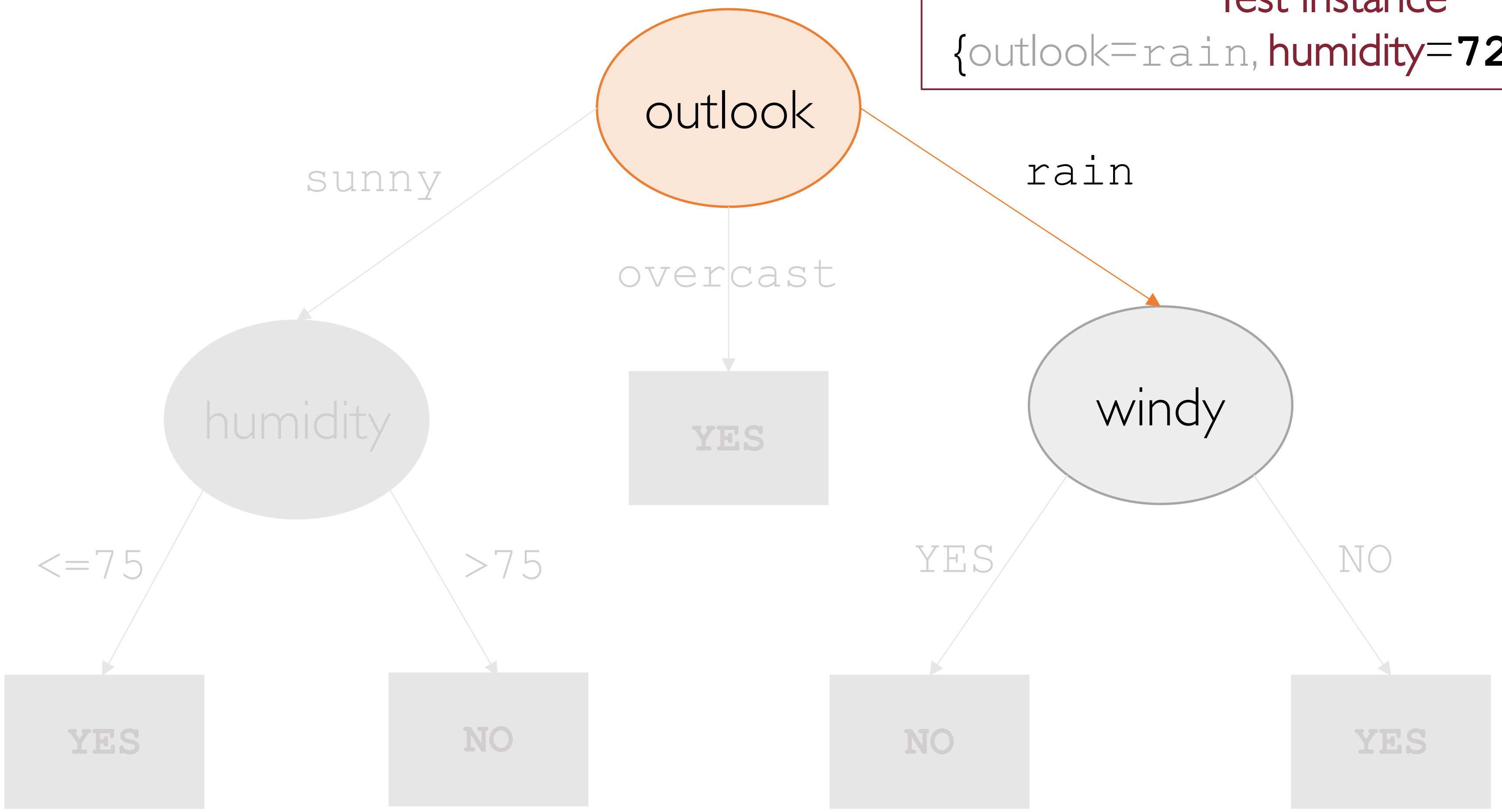
Decision Tree: An Example

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



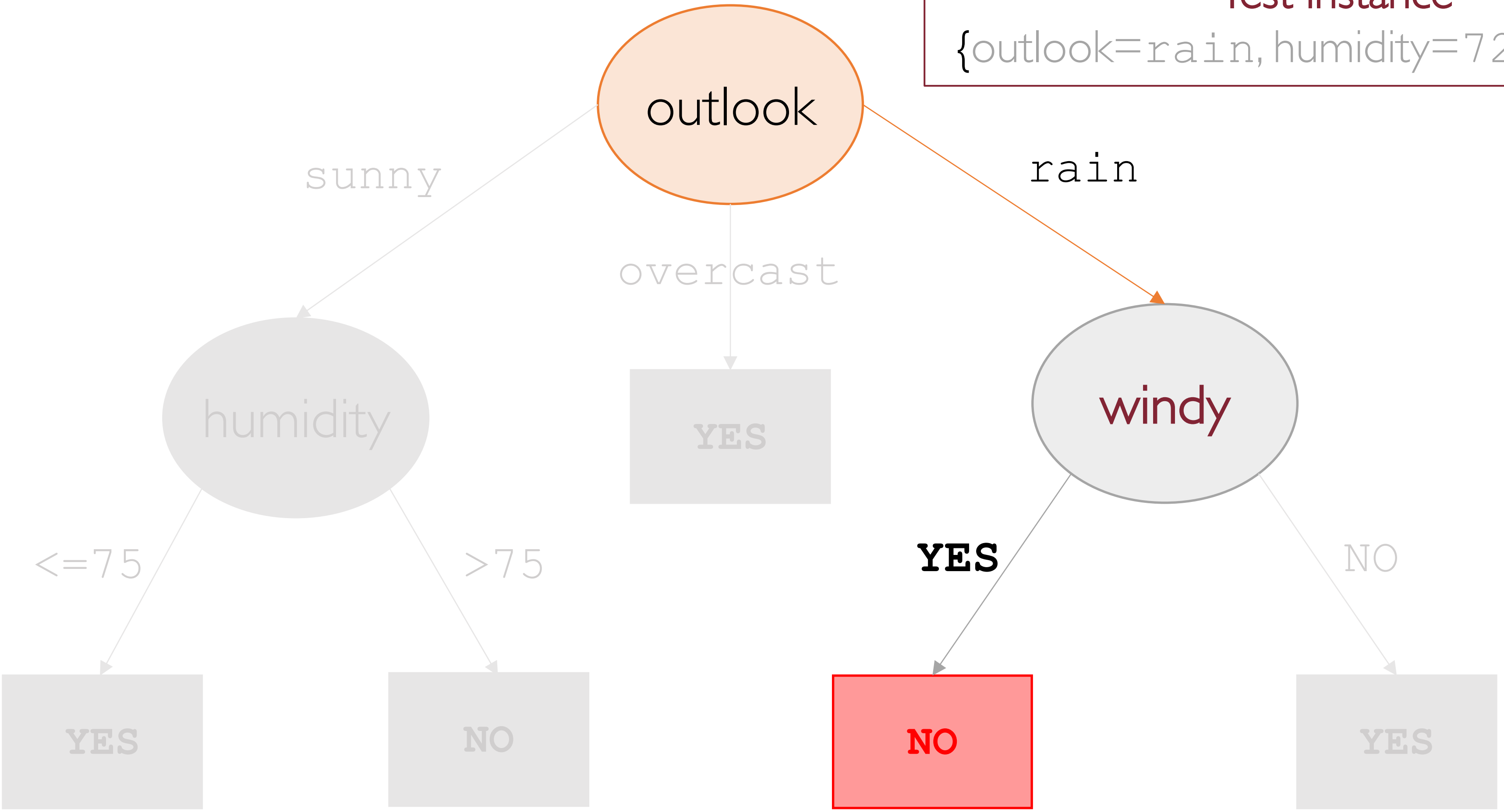
Decision Tree: An Example

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
$\mathcal{Y} \subseteq \mathbb{R}$	real-value label (regression)
$\mathcal{Y} = \{1, \ldots, k\}$	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 $R_1, R_2, \ldots, R_J$
- 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

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
- 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 $\mathbf{x}$ 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

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

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)

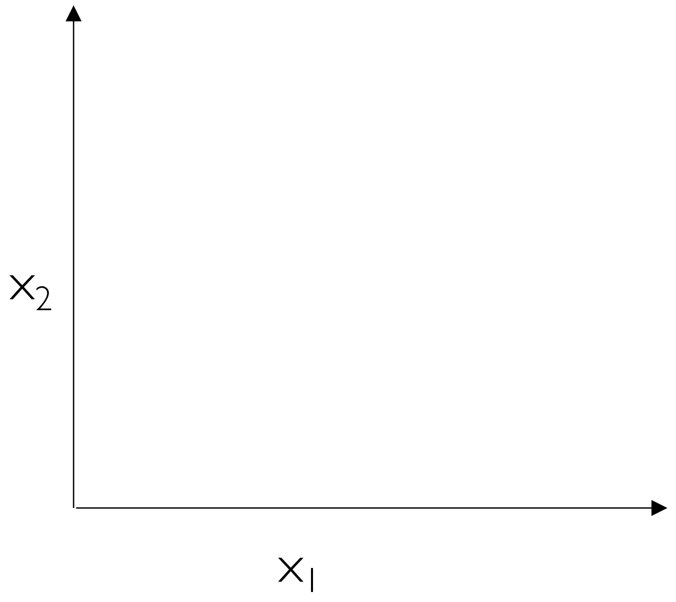
One branch for each value

Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

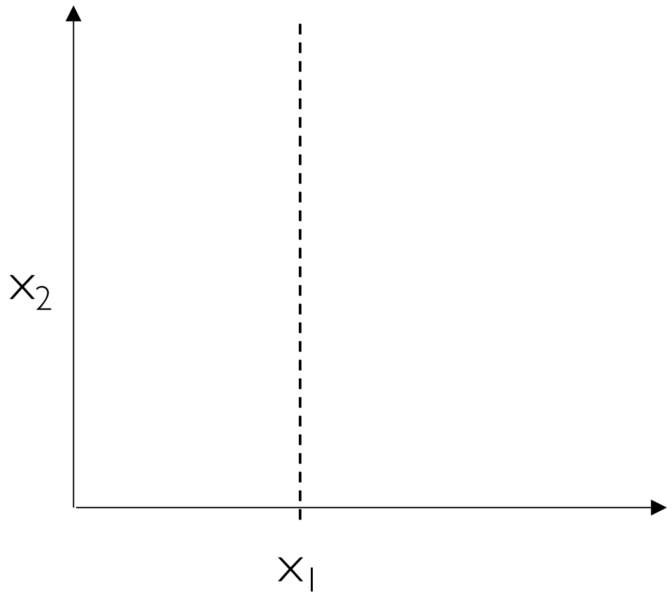
One branch for each value



Discrete vs. Continuous Inputs

Discrete Inputs
(e.g., boolean)

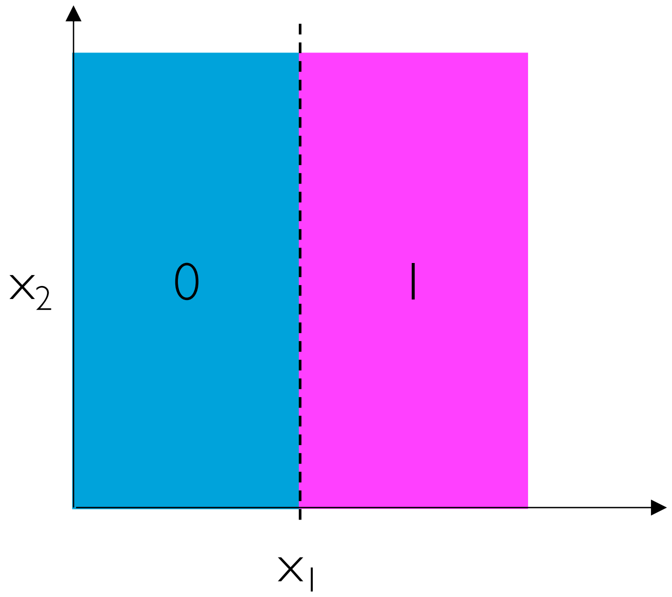
One branch for each value



Discrete vs. Continuous Inputs

Discrete Inputs
(e.g., boolean)

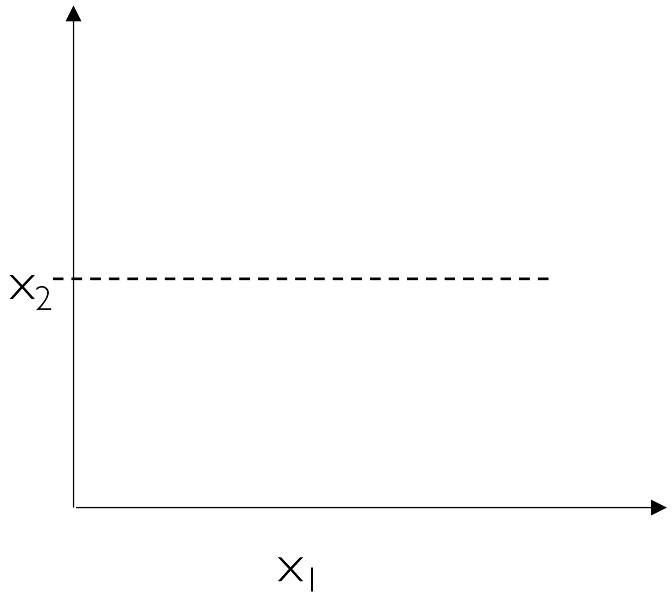
One branch for each value



Discrete vs. Continuous Inputs

Discrete Inputs
(e.g., boolean)

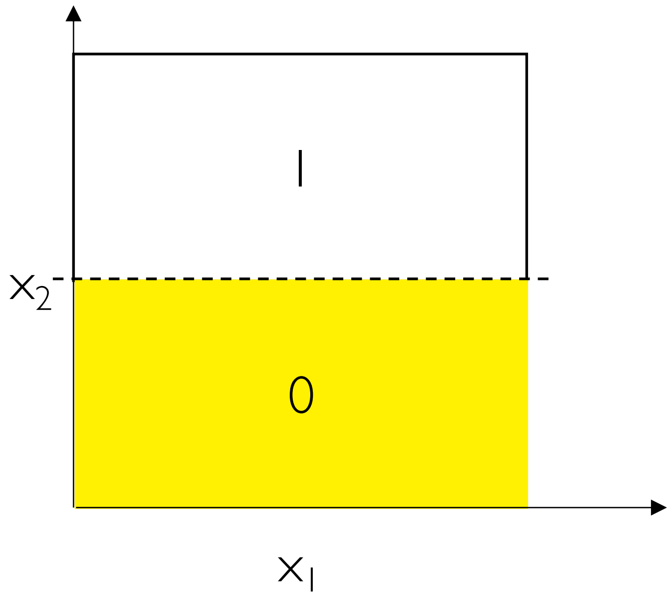
One branch for each value



Discrete vs. Continuous Inputs

Discrete Inputs
(e.g., boolean)

One branch for each value

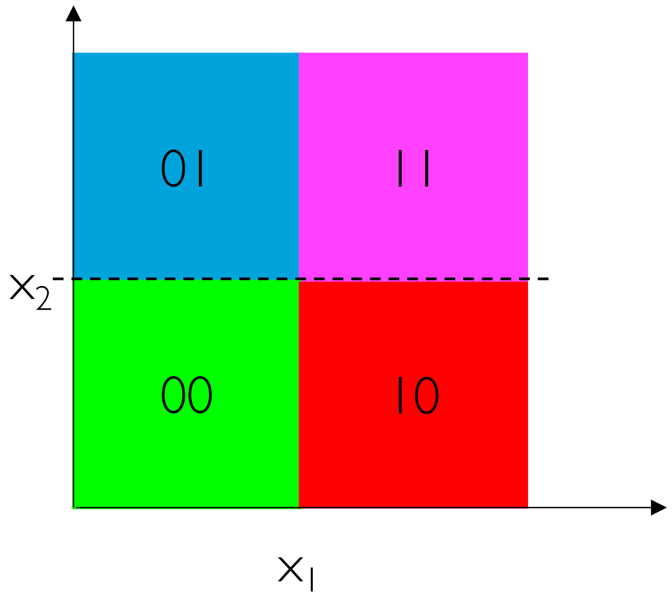


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

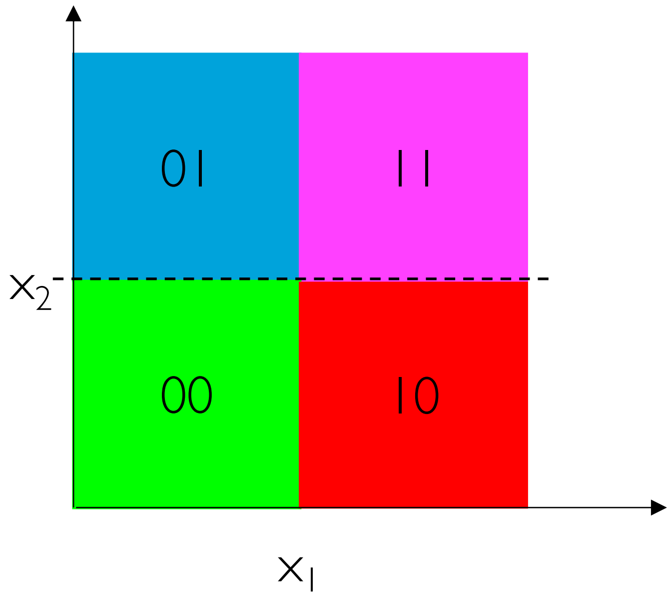


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

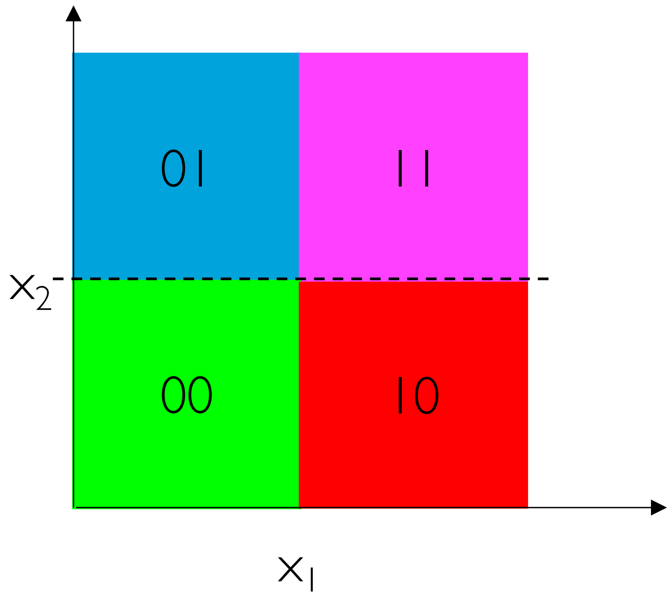


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

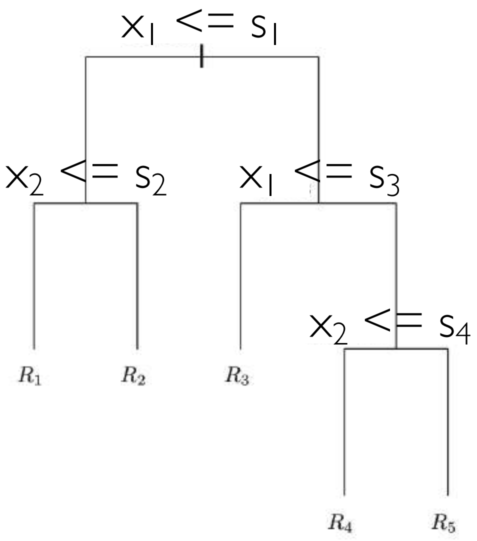


Continuous Inputs

For each attribute, find a splitting point s

Test x_j <= s and create 2 branches

The same attribute may be further split in each subtree

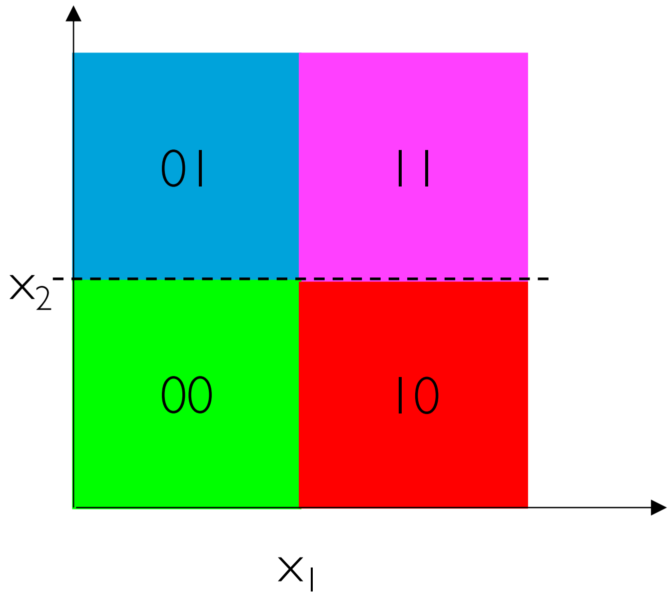


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

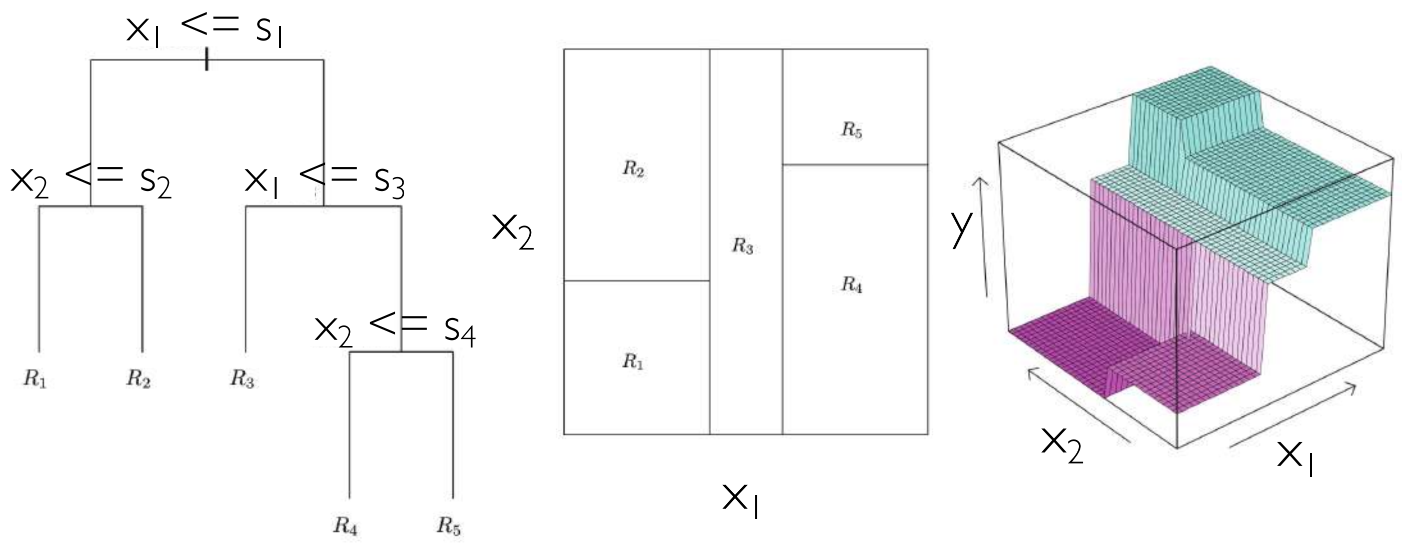


Continuous Inputs

For each attribute, find a splitting point s

Test $x_j <= s$ and create 2 branches

The same attribute may be further split in each subtree

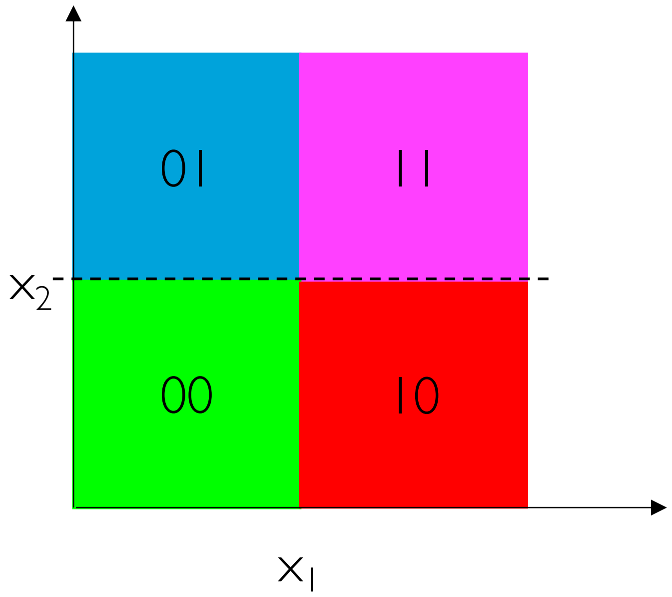


Discrete vs. Continuous Inputs

Discrete Inputs

(e.g., boolean)

One branch for each value

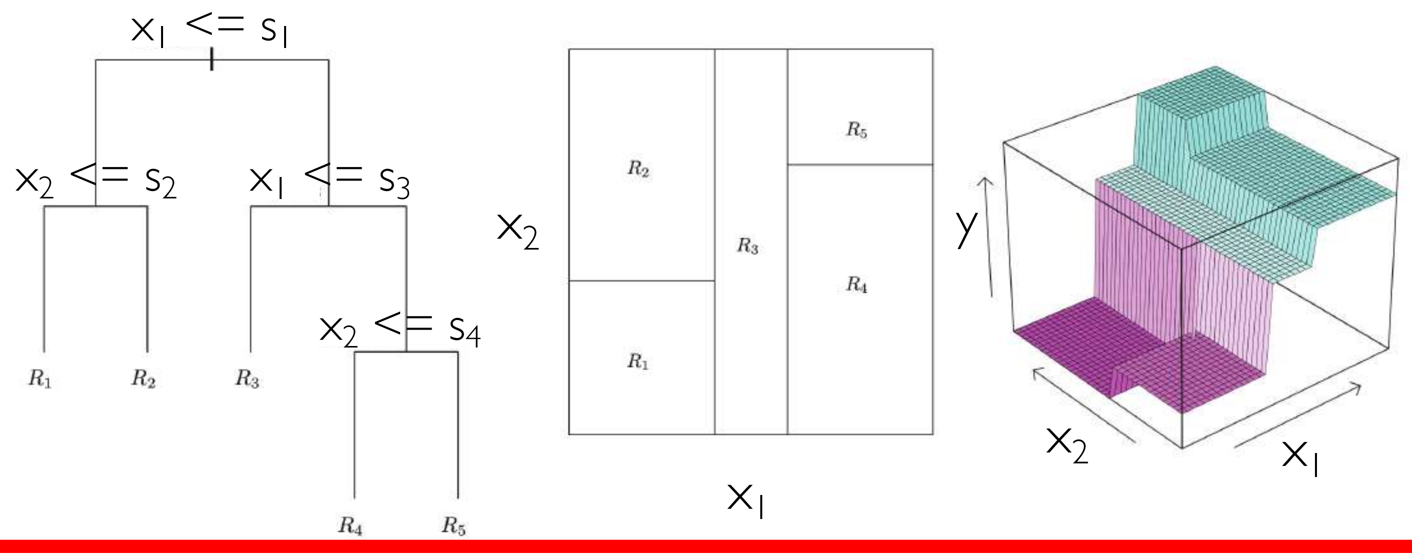


Continuous Inputs

For each attribute, find a splitting 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

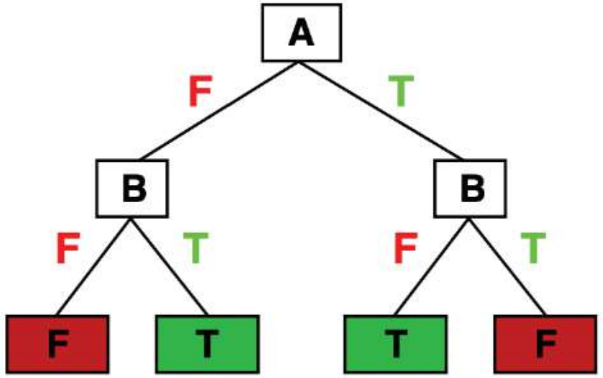
Expressiveness

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

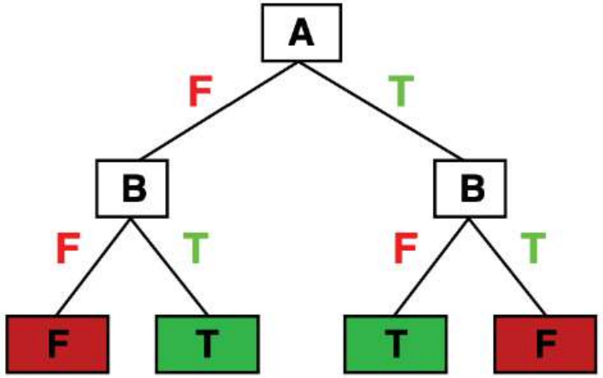
Discrete Inputs/Discrete Outputs

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



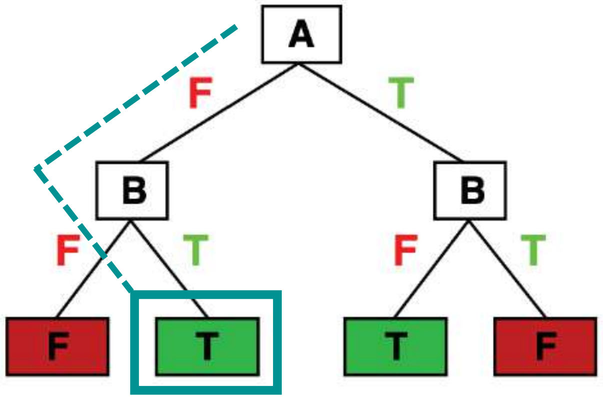
Expressiveness

Discrete Inputs/Discrete Outputs

Decision Trees can express any function of the inputs

Example: 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

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

2^n rows

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:

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

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

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

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

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 training 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 training heuristic
- Regression Trees:
  - Use Residual Sum of Squares (RSS) as splitting criterion
  - At inference time, predictions are the mean of the leaf observations