Decision Trees

Lecture 8

Today: Learning Objectives

1. Understand Decision Trees
2. Discover entropy and information gain
3. Discuss decision tree limitations
4. Learn about an application in XBOX


1. Let’s talk about Tree (using Cars!)

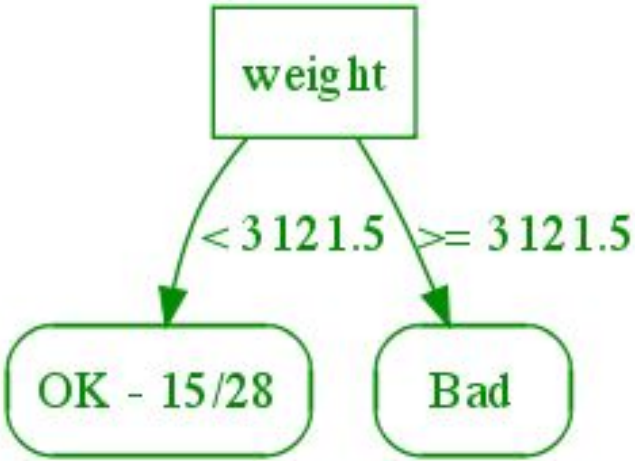
The car dataset

cylinders	displacement	horsepower	weight	acceleration	year	maker	mpg
4	112	88	2395	18	82	America	Good
4	135	84	2370	13	82	America	Bad
4	135	84	2370	13	82	America	Good
4	91	68	1970	17.6	82	Europe	Good
4	105	63	2125	14.7	82	America	Bad
4	105	63	2125	14.7	82	America	Good
6	146	120	2930	13.8	81	Europe	OK
4	156	92	2620	14.4	81	America	OK
4	81	60	1760	16.1	81	Europe	Good
4	140	88	2870	18.1	80	America	OK
8	305	130	3840	15.4	79	America	Bad
6	200	85	2990	18.2	79	America	Bad

Full dataset link here!

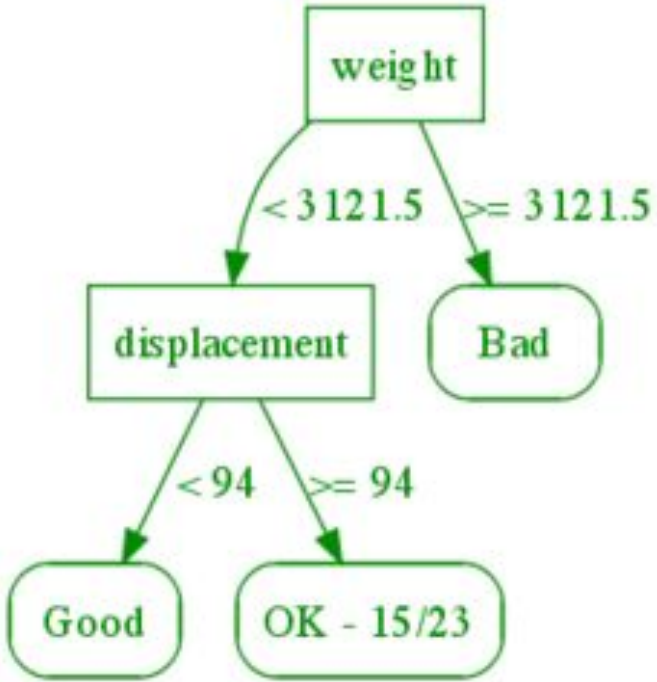
Split data into a tree structure

- Let’s split the dataset on how heavy the car is.
- The root note represents entire dataset (19 bad, 15 okay, and 8 good examples)
- The leaves have some subset of bad/okay/good
- Some leaves are **pure** (all bad, okay, or good)
- Some leaves are **mixed** → split further



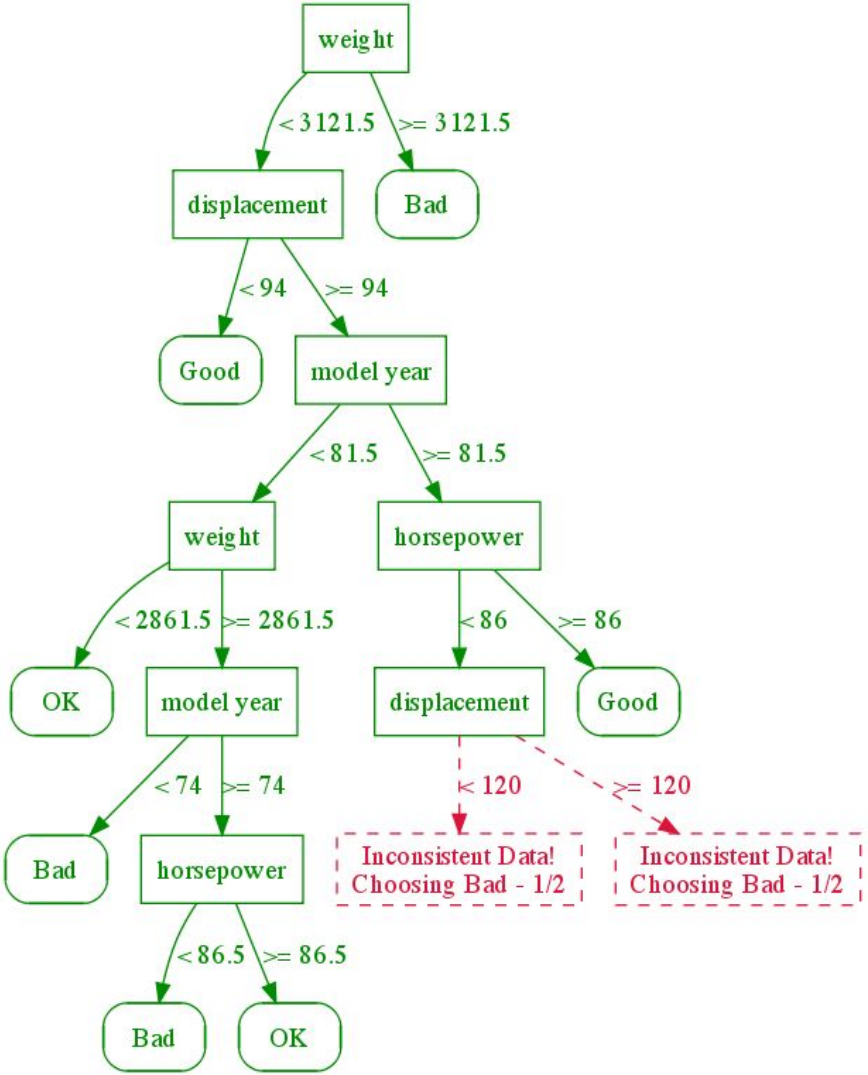
Second Split

- Let’s split the dataset on how displacement
- A leaf is now pure (good!)
- Another leaf is still mixed → split further



Keep Splitting

- Keep on splitting until all leaves are pure
- Some time this actually might not be possible (see the red leaves 1 out of 2 examples is bad)
- We can use this organization of the data as a classifier
- Given a test example, we traverse the tre from root to the leaf and return the leaf label → **decision tree!**



Decision Tree

Like SVMs, Decision Tree (DT) is a versatile ML algorithm for classification and regression (Recall **DecisionTreeRegressor**)

Also, it is a fundamental component of **Random Forest** (later)

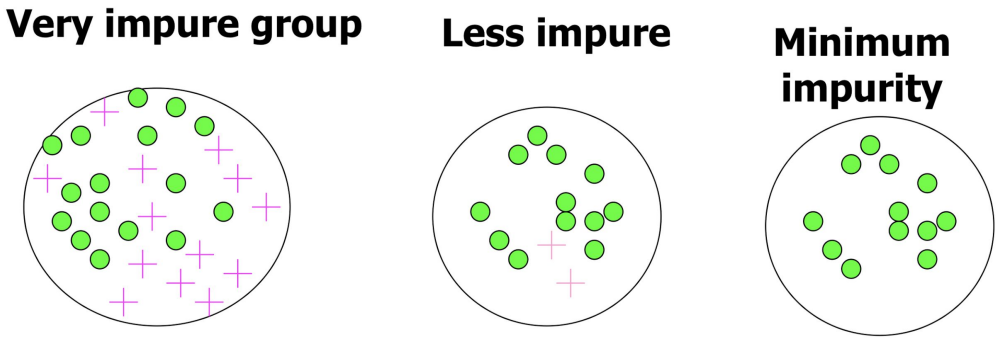
We will discuss how to train, visualize, and predict with Decision Trees, including Classification and Regression Tree (**CART)**.

Choosing what feature to split on

A node is “pure” if all samples it applies to belong to the same class (or its impunity = 0). Otherwise its impurity (**Gini index**) is computed as:

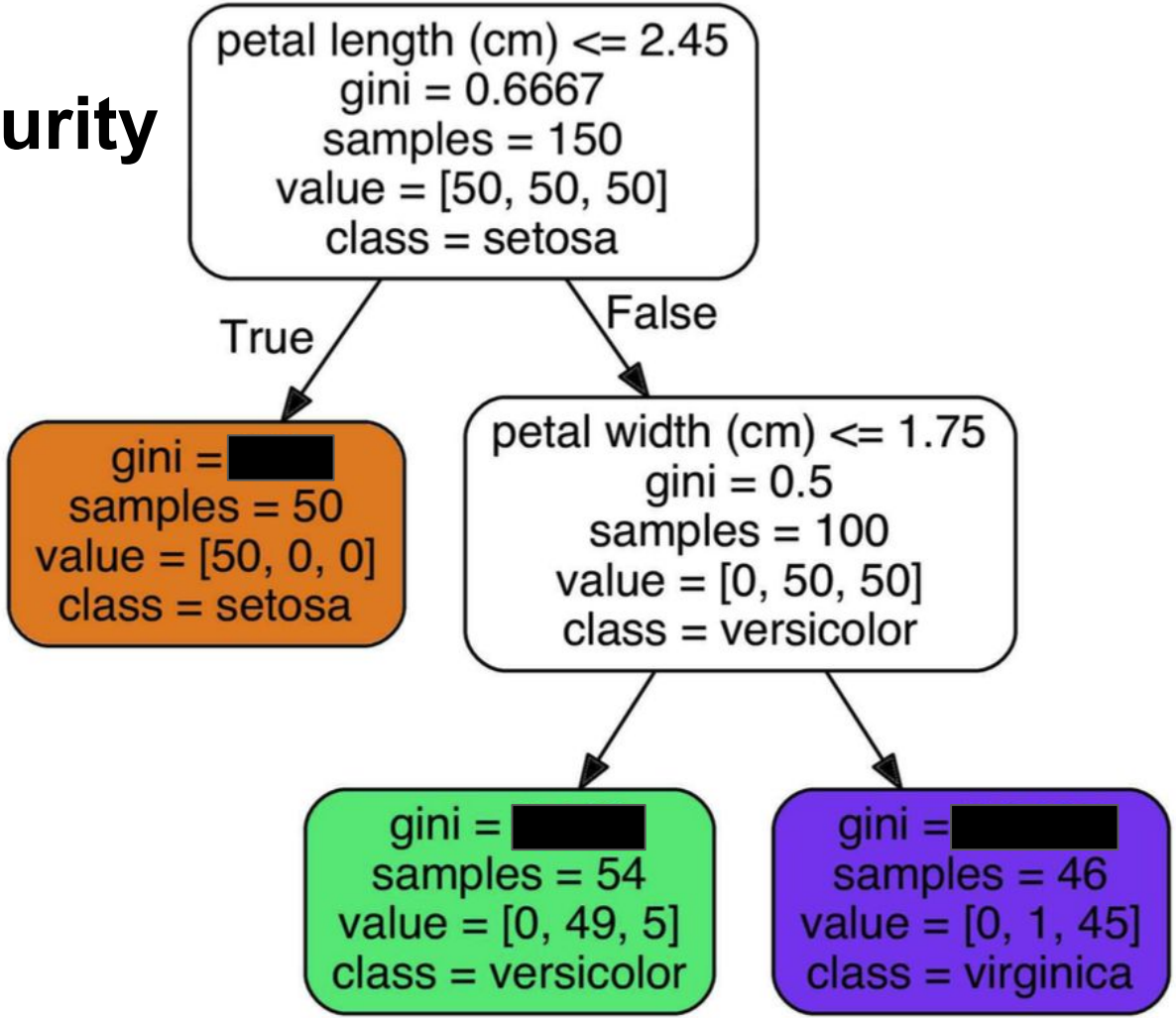
$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

$p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node.

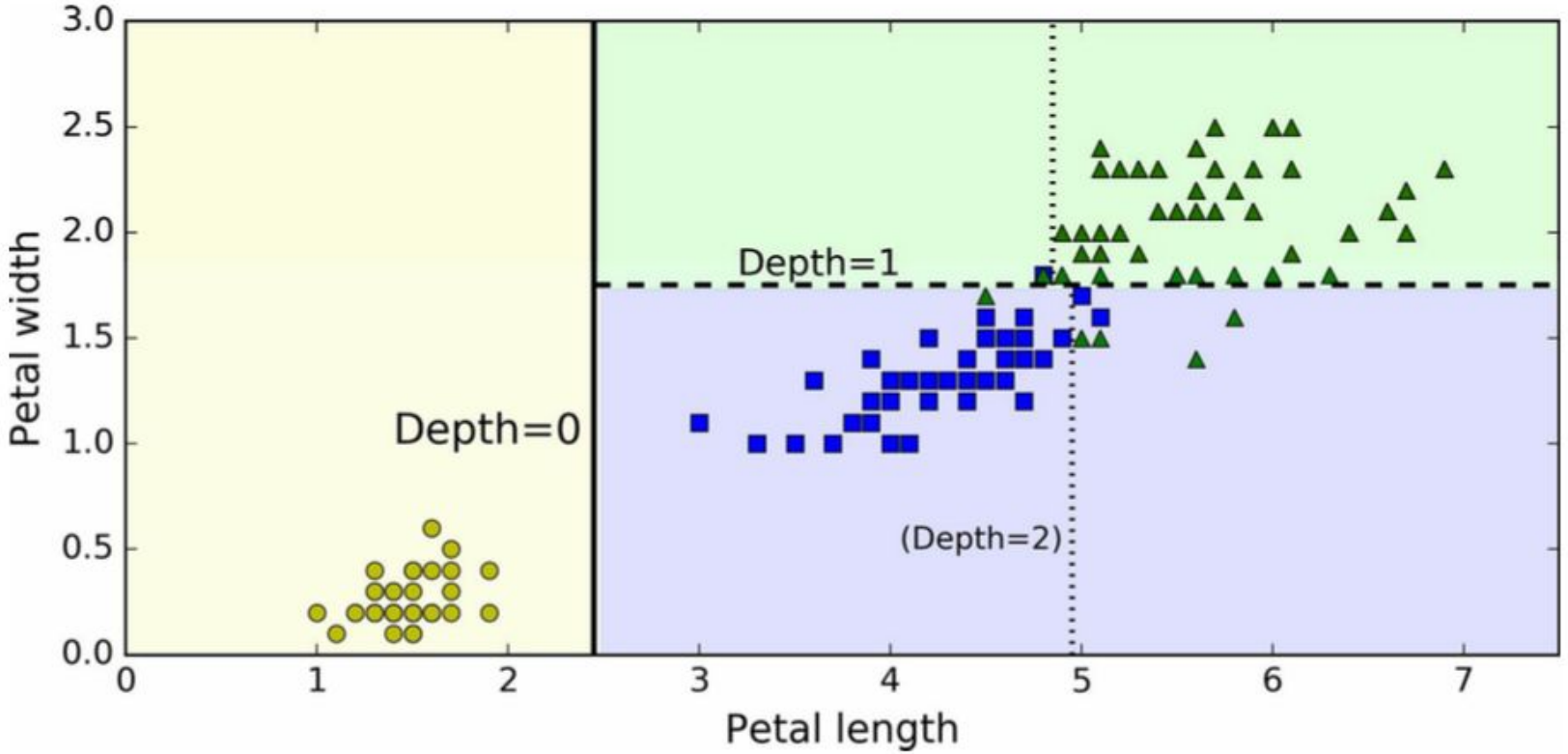


Compute Gini Impurity

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$



Decision Boundary



Model Interpretation

WHITE BOX

- Like Decision Tree
- Fairly Intuitive and easy to interpret the prediction
- Provide nice and simple classification rules

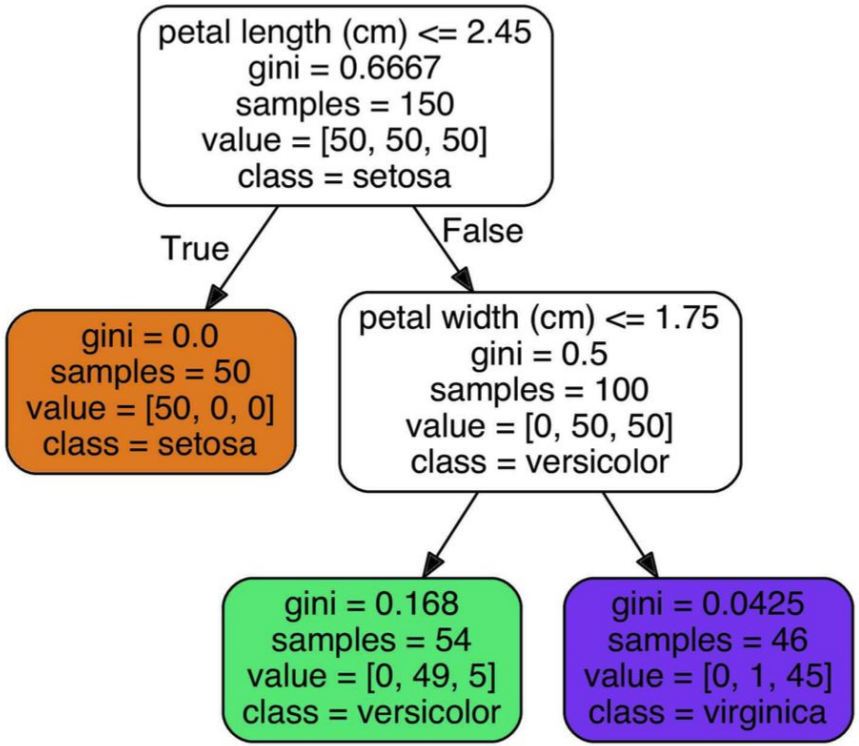
BLACK BOX

- Like Random Forest or Neural Network (later)
- Hard to explain in simple terms why the predictions are made
- Difficulty to know which features actually contribute to the model

Estimating Class Probability

Predict by estimating probability that an sample belong to a particular class `k`

Traverse the tree to leaf node, then return the ratio of training samples of class `k`




```
tree_clf.predict_proba([[5, 1.5]])
```

```
array([[ 0.        ,  0.90740741,  0.09259259]])
```

```
tree_clf.predict([[5, 1.5]])
```

```
array([1])
```

The CART Training Algorithm

The Classification and Regression Tree (CART) algorithm to train Decision Trees

- Search for the pair of a single feature and its threshold (k, t_k) that produce the subset with **lowest Gini index:**

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

$$\text{where} \begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset.} \end{cases}$$

- Split training set into two subsets using feature k and threshold t_k
- Split the subset using the same logic, stop once reaches max depth

2. Entropy, anyone?

DESK ENTROPY

Definition

Desk entropy is a spatiodynamic quantity that measures a workspace's degree of disorder, and the inability to find anything when you really need it.

Any spontaneous activity, whether productive or unproductive, disperses matter and increases overall desk entropy.

Efforts to reverse desk entropy are temporary, and inevitably decrease over time.

Entropy

Entropy is a measure of **disorderness**: it is zero when well-order and identical

A set's entropy is zero when it contains instances of only one class.

Entropy is calculated as H_i:

$$H_i = -\sum_{k=1}^{n} p_{i,k} \log(p_{i,k}) \qquad \text{Vs. Gini} \qquad G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

- Both Entropy and Gini produce similar tree most of the time, but Gini is slightly faster.
- Gini is for numerical features while Entropy is for categorical ones.
- Gini tends to isolate the **largest** class from other classes while Entropy tends to find group that made up ~50% of the data → **more balanced tree**.

Entropy Interpretation

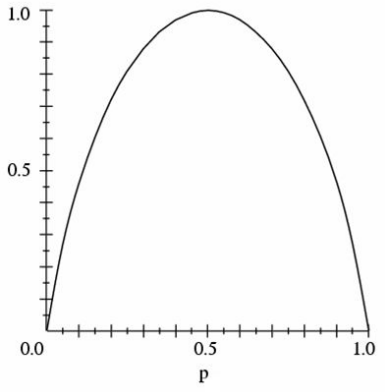
For binary Y, the entropy is a function of `p(y)`:

$$H(Y) = -\sum_{y \in Y} p(y) \log p(y)$$

Note: entropy is 0 when `p=1 or p=0` (all are of the same class)

Entropy is 1 when `p=0.5` (50% in either class)

Another interpretation of entropy is the expected number of bits needed to encode Y. For example, an unbiased coin requires 1 bit, an eight sided dice requires 3 bits.



What about a biased coin that always comes up head?

Information Gain

To measure the reduction in entropy of Y from knowing X, we use Information Gain:

$$\mathrm{IG}\,(Y, X) = H(Y) - H(Y|X)$$

$$\text{where } H(Y|X) = \sum_{x \in X} p(x) H(Y|X = x)$$

To determine which feature in a given set of feature vectors is most useful to discriminating between the classes

We will use IG to decide the ordering of features in the nodes of a decision tree.

Example: when to playing tennis?

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	***No***
Sunny	Hot	High	True	***No***
Overcast	Hot	High	False	***Yes***
Rainy	Mild	High	False	***Yes***
Rainy	Cool	Normal	False	***Yes***
Rainy	Cool	Normal	True	***No***
Overcast	Cool	Normal	True	***Yes***
Sunny	Mild	High	False	***No***
Sunny	Cool	Normal	False	***Yes***
Rainy	Mild	Normal	False	***Yes***
Sunny	Mild	Normal	True	***Yes***
Overcast	Mild	High	True	***Yes***
Overcast	Hot	Normal	False	***Yes***
Rainy	Mild	High	True	***No***

Example

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	***No***
Sunny	Hot	High	True	***No***
Overcast	Hot	High	False	***Yes***
Rainy	Mild	High	False	***Yes***
Rainy	Cool	Normal	False	***Yes***
Rainy	Cool	Normal	True	***No***
Overcast	Cool	Normal	True	***Yes***
Sunny	Mild	High	False	***No***
Sunny	Cool	Normal	False	***Yes***
Rainy	Mild	Normal	False	***Yes***
Sunny	Mild	Normal	True	***Yes***
Overcast	Mild	High	True	***Yes***
Overcast	Hot	Normal	False	***Yes***
Rainy	Mild	High	True	***No***

$$\mathrm{H}(Y) = -\sum_{i=1}^{K} p_k \log_2 p_k$$

$$= -\frac{5}{14}\log_2\frac{5}{14} - \frac{9}{14}\log_2\frac{9}{14}$$

$$= 0.94$$

Example

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	*No*
Sunny	Hot	High	True	*No*
Overcast	Hot	High	False	*Yes*
Rainy	Mild	High	False	*Yes*
Rainy	Cool	Normal	False	*Yes*
Rainy	Cool	Normal	True	*No*
Overcast	Cool	Normal	True	*Yes*
Sunny	Mild	High	False	*No*
Sunny	Cool	Normal	False	*Yes*
Rainy	Mild	Normal	False	*Yes*
Sunny	Mild	Normal	True	*Yes*
Overcast	Mild	High	True	*Yes*
Overcast	Hot	Normal	False	*Yes*
Rainy	Mild	High	True	*No*

$$InfoGain(\text{Humidity}) = H(Y) - \frac{m_L}{m} H_L - \frac{m_R}{m} H_R$$
$$= 0.94 - \frac{7}{14} H_L - \frac{7}{14} H_R$$

Example

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	***No***
Sunny	Hot	High	True	***No***
Overcast	Hot	High	False	***Yes***
Rainy	Mild	High	False	***Yes***
Rainy	Cool	**Normal**	False	***Yes***
Rainy	Cool	**Normal**	True	***No***
Overcast	Cool	**Normal**	True	***Yes***
Sunny	Mild	High	False	***No***
Sunny	Cool	**Normal**	False	***Yes***
Rainy	Mild	**Normal**	False	***Yes***
Sunny	Mild	**Normal**	True	***Yes***
Overcast	Mild	High	True	***Yes***
Overcast	Hot	**Normal**	False	***Yes***
Rainy	Mild	High	True	***No***

$$InfoGain(\text{Humidity}) = H(Y) - \frac{m_L}{m} H_L - \frac{m_R}{m} H_R$$

$$0.94 - \frac{7}{14} H_L - \frac{7}{14} H_R$$

$$H_L = -\frac{6}{7}\log_2\frac{6}{7} - \frac{1}{7}\log_2\frac{1}{7}$$

Example

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	**High**	False	*No*
Sunny	Hot	**High**	True	*No*
Overcast	Hot	**High**	False	*Yes*
Rainy	Mild	**High**	False	*Yes*
Rainy	Cool	Normal	False	***Yes***
Rainy	Cool	Normal	True	***No***
Overcast	Cool	Normal	True	***Yes***
Sunny	Mild	**High**	False	*No*
Sunny	Cool	Normal	False	***Yes***
Rainy	Mild	Normal	False	***Yes***
Sunny	Mild	Normal	True	***Yes***
Overcast	Mild	**High**	True	*Yes*
Overcast	Hot	Normal	False	***Yes***
Rainy	Mild	**High**	True	*No*

$$InfoGain(\text{Humidity}) = H(Y) - \frac{m_L}{m} H_L - \frac{m_R}{m} H_R$$

$$0.94 - \frac{7}{14} H_L - \frac{7}{14} H_R$$

$$H_L = -\frac{6}{7}\log_2\frac{6}{7} - \frac{1}{7}\log_2\frac{1}{7} = 0.592$$

$$H_R = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.985$$

Example

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	***No***
Sunny	Hot	High	True	***No***
Overcast	Hot	High	False	***Yes***
Rainy	Mild	High	False	***Yes***
Rainy	Cool	Normal	False	***Yes***
Rainy	Cool	Normal	True	***No***
Overcast	Cool	Normal	True	***Yes***
Sunny	Mild	High	False	***No***
Sunny	Cool	Normal	False	***Yes***
Rainy	Mild	Normal	False	***Yes***
Sunny	Mild	Normal	True	***Yes***
Overcast	Mild	High	True	***Yes***
Overcast	Hot	Normal	False	***Yes***
Rainy	Mild	High	True	***No***

$$InfoGain(\text{Humidity}) = H(Y) - \frac{m_L}{m} H_L - \frac{m_R}{m} H_R$$

$$0.94 - \frac{7}{14} 0.592 - \frac{7}{14} 0.985$$

$$= 0.94 - 0.296 - 0.4925$$

$$= 0.1515$$

Calculating for each feature:

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	***No***
Sunny	Hot	High	True	***No***
Overcast	Hot	High	False	***Yes***
Rainy	Mild	High	False	***Yes***
Rainy	Cool	Normal	False	***Yes***
Rainy	Cool	Normal	True	***No***
Overcast	Cool	Normal	True	***Yes***
Sunny	Mild	High	False	***No***
Sunny	Cool	Normal	False	***Yes***
Rainy	Mild	Normal	False	***Yes***
Sunny	Mild	Normal	True	***Yes***
Overcast	Mild	High	True	***Yes***
Overcast	Hot	Normal	False	***Yes***
Rainy	Mild	High	True	***No***

`IG(Humidity) = 0.1515`

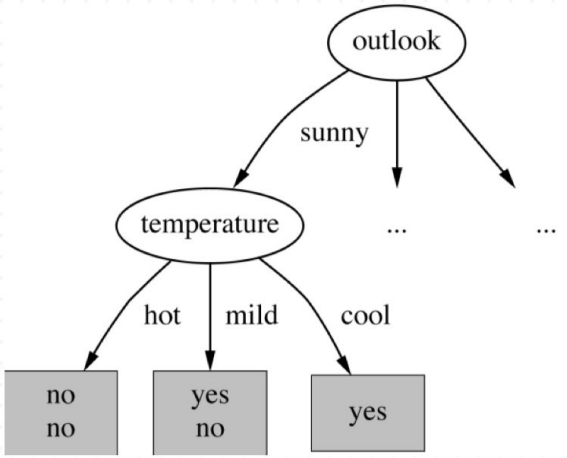
`IG(Outlook) = 0.247`

`IG(Temperature) = 0.029`

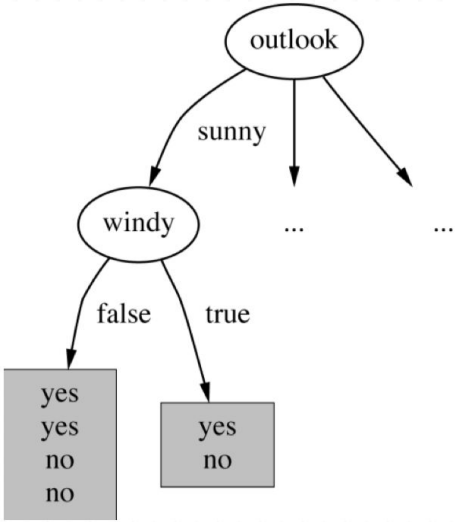
`IG(Windy) = 0.048`

→ Initial split is on **Outlook** because it is the feature with the highest `IG`.

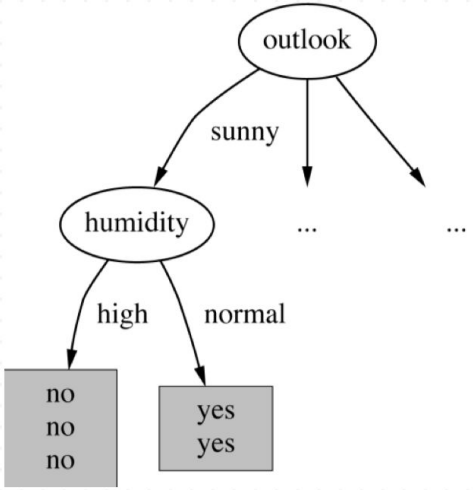
Search for best split at the next level:



Temperature = 0.571

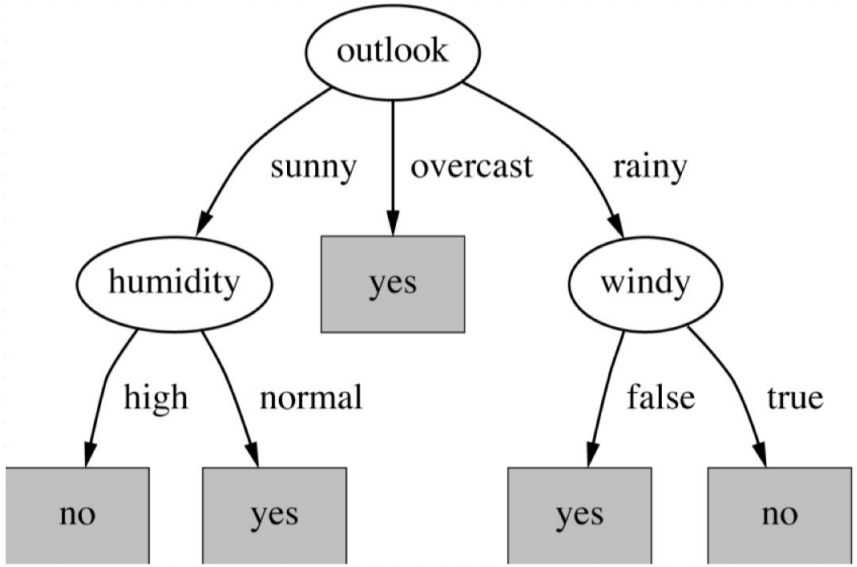


Windy = 0.020



Humidity = 0.971

The final decision tree



Note that not all leaves need to be pure; sometimes similar (even identical) instances have different classes. Splitting stops when data cannot be split any further.

Computational Complexity

- Finding the optimal trees is known as a NP-Complete problem, so we must settle for a “reasonably good” solution.
- Traversing requires roughly $O(\log(m))$ independent of number of features → very fast
- Training comparing all features on all samples at each node $O(nm \log(m))$ → slow down considerably for large training set.

What makes a good tree?

- **Not too small:** need to handle important but subtle distinction in data
- **Not too big:** avoid overfitting training examples
- **Occam's Razor:** find the simplest model (smallest tree) that fits the observations
- **Inductive bias:** small trees with informative nodes near the root
- In practice, you can **regularizes** the construction process to get small but effective tree

Regularization Hyperparameters

Left unconstrained, decision tree will most likely overfitting to the training data.

Need to restrict the tree degree of freedom → **regularization**

- **max_depth**: the maximum depth of the DT
- **min_sample_split**: the minimum number of samples a node must have to split
- **min_sample_leaf**: the minimum number of samples a leaf node must have
- **max_leaf_nodes:** maximum number of leaf nodes
- **max_features:** maximum number features that are evaluated for splitting

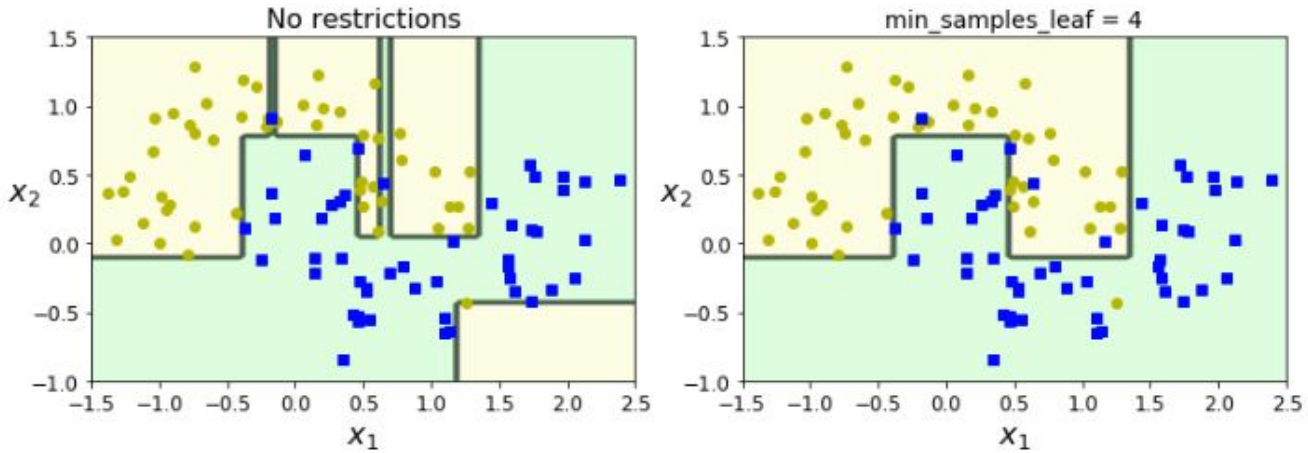
Training a Decision Tree in Python

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X, y)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=42,
            splitter='best')
```

Code Demo

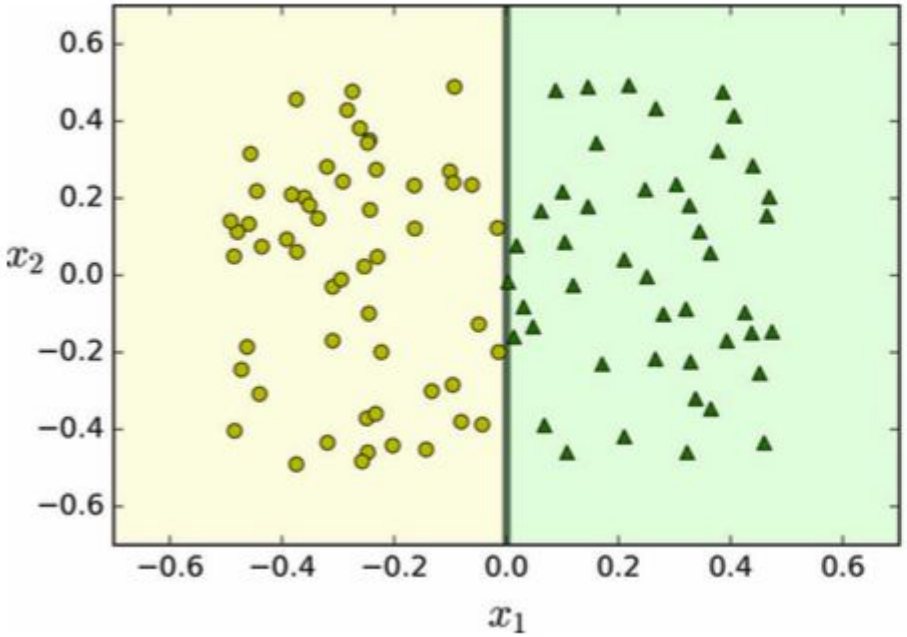
```
from sklearn.datasets import make_moons
Xm, ym = make_moons(n_samples=100, noise=0.25, random_state=53)

deep_tree_clf1 = DecisionTreeClassifier(random_state=42)
deep_tree_clf2 = DecisionTreeClassifier(min_samples_leaf=4, random_state=42)
deep_tree_clf1.fit(Xm, ym)
deep_tree_clf2.fit(Xm, ym)
```

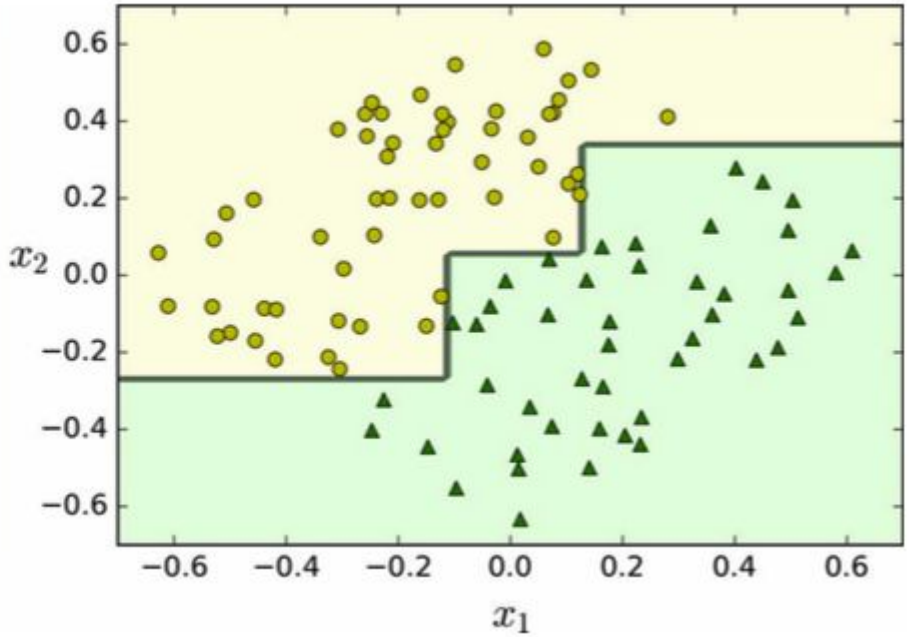


3. Limitations

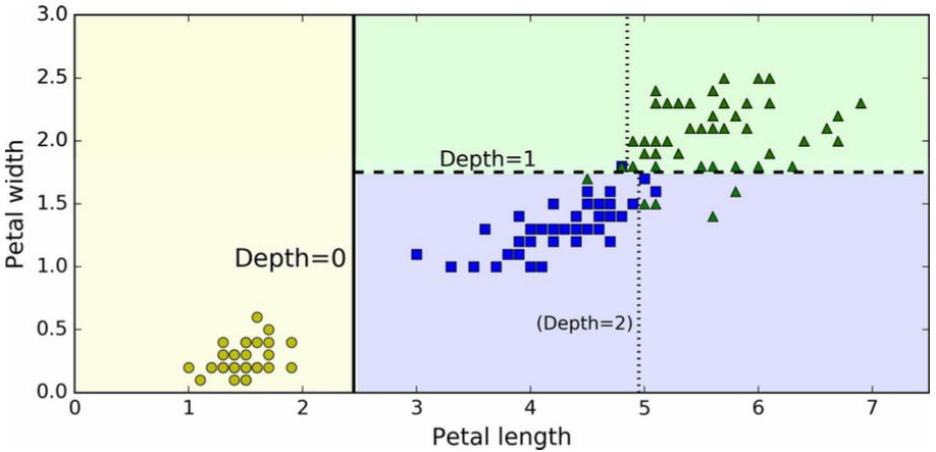
Sensitivity to Training Set

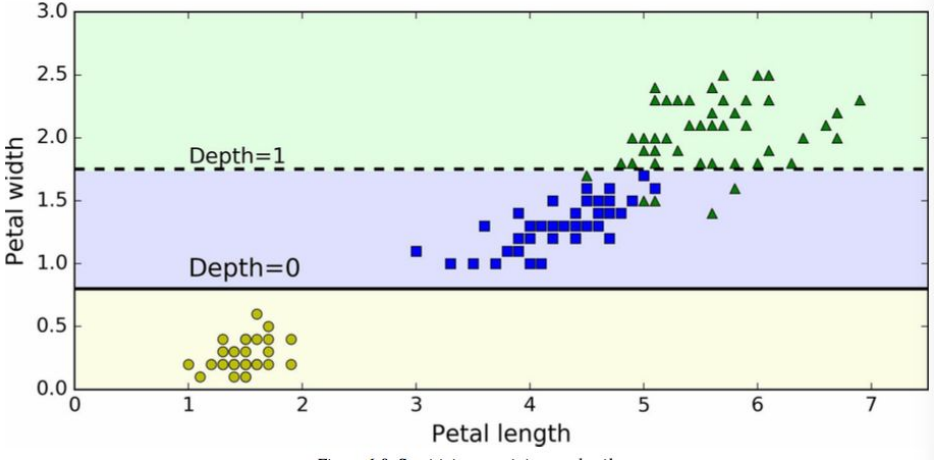
DT loves **orthogonal** decision boundary → sensitive to training set rotation





Instability to small variation of training data





Random Forest might help! (coming soon)

Summary of Decision Trees

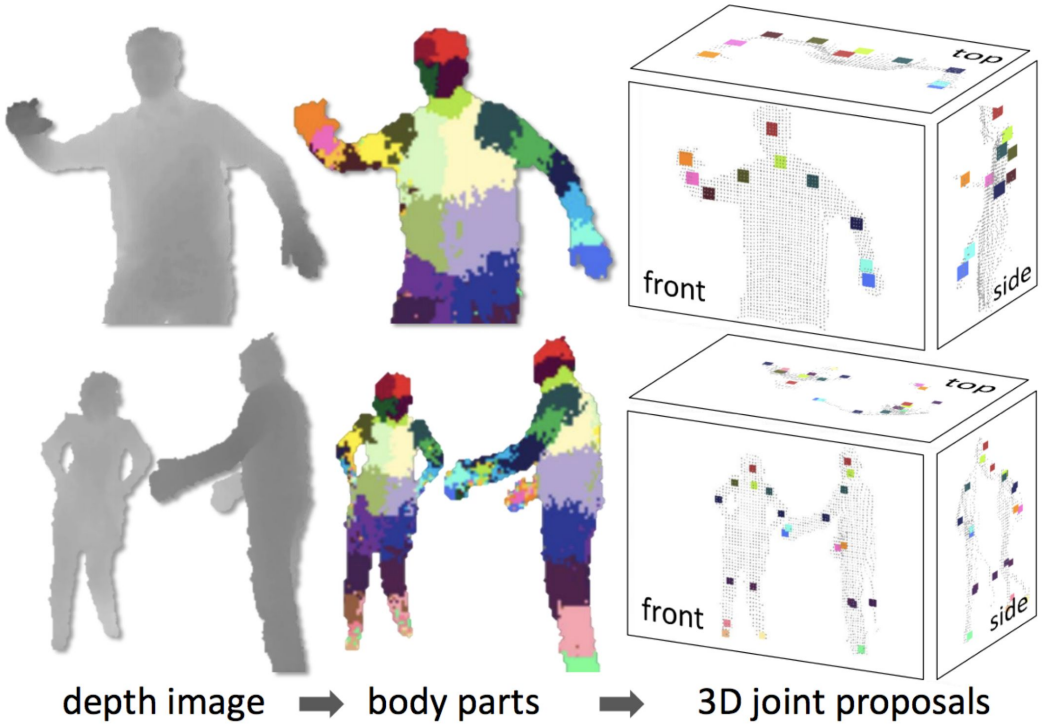
+ Easy to understand (by human)
+ Computationally efficient
+ Handle both numerical and categorical data
+ Parametric thus compact: do not have to carry training data around
+ Building block for various ensemble methods

- Heuristic training techniques
- Finding partition of space that minimizes error is NP-hard
- Use greedy approaches with limited theoretical work

4. Applications: XBox

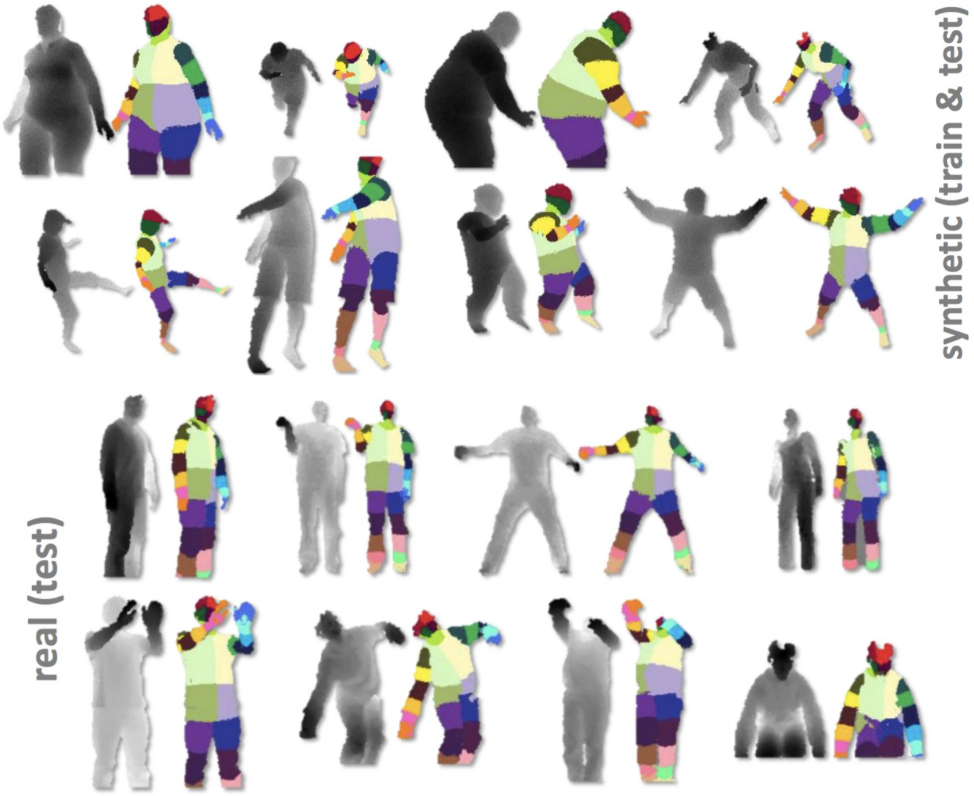
Decision Trees (Random Forest) are in XBox

Classifying using depth images

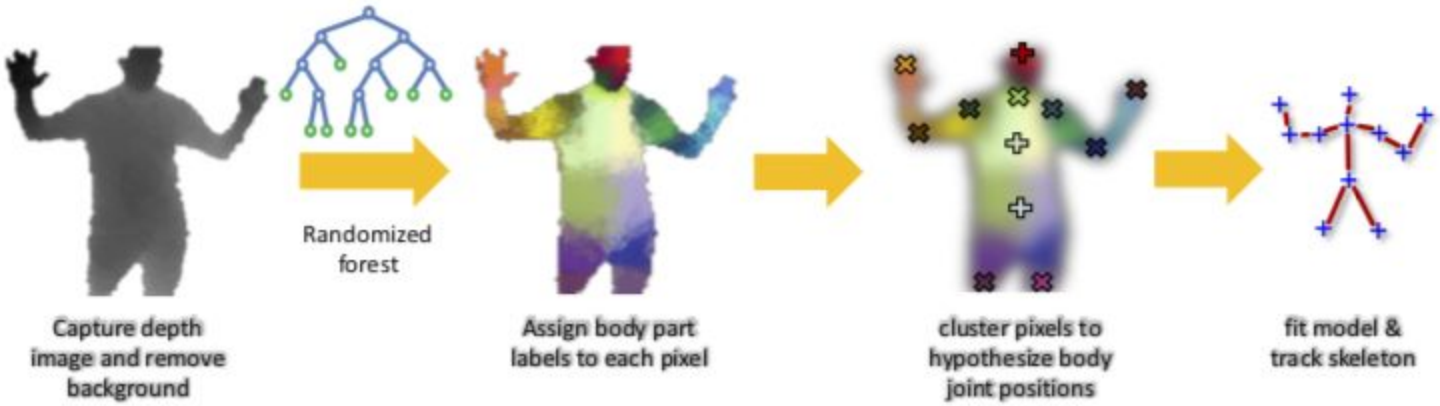


*slides adopted from University of Toronto

Trained on millions of example



Tree Voting


Randomized forest
Capture depth image and remove background
Assign body part labels to each pixel
cluster pixels to hypothesize body joint positions
fit model & track skeleton

Today: Learning Objectives

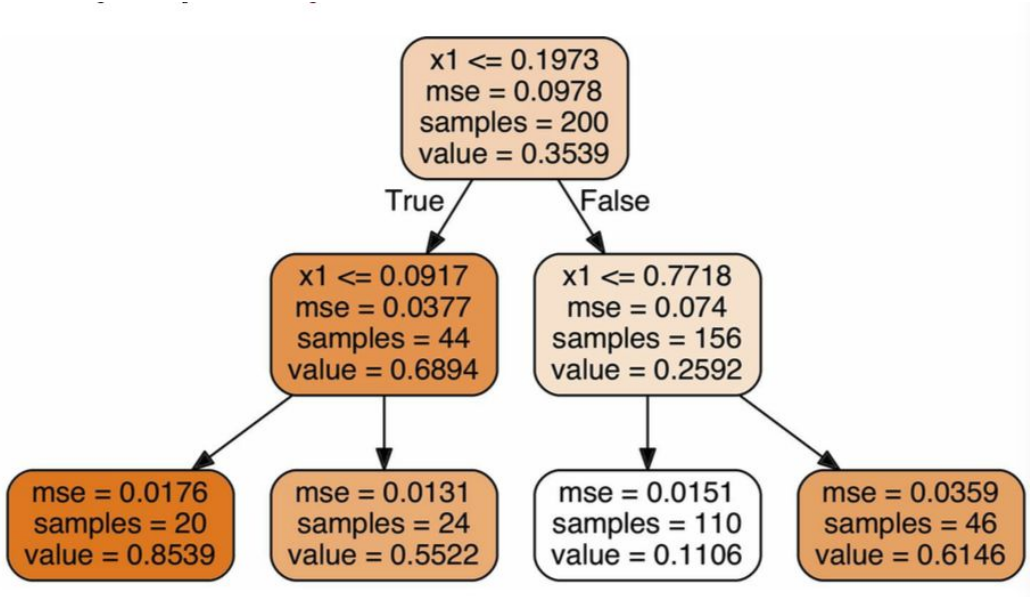
- ✓ Talk about Decision Trees
- ✓ Discover entropy and information gain
- ✓ Discuss decision tree limitations
- ✓ Learn about an application in XBOX

Next: Ensemble Learning and Random Forest

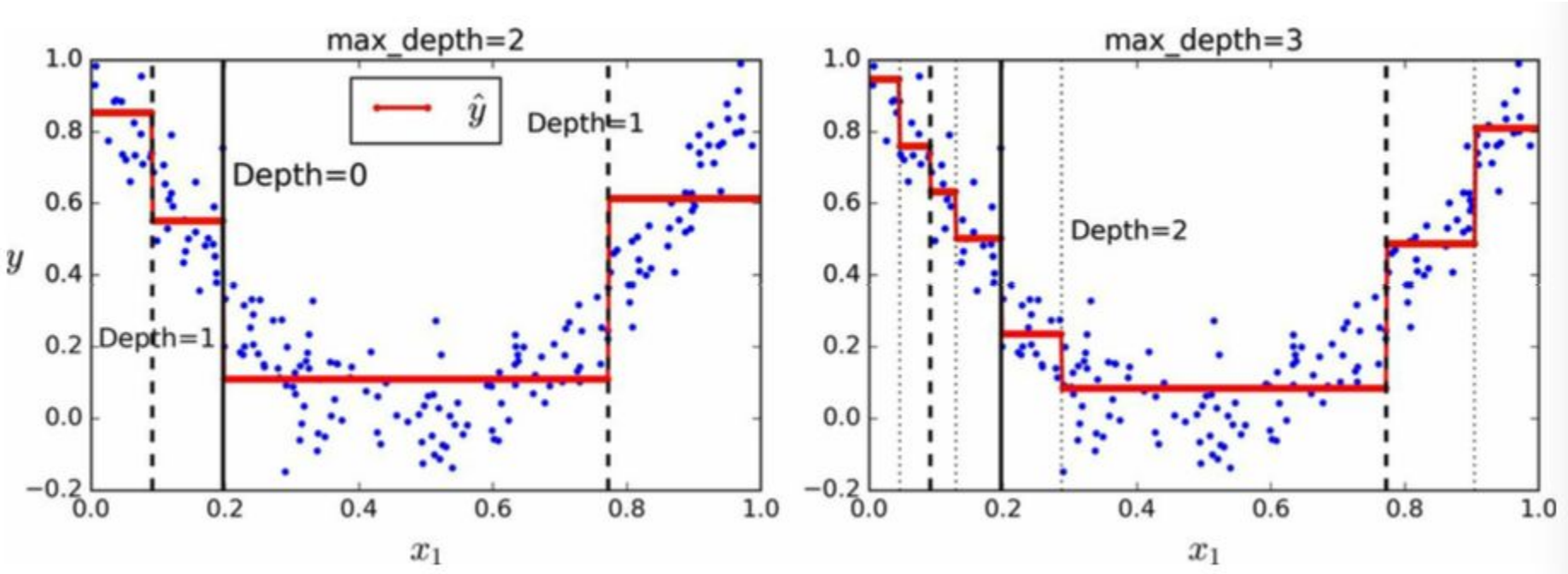
Bonus Slides

Regression

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X, y)
```



Max Depth and Prediction



CART cost function

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}}$$

$$\text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Regression Overfitting and Regularization