Decision Trees

Roi Yehoshua

Agenda

- Decision trees
- Impurity measures
- Entropy
- Tree pruning
- Regression trees

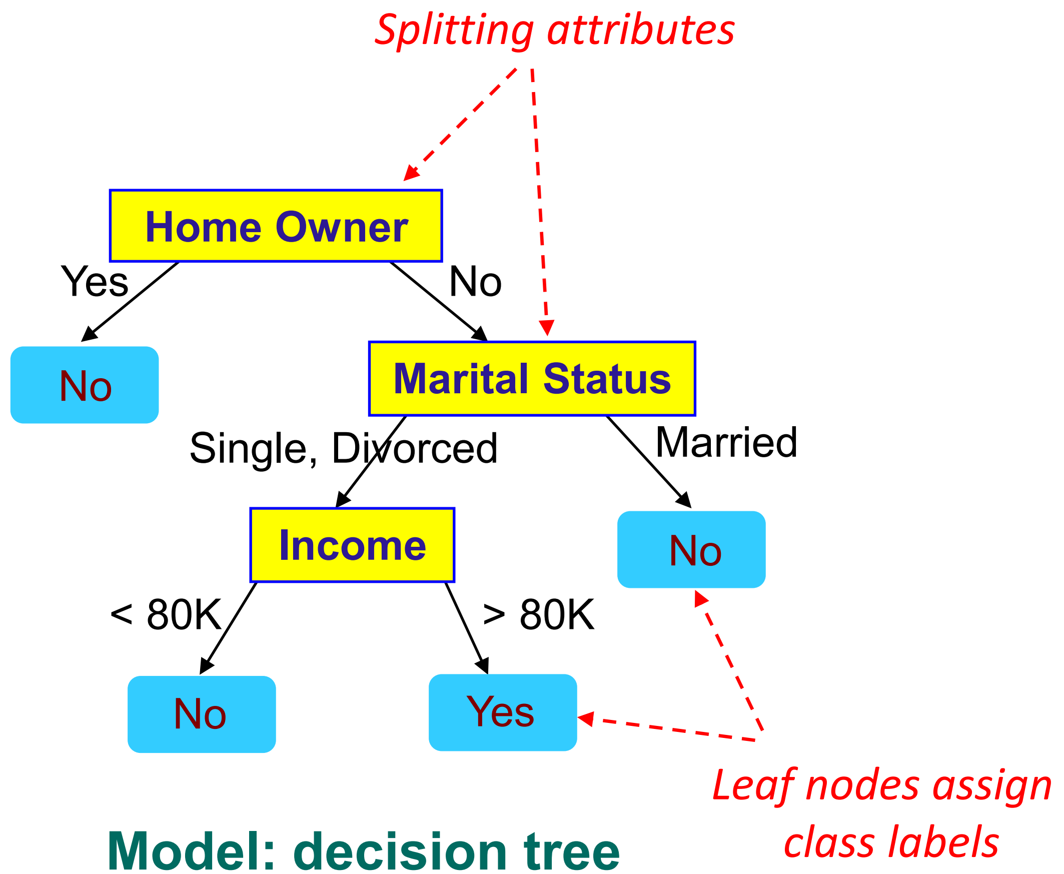
Decision Trees

- Decision trees are one of the most powerful machine learning models
- Based on simple decision rules inferred from the data set
- Can be used both for classification and regression
- Capable of fitting complex datasets
- Can be visualized and easily interpreted
- Have been successfully applied to a broad range of tasks
  - from medical diagnosis to credit loan approvals

Decision Tree Example

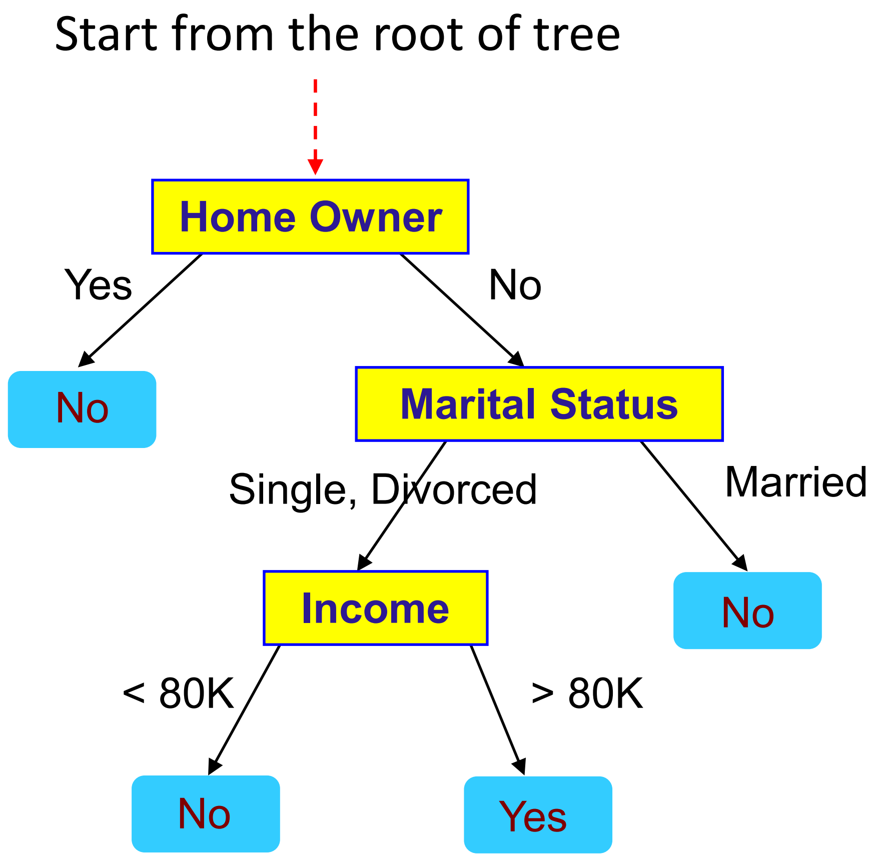
Tid	**Home Owner**	**Marital Status**	**Annual Income**	**Default**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

Training data



Model: decision tree

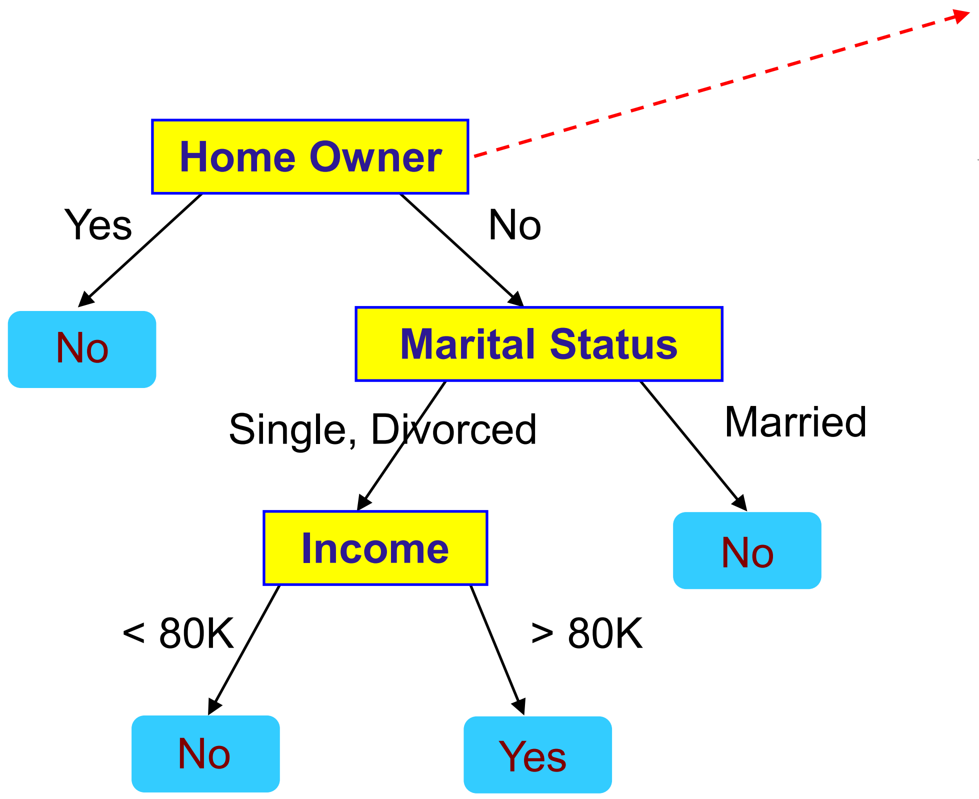
Apply Model to Test Data



Test data

Home Owner	Marital Status	Annual Income	Default
No	Married	80K	?

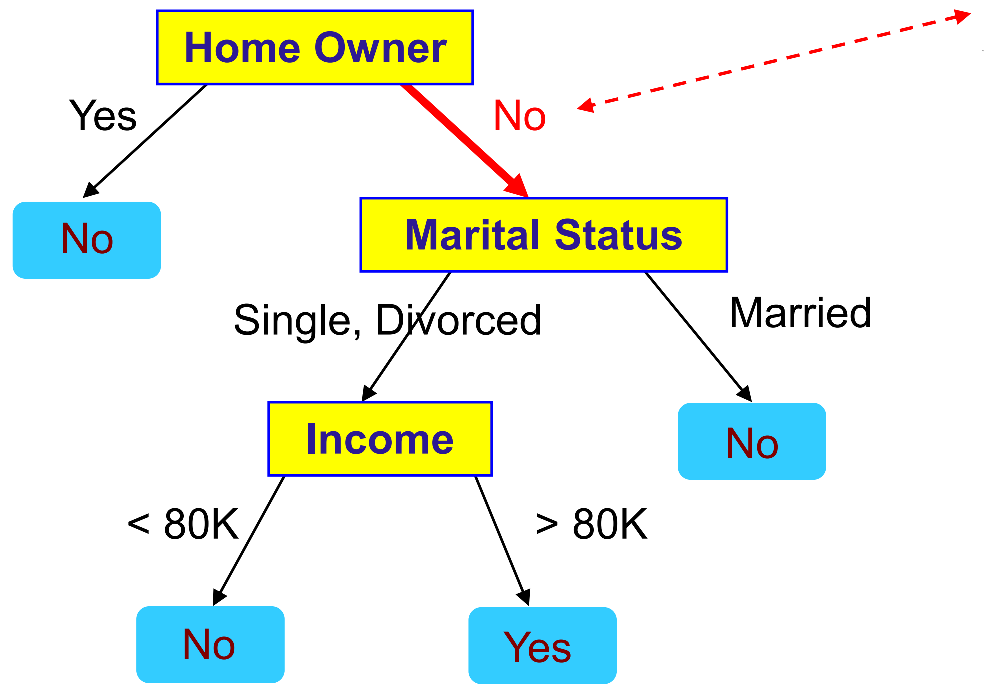
Apply Model to Test Data



Test data

Home Owner	Marital Status	Annual Income	Default
No	Married	80K	?

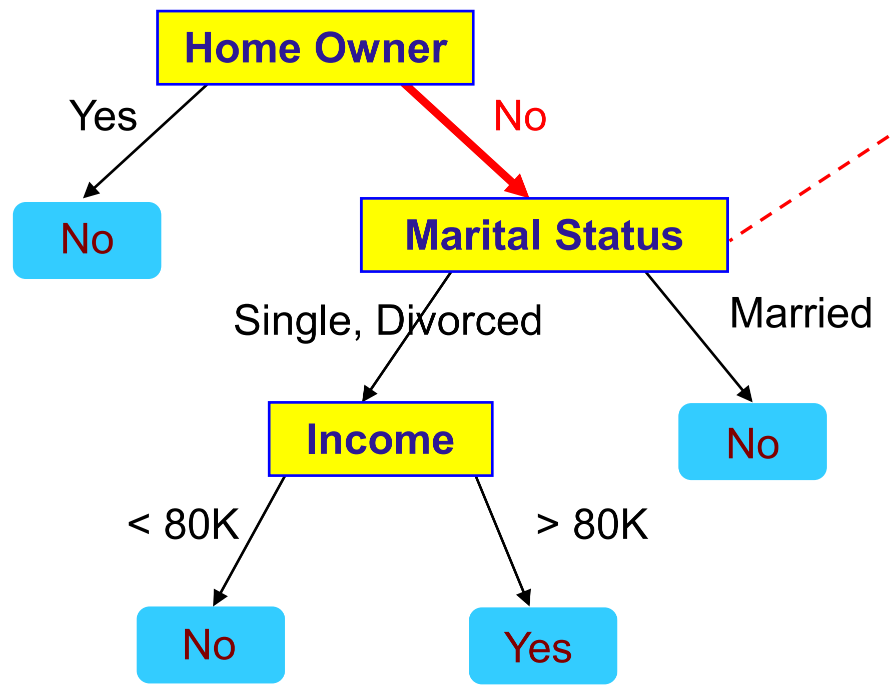
Apply Model to Test Data



Test data

Home Owner	Marital Status	Annual Income	Default
No	Married	80K	?

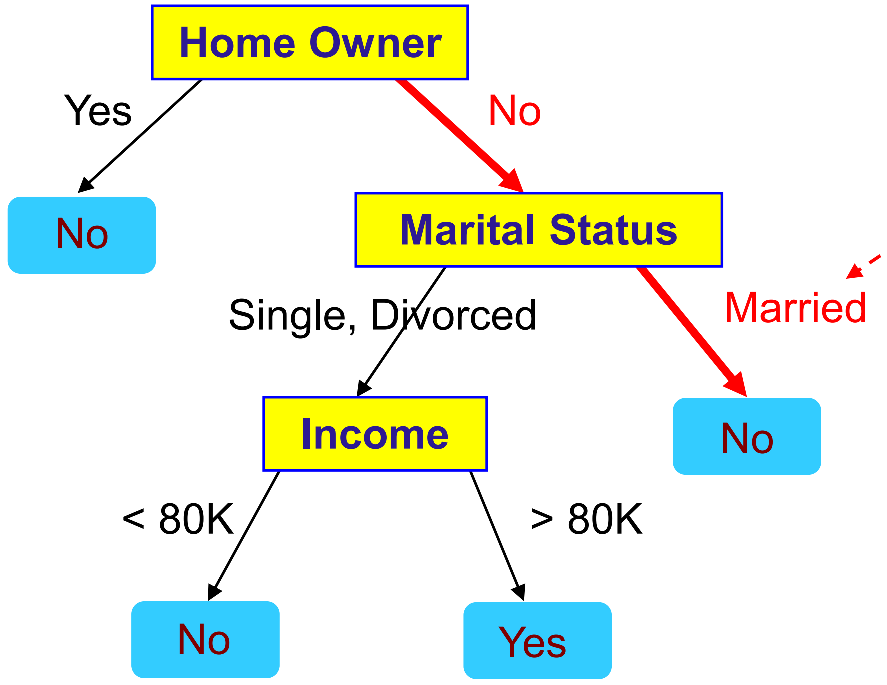
Apply Model to Test Data



Test data

Home Owner	Marital Status	Annual Income	Default
No	Married	80K	**?**

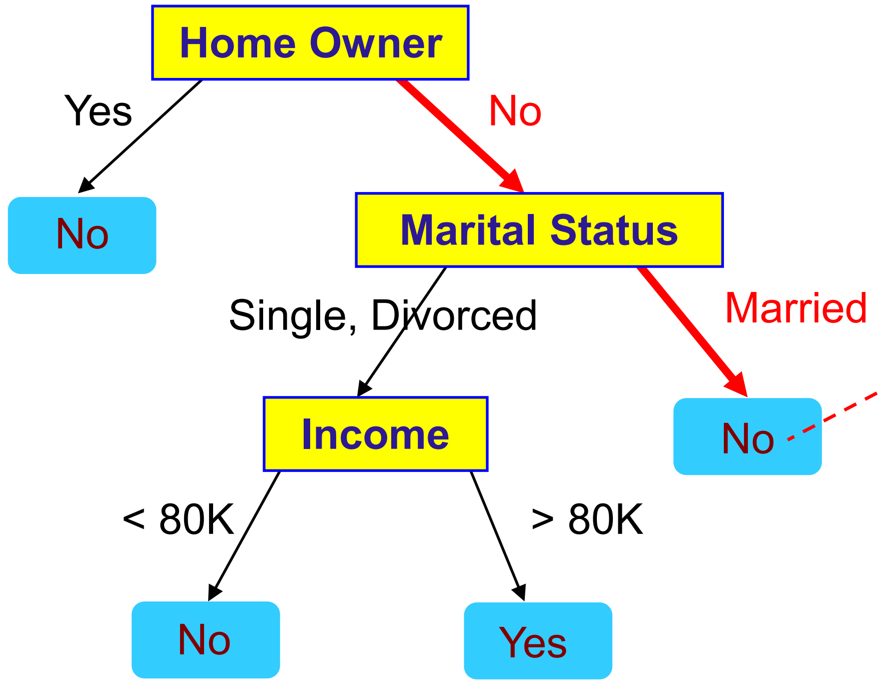
Apply Model to Test Data



Test data

Home Owner	Marital Status	Annual Income	Default
No	Married	80K	?

Apply Model to Test Data



Test data

Home Owner	Marital Status	Annual Income	Default
No	Married	80K	?

Assign Default to "No"

Decision Tree Induction

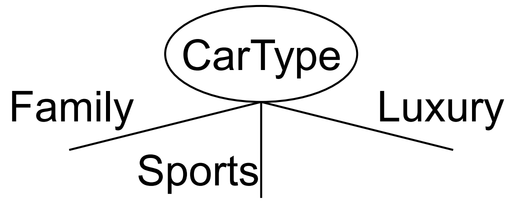
- A decision tree is grown in a top-down, recursive fashion
- Initially, the root node of the tree contains all the samples in the training set
- In every node we split the training set into based on an **attribute value test**
  - e.g., $X > 10$, where X is one of the attributes
- This process is repeated recursively on each child node
- The recursion ends when either:
  - All the samples in the node belong to the same class
  - When there are no more attributes that can be used for splitting

Decision Tree Induction

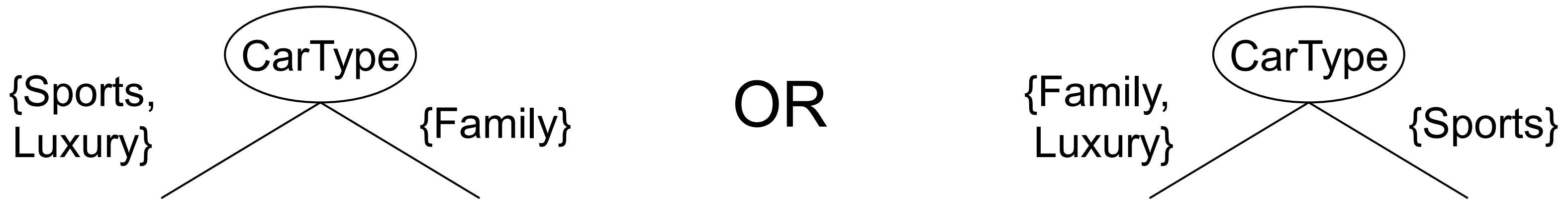
```
Algorithm Decision-Tree-Learning(examples, attributes)
Input:
    examples are the training examples $\{(x_1, y_1), ..., (x_n, y_n)\}$
    attributes is the set of attributes (features) in the training set

 1: Create a root node for the tree
 2: if all examples have the same class label $c$ then
 3:     return root with label = $c$
 4: else if attributes is empty then
 5:     return root with the most common label in examples
 6: else
 7:     Select attribute $A$ from attributes that best classifies examples based on an
    impurity measure
 8:     Set $A$ the attribute for root
 9:     for each value $v_i$ of $A$ do
10:         Add a branch to the tree with label $A = v_i$
11:         Let exp be the subset of examples that have $A = v_i$
12:         if exp is empty then
13:             Add a leaf node to the branch with the most common label in examples
14:         else
15:             subtree $\leftarrow$ Decision-Tree-Learning(exp, attributes $- A$)
16:             Add subtree to the branch
17: return root
```

Split Types

- Multi-way split: use as many partitions as distinct values



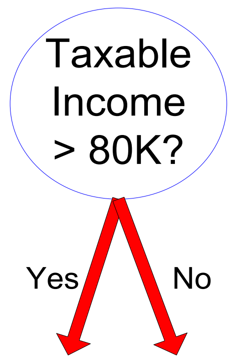
- Binary split: divides values into two subsets
  - Need to find optimal partitioning



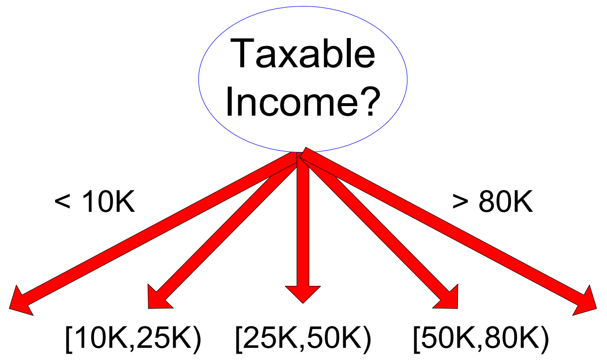
- Some decision tree learning algorithms (e.g., CART) use only binary splits

Continuous Attributes

- Multi-way split: $v_i \leq A < v_{i+1}$ $(i = 1, \dots, k)$
  - v_i are distinct values of feature A in the data set
  - Adjacent intervals can be aggregated into wider ranges as long as the order is preserved
- Binary split: $(A < v)$ or $(A \geq v)$
  - Consider all possible splits and find the best cut
  - Can be more computationally intensive



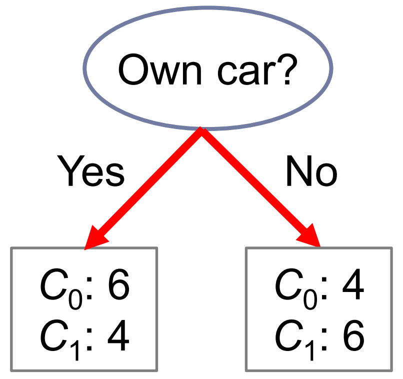
(i) Binary split

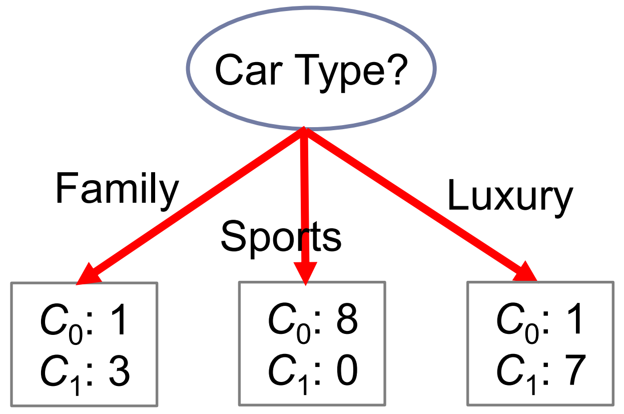


(ii) Multi-way split

How to Determine the Best Split

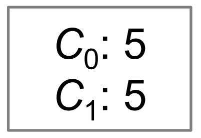
- Which test condition is better?

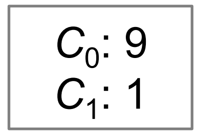




- Idea: a good attribute splits the examples into purer subsets
  - Ideally the subsets are “all positive” or “all negative”

Non-homogeneous,
High degree of impurity





Homogeneous,
Low degree of impurity

- Need a measure of node impurity

Measures of Node Impurity

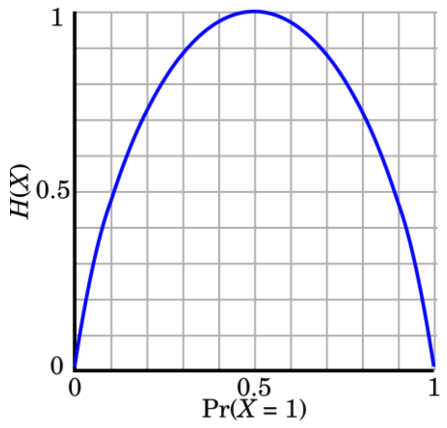
- Entropy
- Gini index

Entropy

- Entropy is a measure of the uncertainty of a random variable
  - The more clueless I am about the value of the variable, the higher its entropy
- The entropy of a random variable X with values $x_1, x_2, \ldots, x_n$ is defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

  - $p\log_2 p$ is considered to be 0 whenever $p = 0$
- For example, the entropy of a Bernoulli variable as a function of its probability p:



Entropy

- Distributions $p(x_i)$ that are sharply peaked around a few values will have lower entropy than those that are spread more evenly across many values

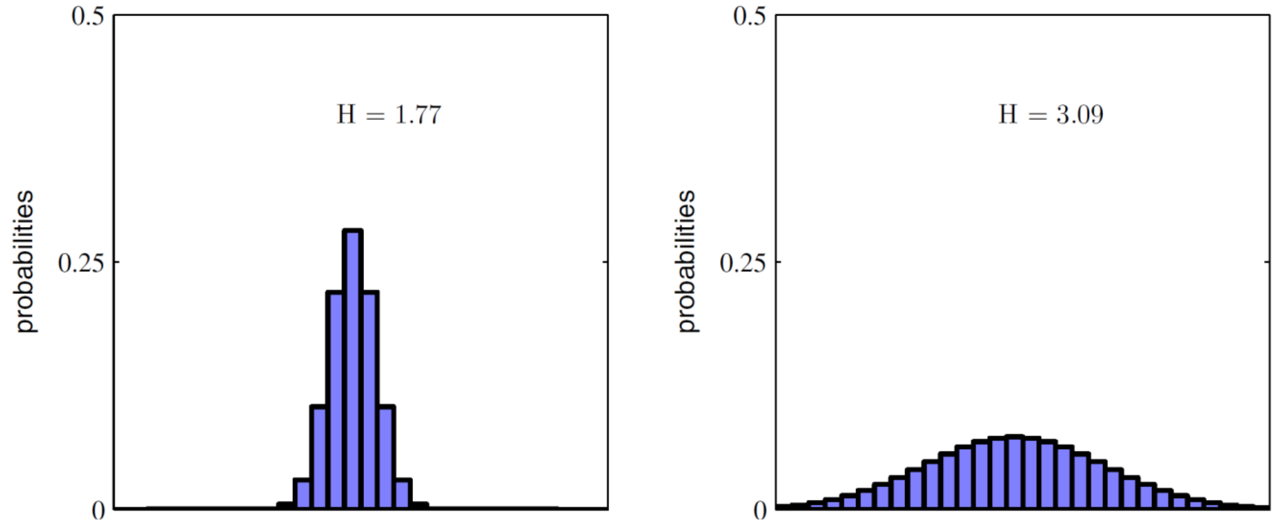


Figure 1.30 Histograms of two probability distributions over 30 bins illustrating the higher value of the entropy H for the broader distribution. The largest entropy would arise from a uniform distribution that would give $H = -\ln(1/30) = 3.40$.

Entropy in Decision Trees

- Entropy at a given node *v* of the decision tree:

$$\boxed{H(v) = -\sum_{k=1}^{K} p(\mathcal{C}_k|v)\log_2 p(\mathcal{C}_k|v)}$$

  - $p(C_k|v)$ is the relative frequency of class *k* at node *v*
- Measures homogeneity of a node
  - Maximum ($\log_2|K|$) when samples are equally distributed among all classes, implying least information
  - Minimum (0) when all samples belong to one class, implying most information

C_0	**0**
C_1	**6**

$H = -0\log_2 0 - 1\log_2 1$
$= 0$

C_0	**2**
C_1	**4**

$H = -(2/6)\log_2(2/6) - (4/6)\log_2(4/6)$
$= 0.918$

C_0	**3**
C_1	**3**

$H = -(3/6)\log_2(3/6) - (3/6)\log_2(3/6)$
$= 1$

Information Gain

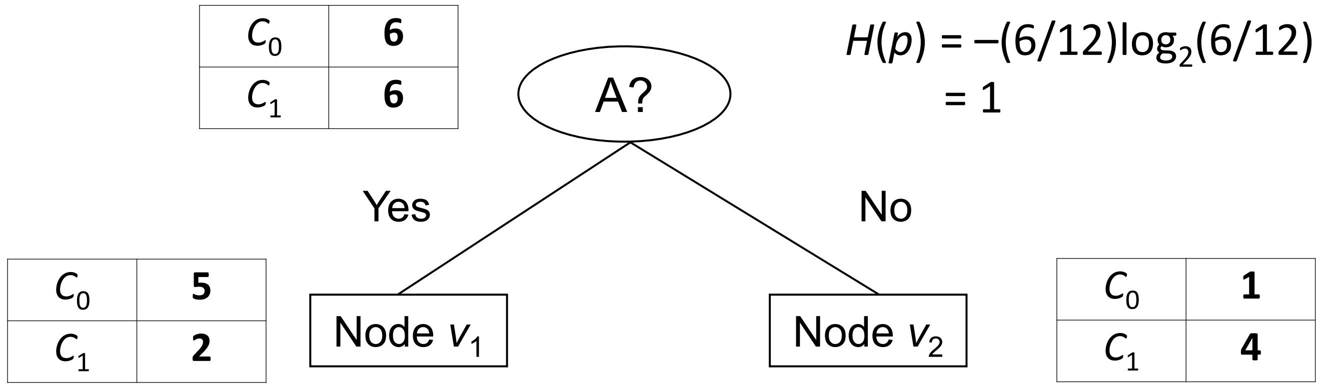
- Measures the reduction in entropy as a result of splitting a node p into child nodes $v_1, \dots, v_k$

$$\boxed{\Delta_{\text{info}} = H(p) - \sum_{i=1}^{k} \frac{n_i}{n} H(v_i)}$$

  - n = number of samples at node p
  - n_i = number of samples at child node v_i

- Our goal is to choose the split that maximizes the information gain
  - achieves most reduction in entropy

Information Gain Example



C_0	**6**
C_1	**6**

C_0	**5**
C_1	**2**

C_0	**1**
C_1	**4**

$$H(p) = -(6/12)\log_2(6/12) - (6/12)\log_2(6/12) = 1$$

$$H(v_1) = -(5/7)\log_2(5/7) - (2/7)\log_2(2/7) = 0.863$$

$$H(v_2) = -(1/5)\log_2(1/5) - (4/5)\log_2(4/5) = 0.722$$

$$\text{IG}(A) = 1 - (7/12) * 0.863 - (5/12) * 0.722 = 0.196$$

Gain Ratio

- Information gain tends to favor attributes that have a large number of distinct values
  - e.g., splitting customer records by customer ID would lead to 0 entropy
- Gain ratio adjusts information gain by the entropy of the partitioning
- For a node *p* that is split into child nodes $v_1, \ldots, v_k$ with n_i samples each, we define

$$\text{SplitInfo} = -\sum_{i=1}^{k} \frac{n_i}{n} \log_2 \frac{n_i}{n}$$

- Then gain ratio is defined as the ratio between information gain and the split info:

$$\text{GainRatio} = \frac{\Delta_{\text{info}}}{\text{SplitInfo}}$$

- Higher entropy partitioning (large number of small partitions) is penalized

Gini Index

- Gini index (aka gini impurity) at a node *v*:

$$\boxed{\text{Gini}(v) = 1 - \sum_{k=1}^{K} [p(\mathcal{C}_k|v)]^2}$$

  - $p(C_k|v)$ is the relative frequency of class *k* at node *v*
  - Maximum ($1 - 1/|K|$) when samples are equally distributed among all classes, implying least interesting information
  - Minimum (0) when all samples belong to one class, implying most interesting information

C_0	**0**
C_1	**6**

Gini = $1 - 0^2 - (6/6)^2$
= 0

C_0	**2**
C_1	**4**

Gini = $1 - (2/6)^2 - (4/6)^2$
= 0.444

C_0	**3**
C_1	**3**

Gini = $1 - (3/6)^2 - (3/6)^2$
= 0.5

Splitting Based on Gini

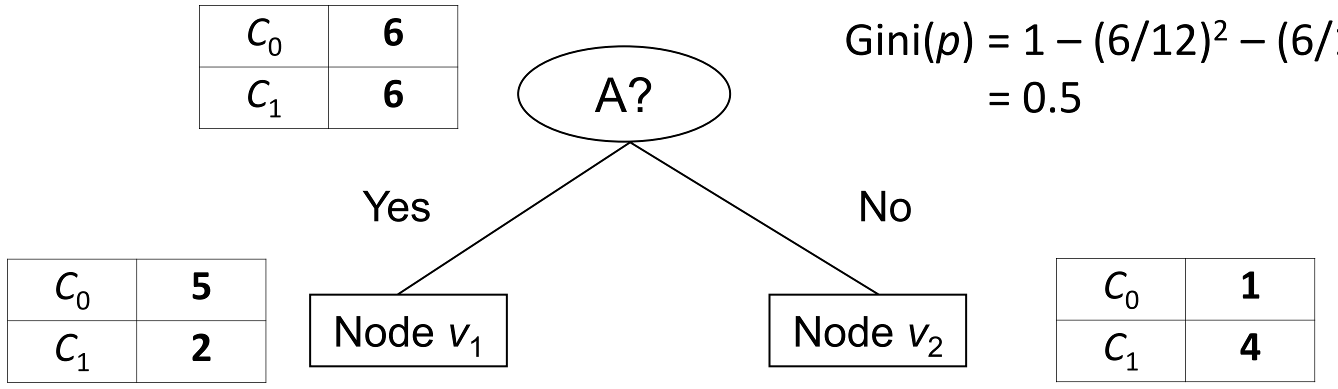
- The reduction of gini impurity as a result of splitting node p into child nodes $v_1, \ldots, v_k$:

$$\boxed{\Delta_{\text{Gini}} = \text{Gini}(p) - \sum_{i=1}^{k} \frac{n_i}{n} \text{Gini}(v_i)}$$

  - n = number of samples at node p
  - n_i = number of samples at child node v_i

- Our goal is to choose the split that achieves the most reduction in impurity

Splitting Based on Gini Example



C_0	**6**
C_1	**6**

C_0	**5**
C_1	**2**

C_0	**1**
C_1	**4**

$$\begin{aligned}\text{Gini}(p) &= 1 - (6/12)^2 - (6/12)^2 \\ &= 0.5\end{aligned}$$

$$\begin{aligned}\text{Gini}(v_1) &= 1 - (5/7)^2 - (2/7)^2 \\ &= 0.408\end{aligned}$$

$$\begin{aligned}\text{Gini}(v_2) &= 1 - (1/5)^2 - (4/5)^2 \\ &= 0.320\end{aligned}$$

$$\begin{aligned}\Delta_{\text{Gini}}(\text{A}) &= 0.5 - (7/12) * 0.408 - (5/12) * 0.320 \\ &= 0.129\end{aligned}$$

Continuous Attributes: Computing Gini Index

- For efficient computation:
  - Sort the samples based on the attribute values
  - Linearly scan these values, each time updating the class counts and computing Gini index
    - Further optimization: consider only split positions between samples with different class labels
  - Choose the split position that has the least Gini index

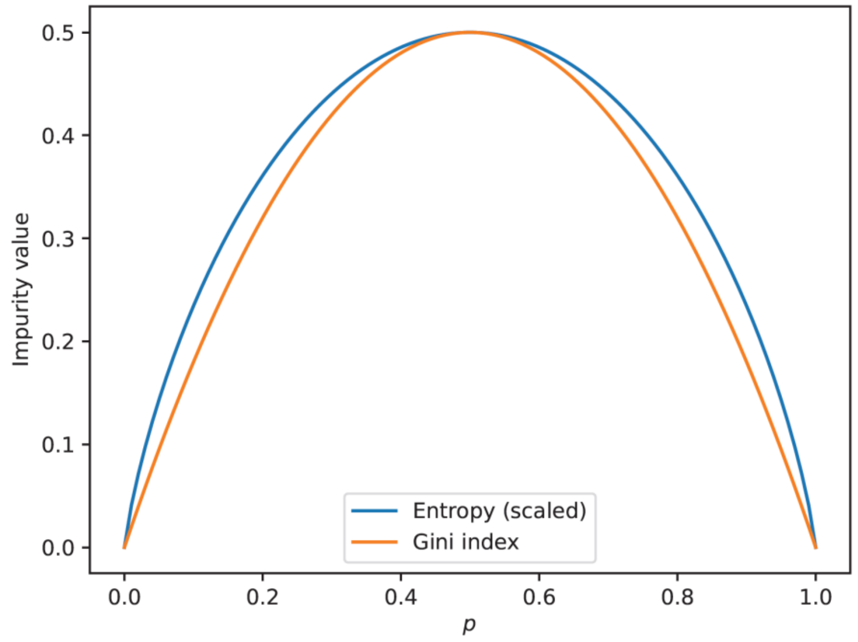
	Class	No	No	No	Yes	Yes	Yes	No	No	No	No
		Annual Income									
Sorted values →		60	70	75	85	90	95	100	120	125	220

Split positions →	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400		0.420	

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Comparing the Impurity Measures

- For a binary classification problem:



- The choice of impurity measure has little effect on the performance
- Gini index is usually preferred as it is less computationally expensive
  - No logarithmic computations involved

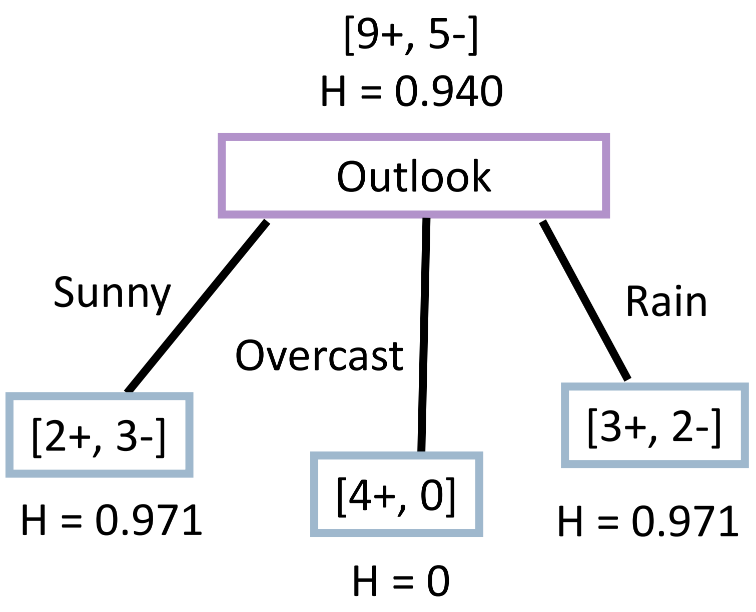
Decision Tree Construction Example

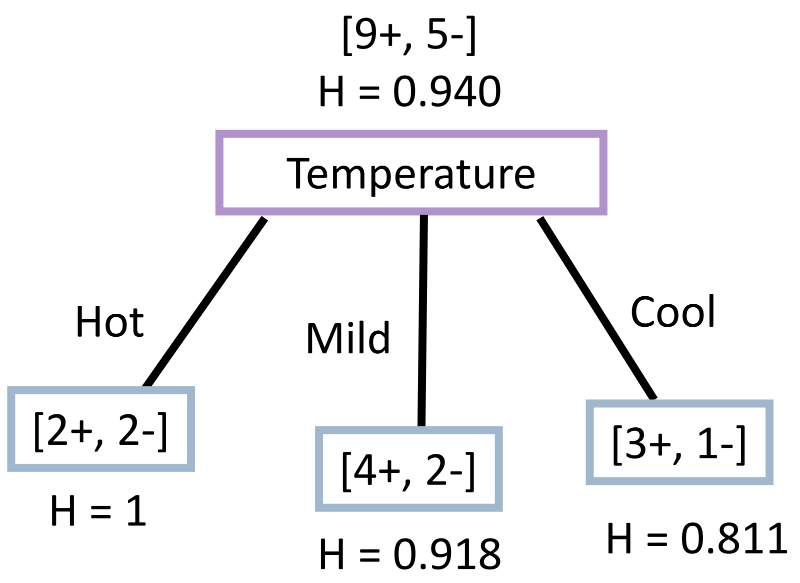
- Build a decision tree for the following classification problem using information gain

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree Construction Example

- Selecting the first splitting attribute:



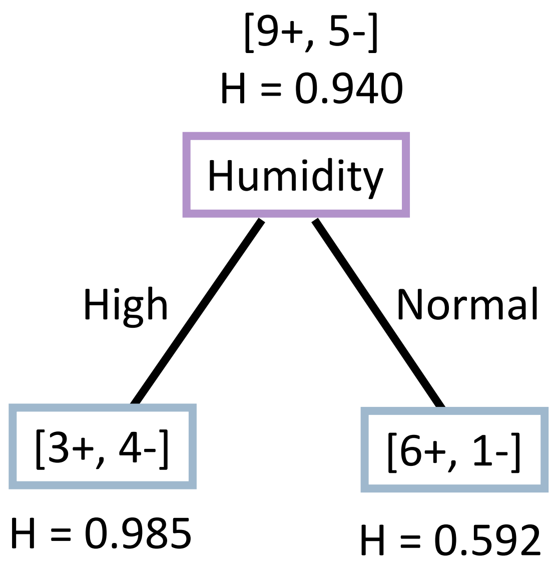


Gain(Outlook) =
= 0.940 - (5/14)*0.971- (4/14)*0 - (5/14)*0.0971
= 0.247

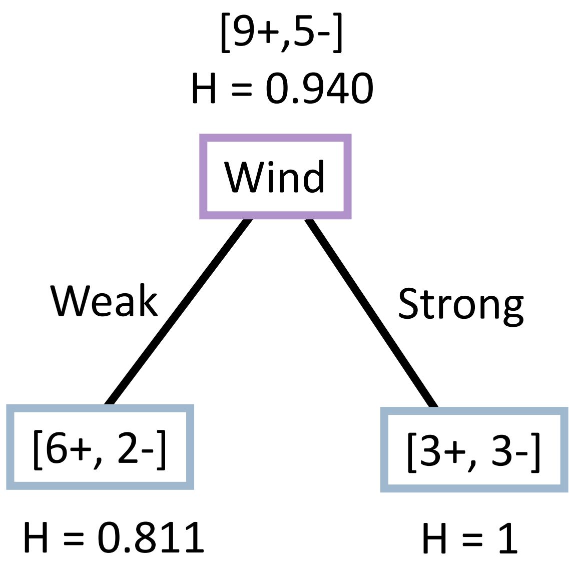
Gain(Temperature) =
= 0.940 - (4/14)*1 - (6/14)*0.918 - (4/14)*0.811
= 0.029

Decision Tree Construction Example

- Selecting the first splitting attribute:



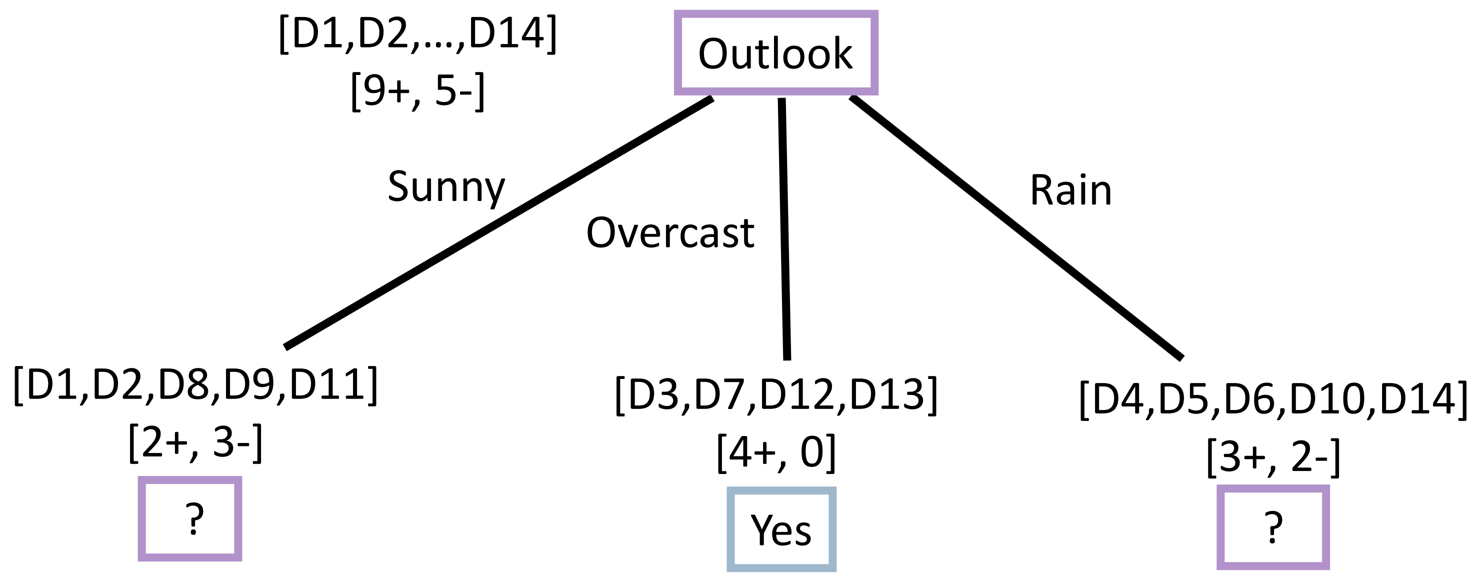
Gain(Humidity) =
= 0.940 – (7/14)*0.985 – (7/14)*0.592
= 0.151



Gain(Wind) =
= 0.940 – (8/14)*0.811 – (6/14)*1
= 0.048

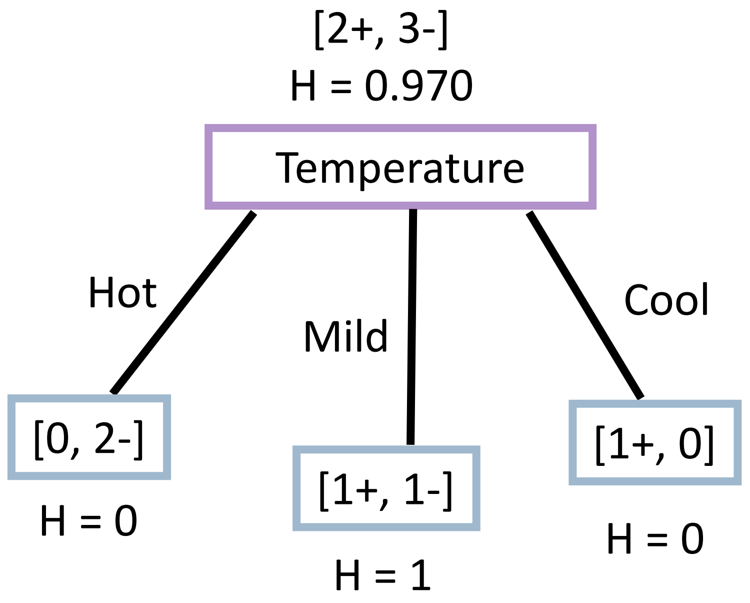
Decision Tree Construction Example

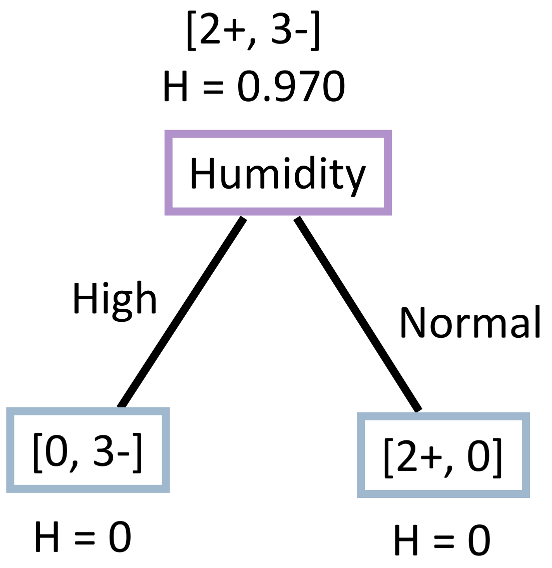
- The attribute that provides the highest information gain is Outlook
- Thus, the first level of the decision tree will look like this:

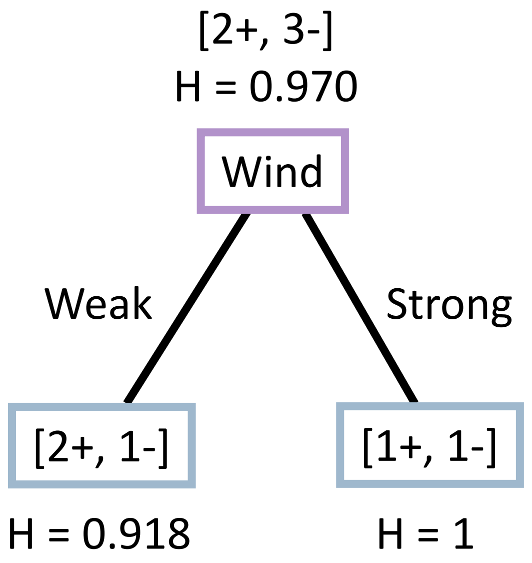


Decision Tree Construction Example

- Choosing the splitting criterion for the Sunny node:







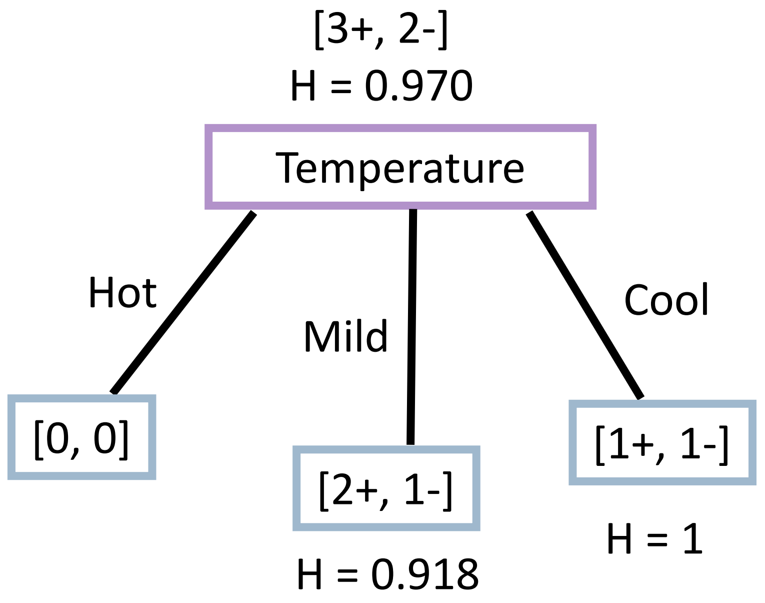
Gain(Temperature) =
= 0.970 – (2/5)*0 – (2/5)*1 – (1/5)*0
= 0.570

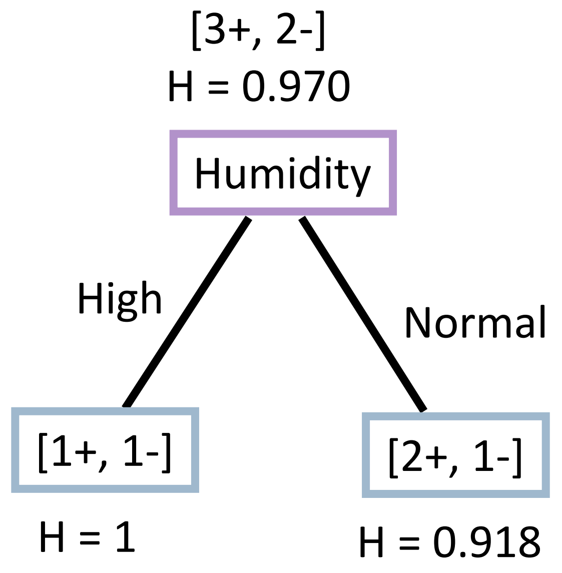
Gain(Humidity) =
= 0.970 – (3/5)*0 – (2/5)*0 =
0.970

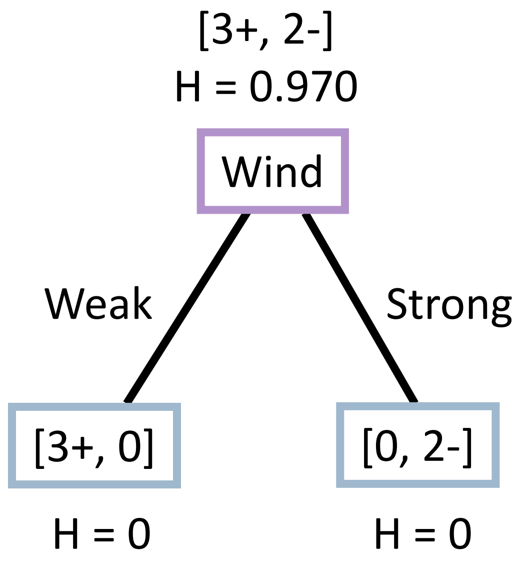
Gain(Wind) =
= 0.970 – (3/5)*0.918 – (2/5)*1.0
= 0.019

Decision Tree Construction Example

- Choosing the splitting criterion for the Rain node:







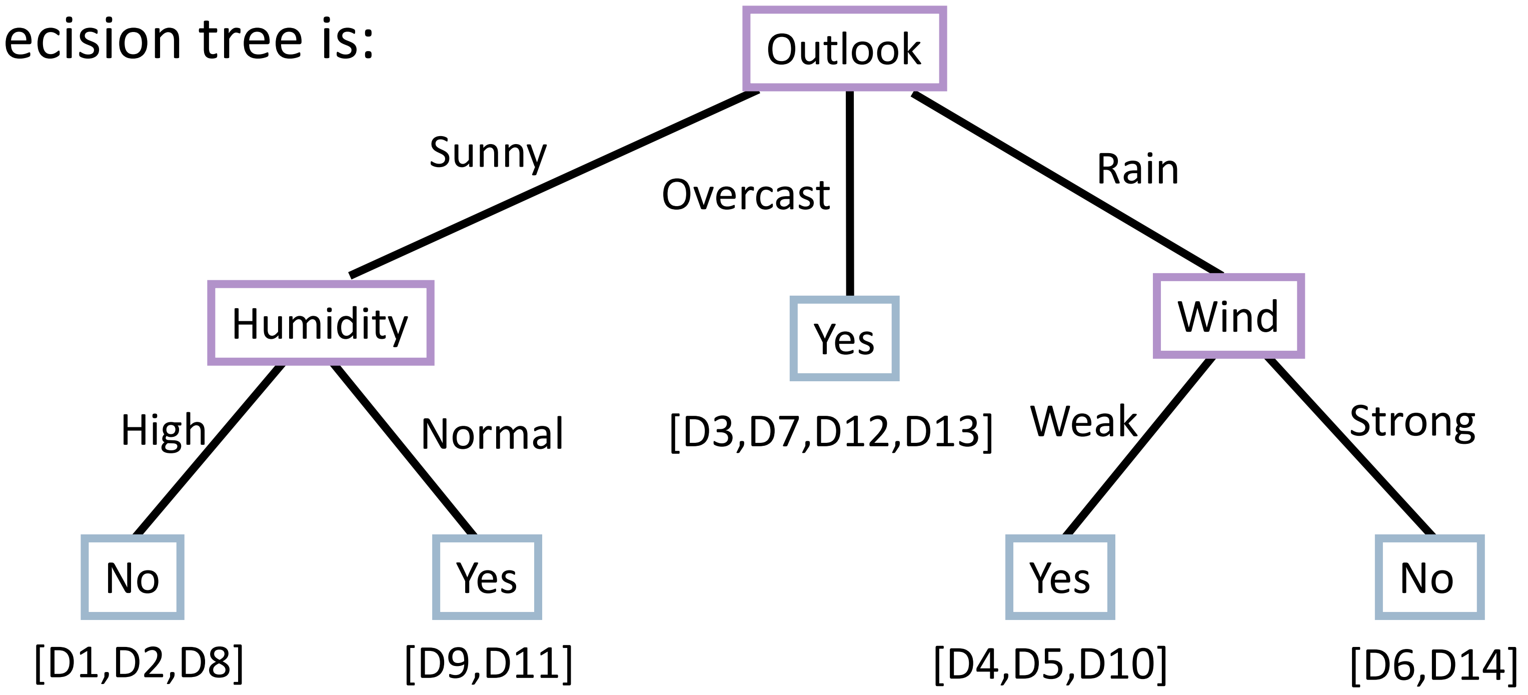
Gain(Temperature) =
= 0.970 – (3/5)*0.918 – (2/5)*1
= 0.0192

Gain(Humidity) =
= 0.970 – (2/5)*1 – (3/5)*0.918
= 0.0192

Gain(Wind) =
= 0.970 – (2/5)*0 – (3/5)*0
= 0.970

Decision Tree Construction Example

- The final decision tree is:



Example: Diabetes Diagnosis

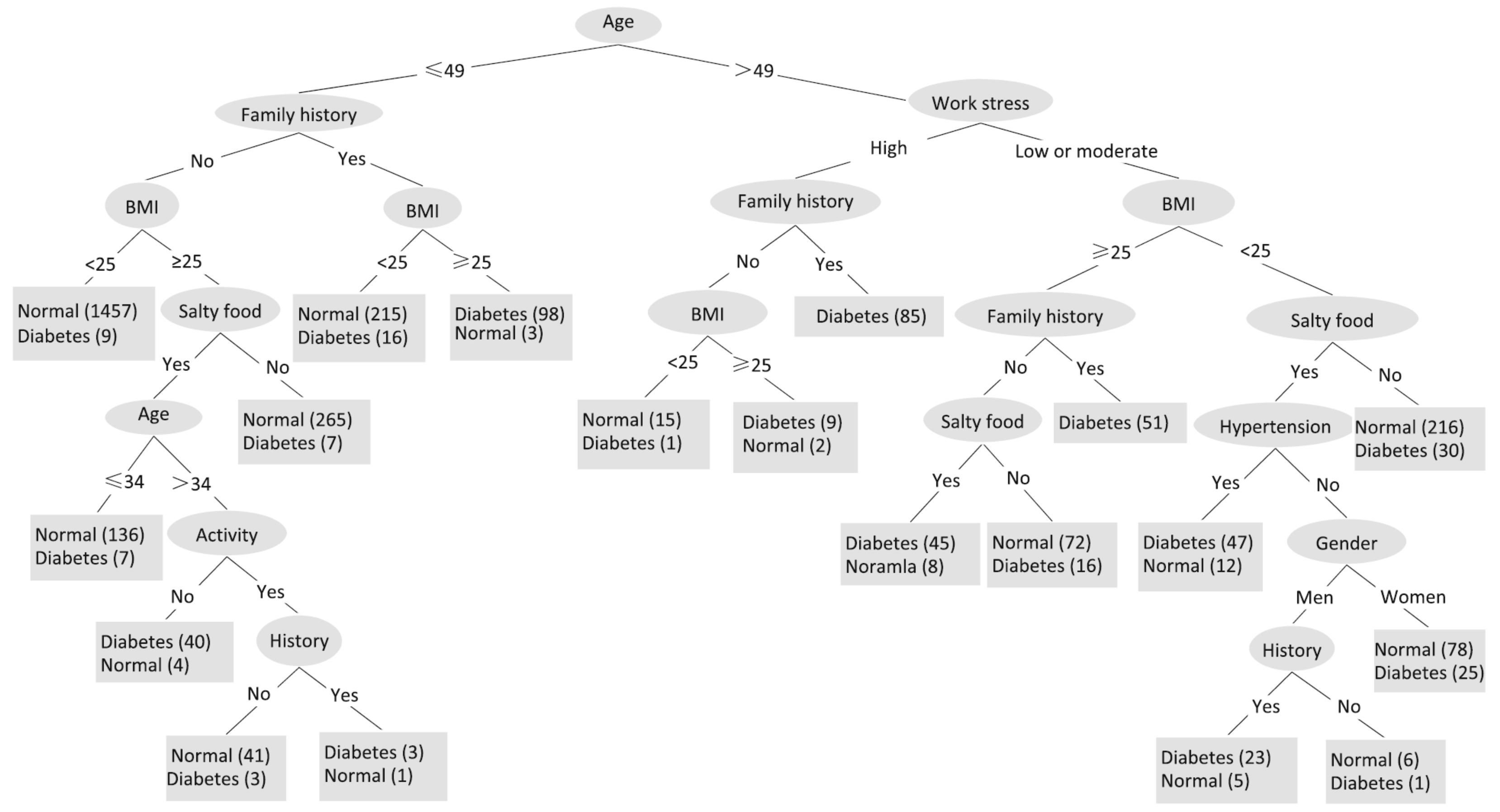


Fig. 3 Decision tree of diabetes classifiers. The sample size is given as the number in parentheses at each node

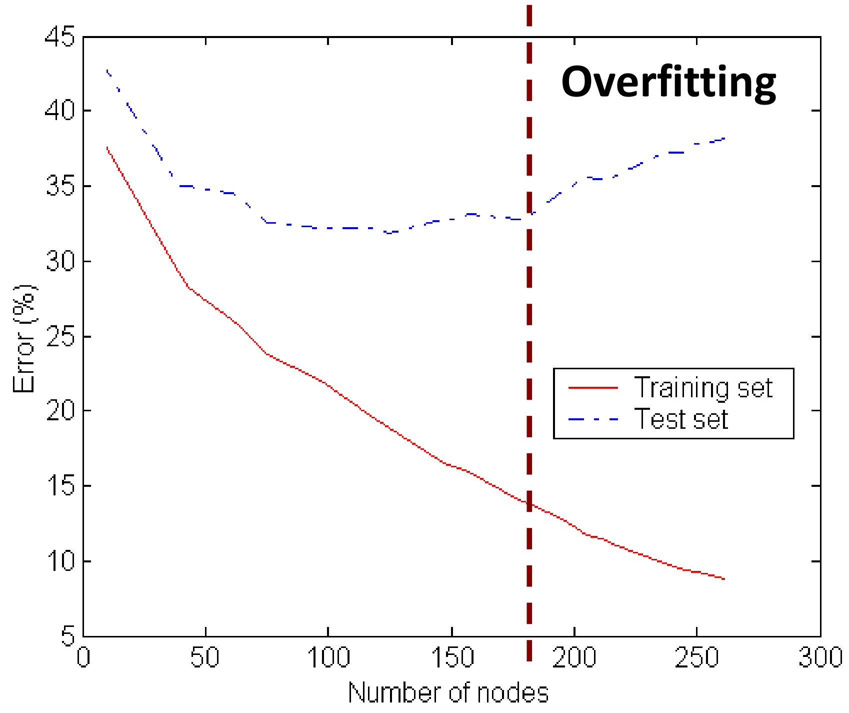
Dongmei Pei et al., “Accurate and rapid screening model for potential diabetes mellitus”, 2019

Time Complexity

- Let *n* be the number of samples and *d* the number of features
- Constructing a tree
  - The cost at each node consists of searching through $O(d)$ features
    - For each continuous feature we need to sort the samples according to the feature values
    - Thus, in the worst case the cost at each node is $O(dn\log n)$
  - The tree contains at most $O(n)$ nodes
    - Since each node contains at least one sample less than its parent node
  - Therefore, the total cost for construing a tree is $O(dn^2\log n)$
- Making a prediction
  - Worst case complexity is $O(m)$, where m is the maximum depth of the tree
  - When the decision tree is a binary balanced tree, $m = O(logn)$
    - Although the tree construction algorithm tries to generate balanced trees, they won't always be

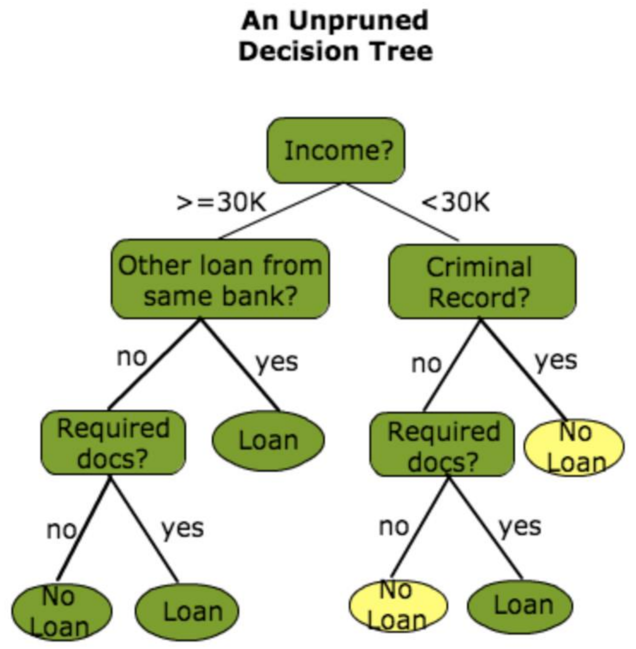
Overfitting

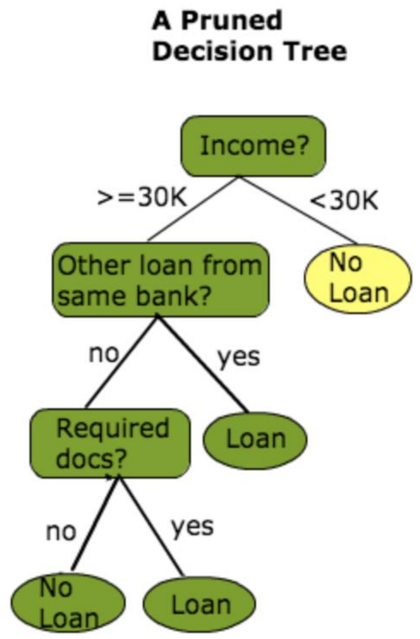
- One of the biggest issues with decision trees is overfitting
  - The leaf nodes of the tree can be expanded until it perfectly fits the training set
  - Although the training error for such a tree is 0, the test error can be large
  - Some of the nodes may accidentally fit the noise in the training data



Decision Tree Pruning

- We can use decision tree **pruning** to reduce overfitting
- Two main methods to avoid overfitting in decision tree learning:
  - Pre-pruning
  - Post-pruning





Pre-Pruning

- Stop the tree construction before it becomes a fully-grown tree
- Common stopping conditions:
  - **Maximum depth**: stop when the tree reaches a maximum depth
  - **Minimum samples to split**: split only if a node contains a minimum number of samples
  - **Minimum samples per leaf**: ensure that each leaf has a minimum number of samples
  - **Maximum number of leaf nodes**:
    - The tree is grown in a best-first manner (prioritizing splits with higher impurity reduction)
    - Stop when maximum number of leaf nodes is reached
  - **Minimum impurity decrease**: don’t split a node if it doesn’t reduce the impurity by a specified amount
- Multiple criteria can be combined

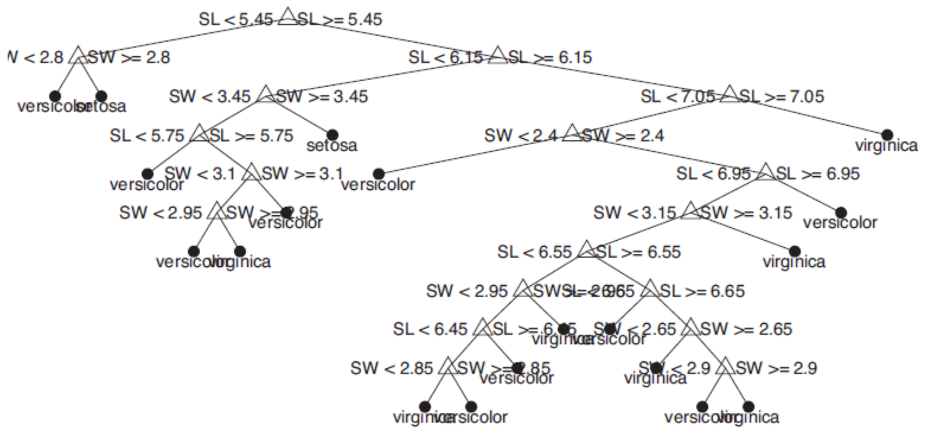
Post Pruning

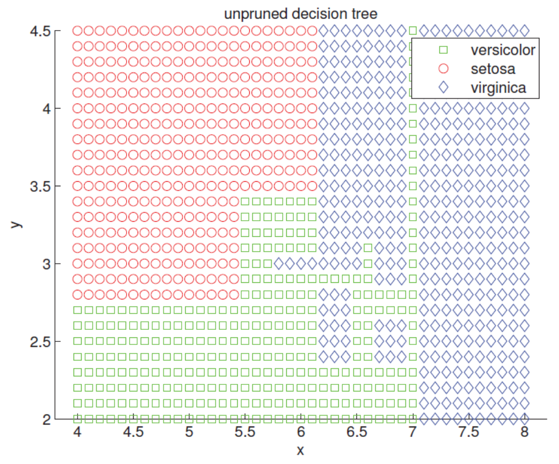
- A simple and fast approach for post-pruning
  - Grow decision tree to its entirety
  - Trim the nodes of the tree in a bottom-up fashion
  - Each node is replaced by a leaf node
    - whose class label is determined by the majority class of the samples in its sub-tree
  - If the generalization error improves after trimming, the change is kept

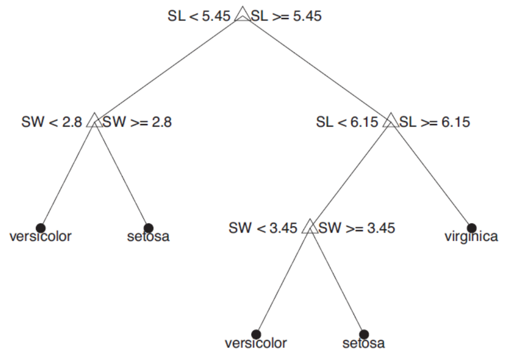
- Pros:
  - Tends to give better results than pre-pruning because the pruning decisions are made based on a fully grown tree
- Cons:
  - Requires more computations to grow the full tree

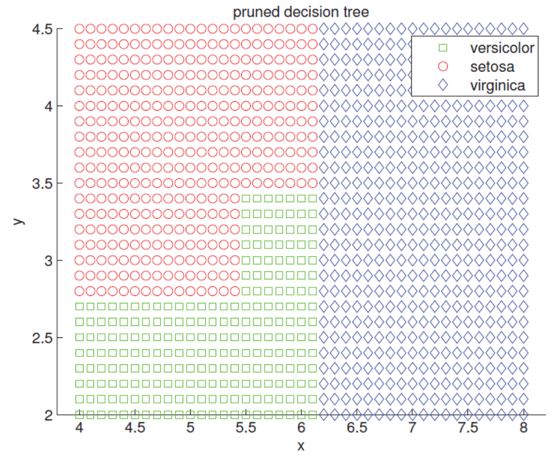
Post Pruning Example

- A decision tree for the Iris data using the first two features (sepal length and width)









Other Issues

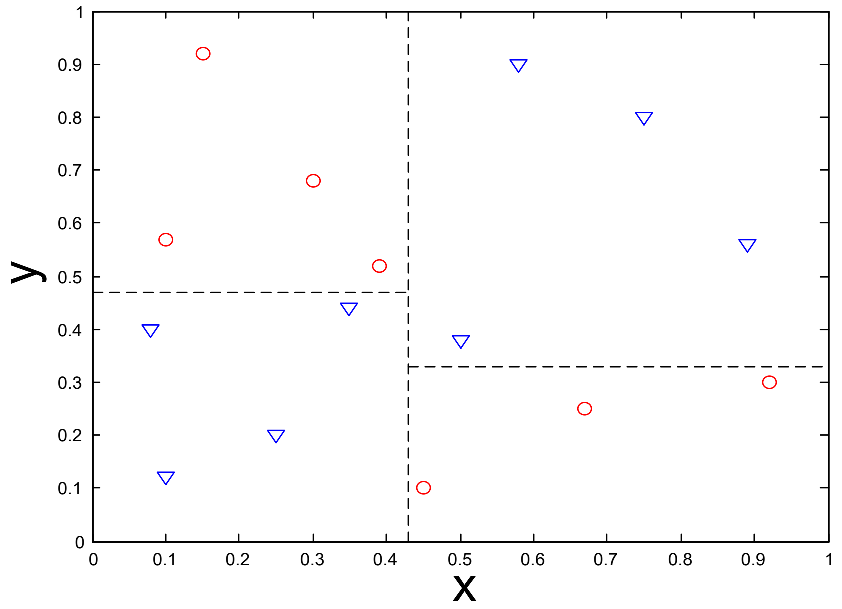
- Optimality
- Decision boundaries
- Expressiveness
- Instability

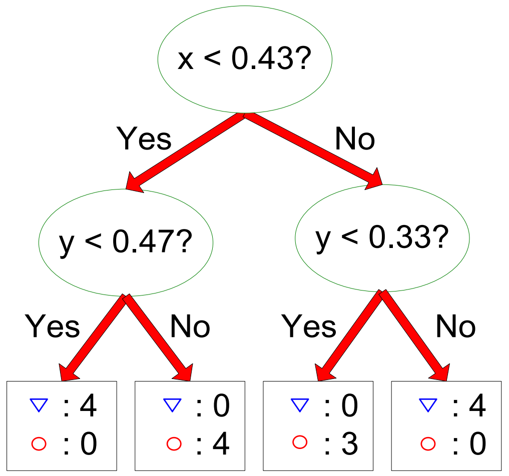
Tree Optimality

- Finding an optimal decision tree is an NP-complete problem
- An optimal decision tree is one that minimizes the number of tests required to classify correctly all the training samples
- The algorithm presented so far uses a greedy, recursive partitioning strategy to induce a reasonable solution
- Other approaches?
  - Bottom-up
  - Bi-directional

Decision Boundaries

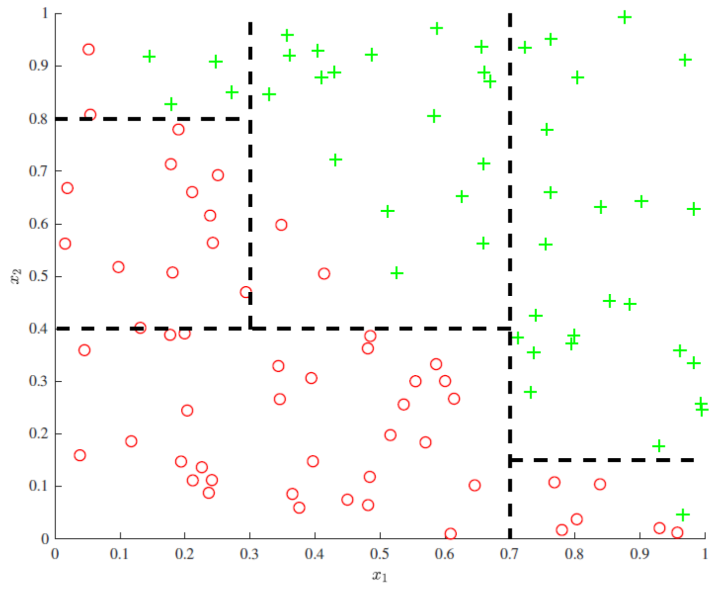
- The decision boundaries learned by a DT are always **rectilinear** (parallel to the axes)
  - Since the test conditions involve a single attribute at-a-time
- This limits the expressiveness of the decision tree representation

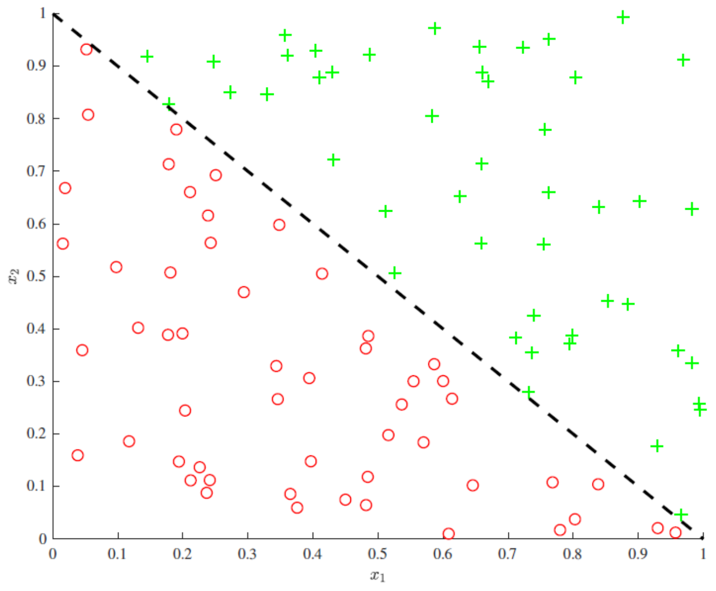




Decision Boundaries

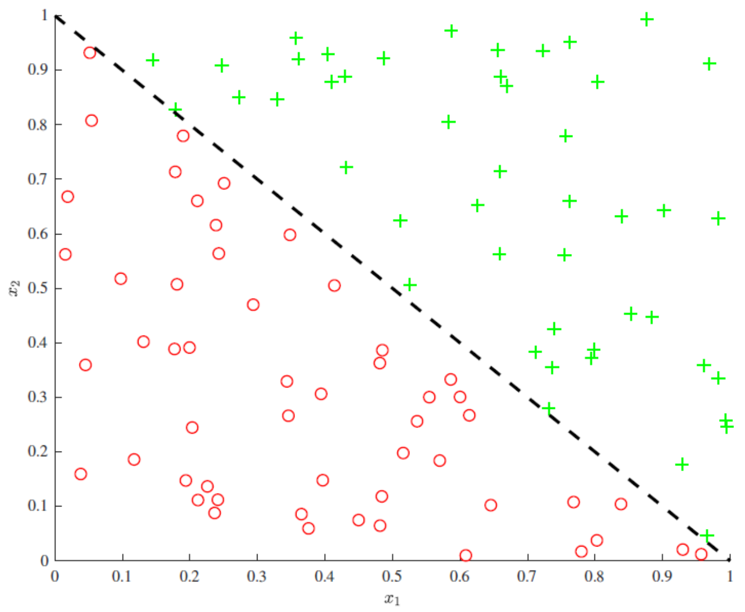
- For example, a simple decision boundary of the form $x_1 + x_2$ could only be approximated through the use of many splits
- On the other hand, a linear model could directly derive this boundary

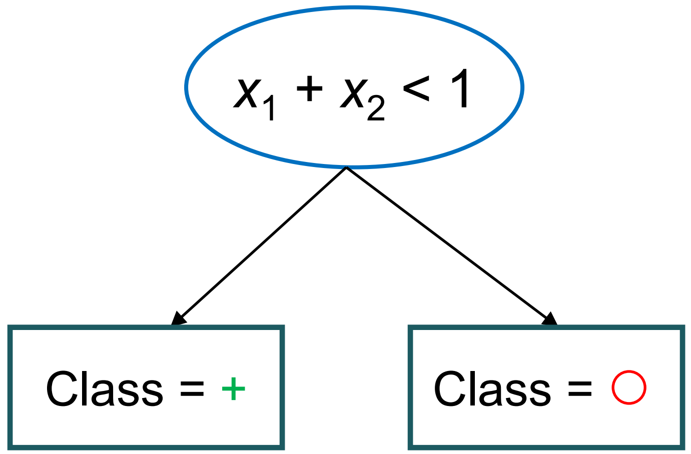




Oblique Decision Trees

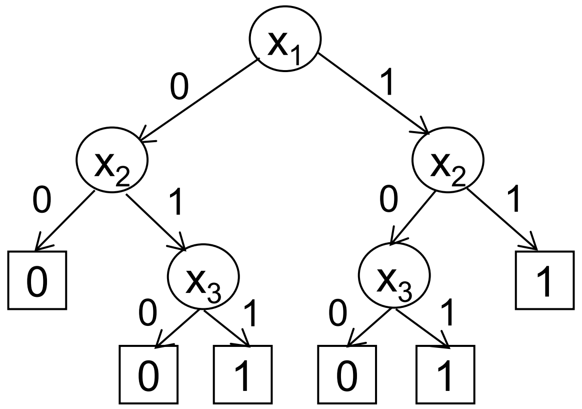
- A test condition may involve multiple features
  - More expressive representation
  - Finding optimal test condition is computationally expensive
  - May increase variance and reduce interpretability





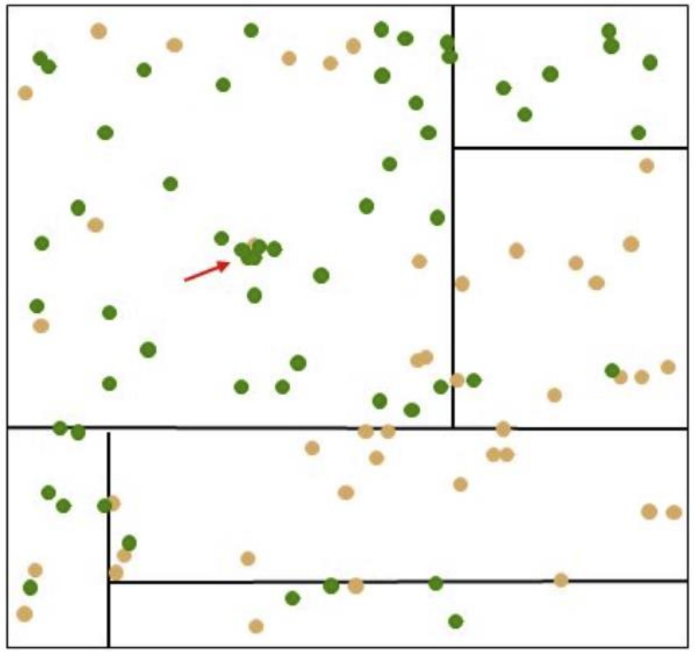
Expressiveness

- Decision trees provide expressive representations for discrete-valued functions
- However, they don't generalize well to certain types of Boolean functions
- Example: **the majority function**
  - Class = 1 if more than half of the Boolean attributes are True
  - Class = 0 otherwise
- For accurate modeling, we need an exponentially large decision tree
- A simple linear model (e.g., perceptron) can represent this function very compactly

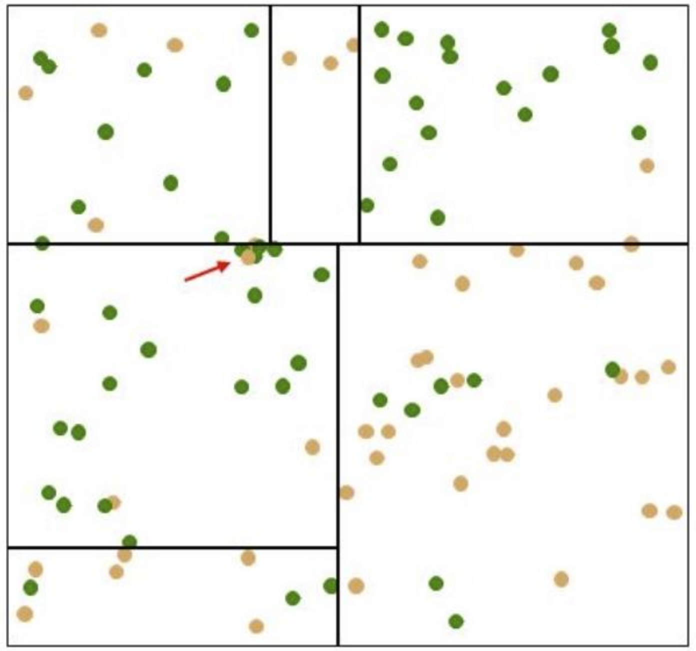


Instability

- Small changes in the training data can cause large changes in the topology of the tree
  - However, the overall performance of the tree remains stable


Accuracy = 81%

One reversal


Accuracy = 80%

Decision Tree Algorithms

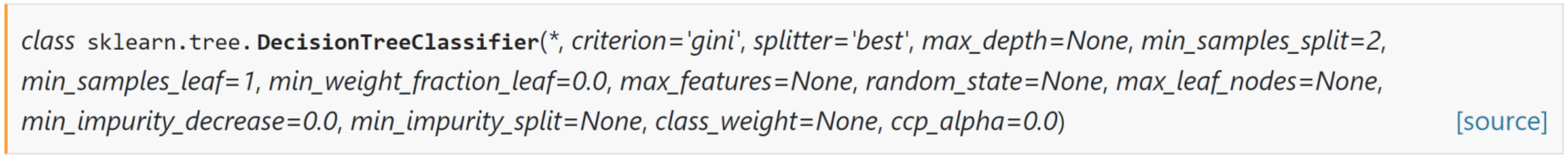
- There is a large variety of decision tree algorithms
- The most popular ones are ID3, C4.5, and CART
- Most decision tree algorithms differ in the following aspects:
  - Node impurity measure (information gain, gini impurity)
  - Binary splits vs. multi-way splits
  - Discrete vs. continuous variables
  - Pre-pruning vs. post-pruning

Decision Tree Algorithms

Algorithm	Author	Impurity measures	Other features
ID3 (Iterative Dichotomizer 3)	Quinlan, 1986	Information gain	Supports only discrete features Multi-category splits No pruning
C4.5	Quinlan, 1993	Information gain Gain ratio	Supports both continuous and discrete features Handles missing values Multi-way splits Performs post-pruning (bottom-up pruning) Generates decision rules from the tree
CART (Classification and Regression Trees)	Breiman et al, 1984	Gini impurity	Can be used for both classification and regression Supports both continuous and discrete features Handles missing values Strictly binary splits Performs cost-complexity pruning

Decision Trees in Scikit-Learn

- DecisionTreeClassifier implements an optimized version of the CART algorithm



```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, ccp_alpha=0.0)                  [source]
```

Argument	**Description**
criterion	The function to measure the quality of a split. Supported criteria are “gini” and “entropy”
splitter	The strategy used to choose the split at each node. Supported strategies are “best” and “random”.
max_depth	The maximum depth of the tree
min_samples_split	The minimum number of samples required to split an internal node
min_samples_leaf	The minimum number of samples required to be at a leaf node
max_features	The number of features to consider when looking for the best split
max_leaf_nodes	Maximum number of leaf nodes
min_impurity_decrease	A node will be split if this split induces a decrease of the impurity greater than this value
ccp_alpha	Complexity parameter used for cost-complexity pruning

Decision Trees in Scikit-Learn

- For example, let’s build a decision tree classifier for the Iris data set:

```python
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```python
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

print(f'Training set accuracy: {clf.score(X_train, y_train):.4f}')
print(f'Test set accuracy: {clf.score(X_test, y_test):.4f}')
```

```
Training set accuracy: 1.0000
Test set accuracy: 0.9737
```

- Note that scaling isn’t required

Decision Tree Prediction

- After being fitted, the tree can be used to predict the class of new samples:

```
new_X = [[6, 3, 5, 1.5]]
clf.predict(new_X)
```

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

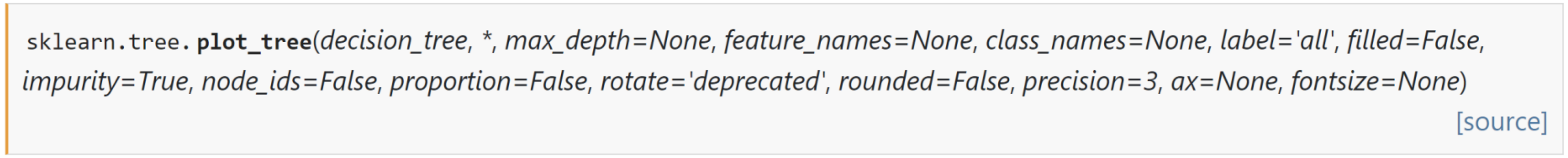
- The tree can also output class probabilities, which are based on the fraction of training samples of each class in the leaf node:

```
clf.predict_proba(new_X)
```

```
array([[0., 1., 0.]])
```

Visualizing the Decision Tree

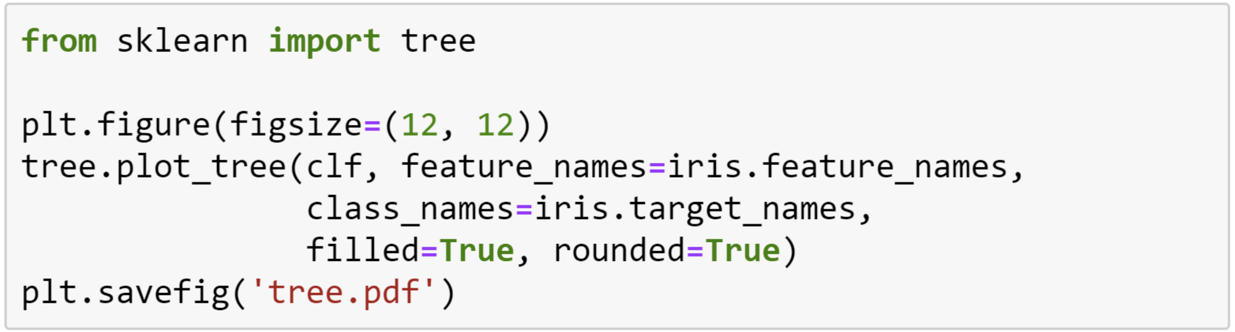
- To visualize the decision tree you can use the following function from sklearn.tree:



```
sklearn.tree.plot_tree(decision_tree, *, max_depth=None, feature_names=None, class_names=None, label='all', filled=False,
impurity=True, node_ids=False, proportion=False, rotate='deprecated', rounded=False, precision=3, ax=None, fontsize=None)
```

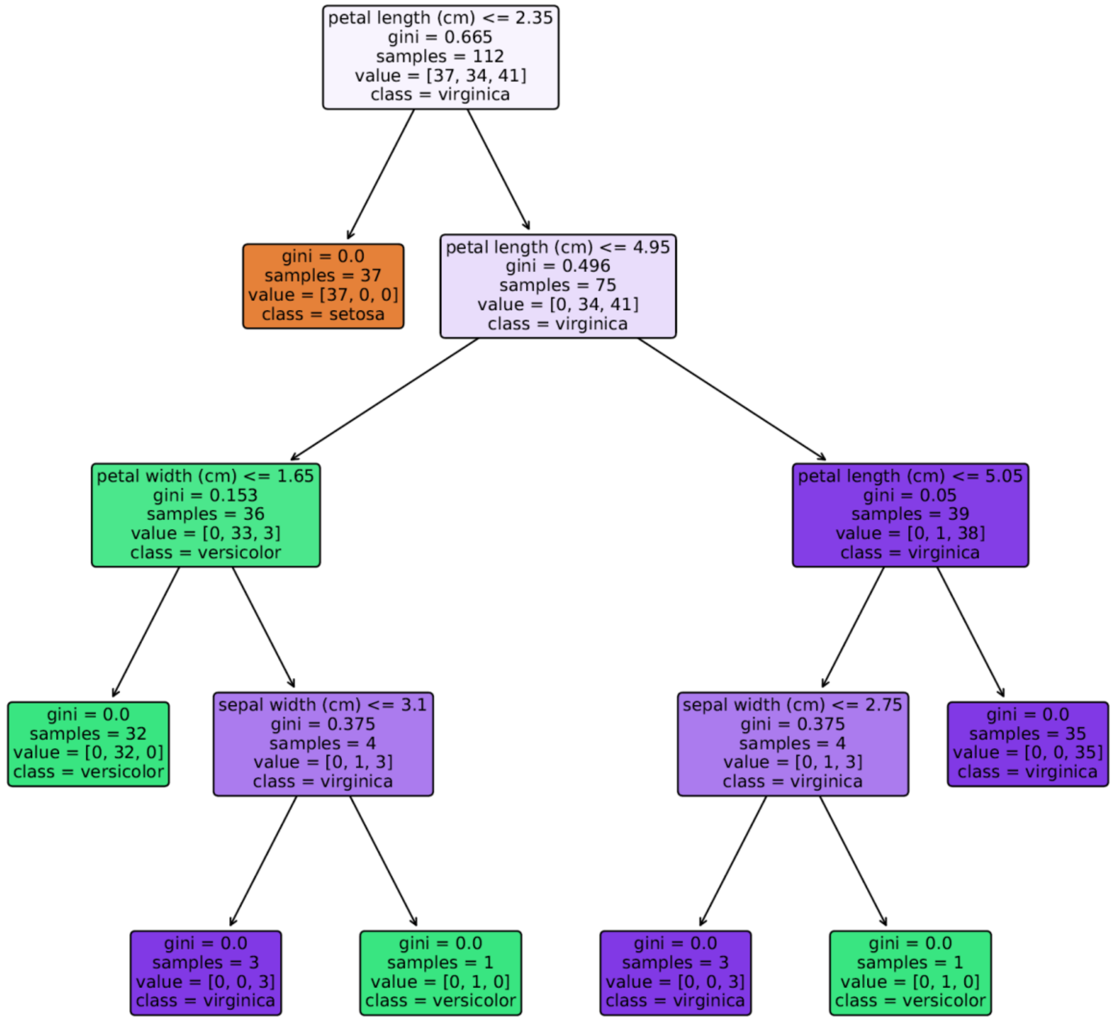
[source]

- For example, to visualize the decision tree for the Iris classification problem:



```python
from sklearn import tree

plt.figure(figsize=(12, 12))
tree.plot_tree(clf, feature_names=iris.feature_names,
               class_names=iris.target_names,
               filled=True, rounded=True)
plt.savefig('tree.pdf')
```

Visualizing the Decision Tree



Plotting Decision Boundaries

- In the case of 2D data, we can also plot the decision boundaries
- For example, let's build a decision tree classifier using only the first two features of the Iris data set (sepal length and sepal width)

```
X = iris.data[:, :2]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

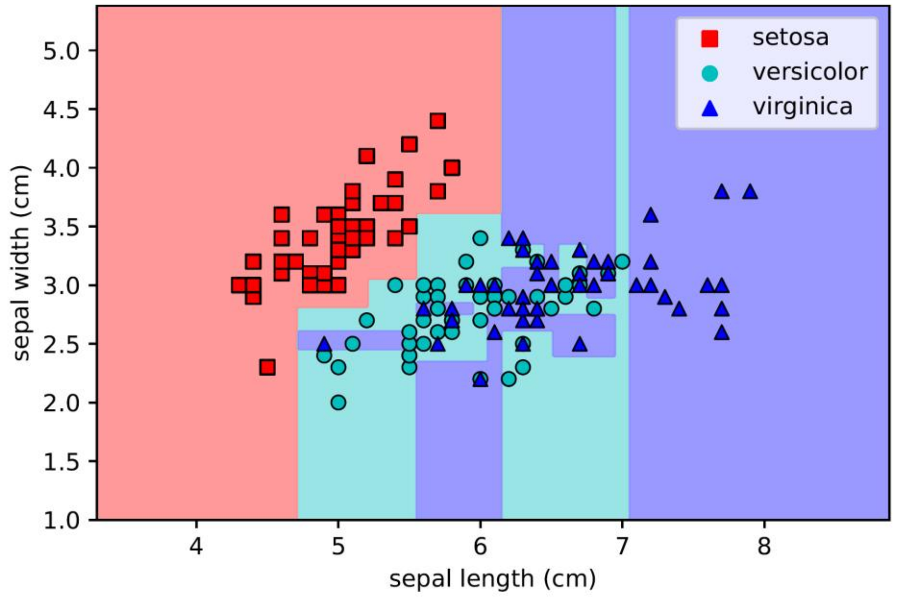
```
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

print(f'Training set accuracy: {clf.score(X_train, y_train):.4f}')
print(f'Test set accuracy: {clf.score(X_test, y_test):.4f}')
```

```
Training set accuracy: 0.9464
Test set accuracy: 0.6053
```

Plotting Decision Boundaries

- We can now plot the decision boundaries using the same function as before:

```
plot_decision_boundaries(clf, X, y, iris.feature_names, iris.target_names)
```



- Clearly, this tree is overfitting the training data

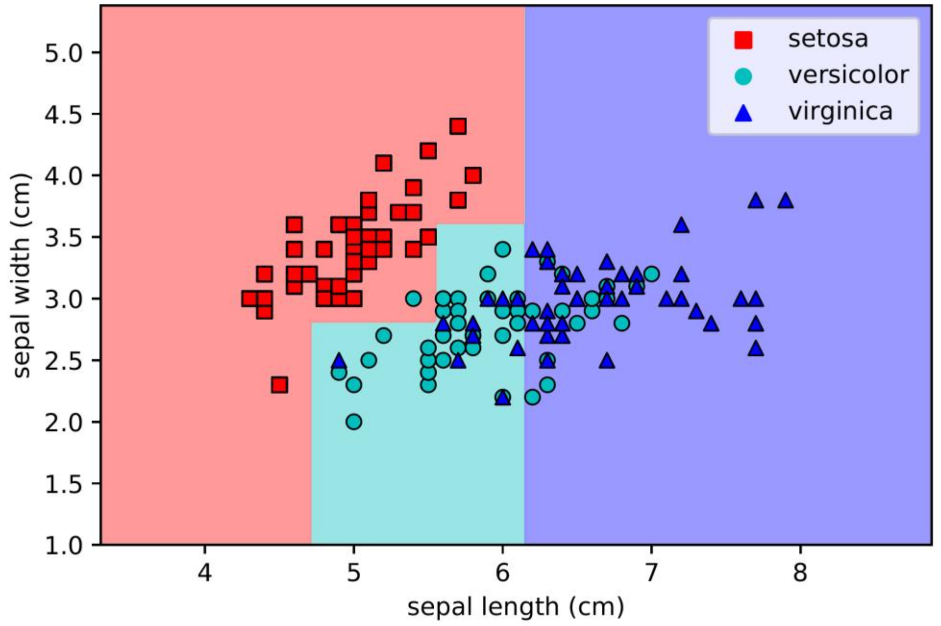
Tree Pruning

- To avoid overfitting, we can restrict the size of the decision tree during training
- For example, let's set its maximum depth to 3

```
clf = DecisionTreeClassifier(max_depth=3)
clf.fit(X_train, y_train)

print(f'Training set accuracy: {clf.score(X_train, y_train):.4f}')
print(f'Test set accuracy: {clf.score(X_test, y_test):.4f}')
```

```
Training set accuracy: 0.8304
Test set accuracy: 0.7632
```

```
plot_decision_boundaries(clf, X, y, iris.feature_names,
                         iris.target_names)
```



Regression Trees

- Decision trees are also capable of performing regression tasks
- The most common split criterion in regression trees is MSE (mean squared error)
- The MSE at node *v* is defined as the mean squared difference between the labels of the samples at node *v* and their average:

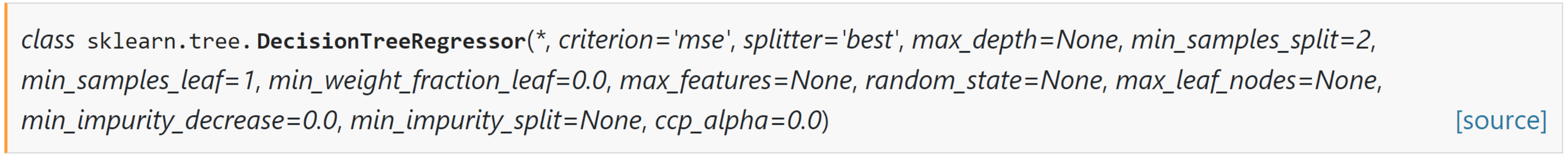
$$\bar{y}_v = \frac{1}{n_v} \sum_{(\mathbf{x}_i, y_i) \in v} y_i$$

$$\mathrm{MSE}(v) = \sum_{(\mathbf{x}_i, y_i) \in v} (y_i - \bar{y}_v)^2$$

- We choose the split that achieves the most reduction in MSE
- The predicted value at a leaf node is the mean value of the labels at that node

DecisionTreeRegressor

- Scikit-Learn’s DecisionTreeRegressor implements a decision tree regressor



```
class sklearn.tree.DecisionTreeRegressor(*, criterion='mse', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, ccp_alpha=0.0)                                        [source]
```

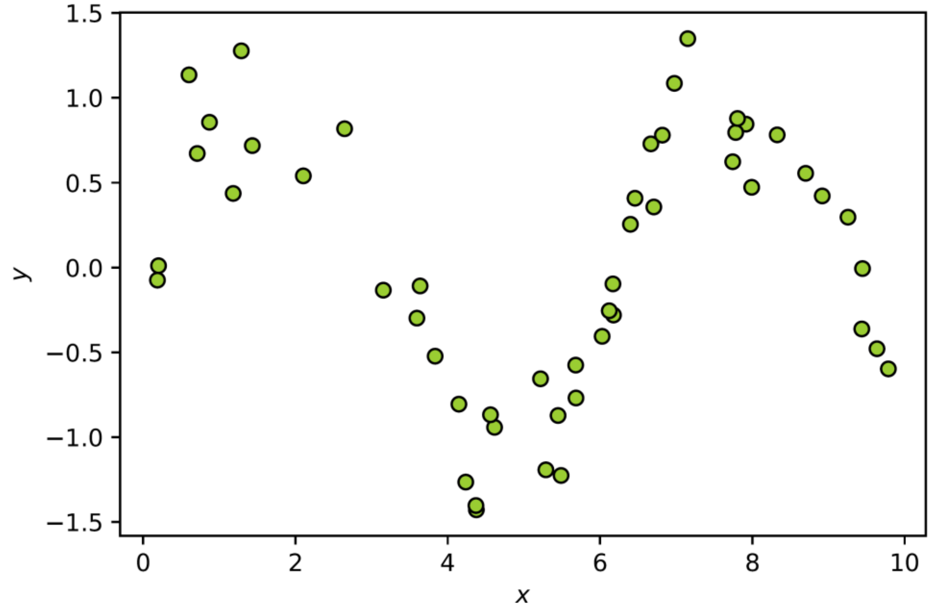
- It has similar parameters to DecisionTreeClassifier, except for the splitting criterion that has the following options:
  - ‘mse’ – minimizes the L2 loss using the mean of each leaf node
  - ‘friedman_mse’ – uses MSE with Friedman’s improvement score for potential splits
  - ‘mae’ – minimizes the L1 loss using the median of each leaf node
  - ‘poisson’ – uses reduction in Poisson deviance to find splits

DecisionTreeRegressor

- For example, let's fit a regression tree to our noisy sine curve:

```
def make_data(n=50):
    rng = np.random.RandomState(0)
    X = rng.rand(n, 1) * 10
    err = rng.normal(size=n) * 0.3
    y = np.sin(X).ravel() + err
    return X, y
```

```
X, y = make_data()
plt.scatter(X, y, color='yellowgreen', edgecolor='k')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.savefig('figures/sine.pdf')
```



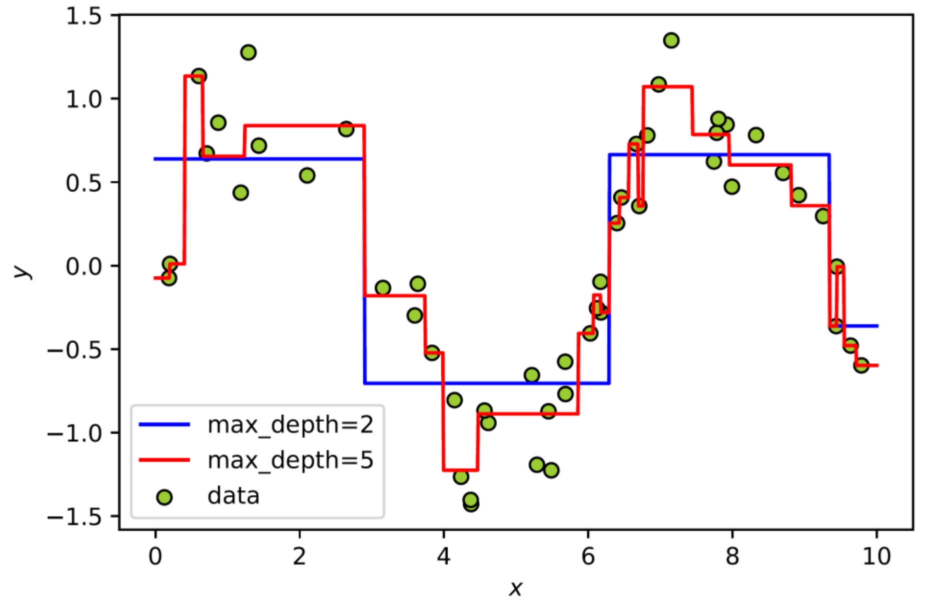
DecisionTreeRegressor

- We'll fit two decision trees with max depths 2 and 5:

```python
from sklearn.tree import DecisionTreeRegressor

reg1 = DecisionTreeRegressor(max_depth=2)
reg2 = DecisionTreeRegressor(max_depth=5)

reg1.fit(X, y)
reg2.fit(X, y)
```

```
DecisionTreeRegressor(max_depth=5)
```

```python
X_test = np.linspace(0, 10, 1000).reshape(-1, 1)
y1_test = reg1.predict(X_test)
y2_test = reg2.predict(X_test)

plt.scatter(X, y, color='yellowgreen', edgecolor='k', label='data')
plt.plot(X_test, y1_test, color='b', label='max_depth=2')
plt.plot(X_test, y2_test, color='r', label='max_depth=5')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend()
plt.savefig('figures/dt_regressor.pdf')
```



Decision Trees Summary

Pros

- Easy to understand and interpret
  - Trees can be visualized
- Capable of fitting complex and large data sets
- Classifying a new sample is very fast
  - Logarithmic in the size of the training set
- Able to handle various types of features
- Requires little data preparation
  - e.g., no scaling is required
- Can deal with redundant attributes
- Capable of both binary and multiclass classification, and regression

Cons

- Can create over-complex trees that don't generalize well (overfitting)
  - Tree pruning is necessary to avoid this problem
- Constrained to rectilinear decision boundaries
- Not guaranteed to return the globally optimal solution (a greedy algorithm)
- Unstable (small variations in the data cause large changes in the tree)
- Some concepts are hard to learn by decision trees, such as parity or majority
- Trees may be biased if some classes dominate