1. Decision trees.
Decision trees
**K-D tree**: data structure for supporting NN search in $\mathbb{R}^d$ (and hence for implementing NN classifiers where $\mathcal{X} = \mathbb{R}^d$).

**Construction procedure**

Given points $S \subset \mathbb{R}^d$:

1. Pick a coordinate $j \in \{1, 2, \ldots, d\}$.
2. Let $m$ be the median of $\{x_j : x \in S\}$.
3. Split points into halves:
   - $L := \{x \in S : x_j < m\}$,
   - $R := \{x \in S : x_j \geq m\}$.
4. Recurse on $L$ and $R$. 

Fast and loose alternative: defeatist search—route test point to a single leaf without any backtracking; might not return actual NN.
**Tree structures for fast NN search in \( \mathbb{R}^d \)**

**K-D tree**: data structure for supporting NN search in \( \mathbb{R}^d \) (and hence for implementing NN classifiers where \( \mathcal{X} = \mathbb{R}^d \)).

**Construction procedure**

Given points \( S \subset \mathbb{R}^d \):

1. Pick a coordinate \( j \in \{1, 2, \ldots, d\} \).
2. Let \( m \) be the median of \( \{x_j : x \in S\} \).
3. Split points into halves:
   
   \[
   L := \{x \in S : x_j < m\}, \\
   R := \{x \in S : x_j \geq m\}.
   \]
4. Recurse on \( L \) and \( R \).

Searching a K-D tree for a **NN**: requires some *backtracking*, still fast when \( d \) is small, but can be slow when \( d \) is large.
**Tree structures for fast NN search in \( \mathbb{R}^d \)**

**K-D tree:** data structure for supporting NN search in \( \mathbb{R}^d \) (and hence for implementing NN classifiers where \( \mathcal{X} = \mathbb{R}^d \)).

**Construction procedure**
Given points \( S \subset \mathbb{R}^d \):

1. Pick a coordinate \( j \in \{1, 2, \ldots, d\} \).
2. Let \( m \) be the median of \( \{x_j : x \in S\} \).
3. Split points into halves:
   
   \[
   L := \{x \in S : x_j < m\}, \\
   R := \{x \in S : x_j \geq m\}.
   \]

4. Recurse on \( L \) and \( R \).

Searching a K-D tree for a NN: requires some *backtracking*,
still fast when \( d \) is small, but can be slow when \( d \) is large.

**Fast and loose alternative:** *defeatist search*—route test point to a single leaf *without any backtracking*; might not return actual NN.
**K-D tree**: data structure for supporting NN search in $\mathbb{R}^d$ (and hence for implementing NN classifiers where $\mathcal{X} = \mathbb{R}^d$).

**Construction procedure**

Given points $S \subset \mathbb{R}^d$:

1. Pick a coordinate $j \in \{1, 2, \ldots, d\}$.
2. Let $m$ be the median of $\{x_j : x \in S\}$.
3. Split points into halves:
   
   $L := \{x \in S : x_j < m\}$,
   
   $R := \{x \in S : x_j \geq m\}$.

4. Recurse on $L$ and $R$.

Searching a K-D tree for a NN: requires some *backtracking*, still fast when $d$ is small, but can be slow when $d$ is large.

**Fast and loose alternative**: *defeatist search*—route test point to a single leaf *without any backtracking*; might not return actual NN.

**K-D tree construction** doesn’t even look at the labels!
Decision trees

Directly optimize tree structure for good classification.

A decision tree is a function \( f : \mathcal{X} \rightarrow \mathcal{Y} \), represented by a binary tree in which:

- Each tree node is associated with a splitting rule \( h : \mathcal{X} \rightarrow \{0,1\} \).
- Each leaf node is associated with a label \( y \in \mathcal{Y} \).
**Decision trees**

Directly optimize tree structure for good classification.

A **decision tree** is a function $f : \mathcal{X} \to \mathcal{Y}$, represented by a binary tree in which:

- Each **tree node** is associated with a **splitting rule** $h : \mathcal{X} \to \{0, 1\}$.
- Each **leaf node** is associated with a label $y \in \mathcal{Y}$.

When $\mathcal{X} = \mathbb{R}^d$, typically only consider splitting rules of the form

$$h(x) = 1\{x_i > t\}$$

for some $i \in [d]$ and $t \in \mathbb{R}$.

Called **axis-aligned**, **coordinate**, or **Stoller** splits.

(Notation: $[d] := \{1, 2, \ldots, d\}$)
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 = \text{ratio of sepal length to width}$
- $x_2 = \text{ratio of petal length to width}$

$\hat{y} = 2$
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

Decision tree example

- Sepal length/width:
  - 1.5
  - 2
  - 2.5
  - 3

- Petal length/width:
  - 2
  - 2.5
  - 3
  - 3.5
  - 4
  - 4.5
  - 5
  - 5.5
  - 6

- $x_1 > 1.7$
Decision tree example

Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 = \text{ratio of sepal length to width}$
- $x_2 = \text{ratio of petal length to width}$

$X = R^2, Y = \{1, 2, 3\}$
Decision tree example

Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

$x_1 > 1.7$
$\hat{y} = 1$

$x_2 > 2.8$
Decision tree example

Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

$x_1 > 1.7 \Rightarrow \hat{y} = 1$
$x_2 > 2.8 \Rightarrow \hat{y} = 2, \hat{y} = 3$
Basic decision tree learning algorithm

- Initially, tree is a single leaf node containing all (training) data.
- Loop:
  - Pick the leaf $\ell$ and rule $h$ that **maximally reduces uncertainty**.
  - Split data in $\ell$ using $h$, and grow tree accordingly.

  ... until some **stopping criterion** is satisfied.

[Label of a leaf is the **plurality label** among the data contained in the leaf.]
Notions of uncertainty: binary case ($\mathcal{Y} = \{0, 1\}$)

Suppose in a set of examples $S \subseteq \mathcal{X} \times \{0, 1\}$, a $p$ fraction are labeled as 1.

1. **Classification error:**
   \[
u(S) := \min\{p, 1 - p\}\]

2. **Gini index:**
   \[
u(S) := 2p(1 - p)\]

3. **Entropy:**
   \[
u(S) := p \log \frac{1}{p} + (1 - p) \log \frac{1}{1 - p}\]

Gini index and entropy (after some rescaling) are concave upper-bounds on classification error.
Notions of uncertainty: general case

Suppose in $S \subseteq \mathcal{X} \times \mathcal{Y}$, a $p_k$ fraction are labeled as $k$ (for each $k \in \mathcal{Y}$).

1. **Classification error:**
   $$u(S) := 1 - \max_{k \in \mathcal{Y}} p_k$$

2. **Gini index:**
   $$u(S) := 1 - \sum_{k \in \mathcal{Y}} p_k^2$$

3. **Entropy:**
   $$u(S) := \sum_{k \in \mathcal{Y}} p_k \log \frac{1}{p_k}$$
Notions of uncertainty: general case

Suppose in $S \subseteq X \times Y$, a $p_k$ fraction are labeled as $k$ (for each $k \in Y$).

1. **Classification error:**

   $$u(S) := 1 - \max_{k \in Y} p_k$$

2. **Gini index:**

   $$u(S) := 1 - \sum_{k \in Y} p_k^2$$

3. **Entropy:**

   $$u(S) := \sum_{k \in Y} p_k \log \frac{1}{p_k}$$

Each is *maximized* when $p_k = 1/|Y|$ for all $k \in Y$ (i.e., equal numbers of each label in $S$).

Each is *minimized* when $p_k = 1$ for a single label $k \in Y$ (so $S$ is pure in label).
Uncertainty reduction

Suppose the data $S$ at a leaf $\ell$ is split by a rule $h$ into $S_L$ and $S_R$, where $p_L := |S_L|/|S|$ and $p_R := |S_R|/|S|$.

The reduction in uncertainty from using rule $h$ at leaf $\ell$ is

$$u(S) - (p_L \cdot u(S_L) + p_R \cdot u(S_R))$$
Suppose the data $S$ at a leaf $\ell$ is split by a rule $h$ into $S_L$ and $S_R$, where $p_L := |S_L|/|S|$ and $p_R := |S_R|/|S|$.

The reduction in uncertainty from using rule $h$ at leaf $\ell$ is

$$u(S) - \left( p_L \cdot u(S_L) + p_R \cdot u(S_R) \right).$$
One leaf (with \( \hat{y} = 1 \)) already has zero uncertainty (a pure leaf).
One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

Other leaf (with $\hat{y} = 3$) has Gini index

$$u(S) = 1 - \left( \frac{1}{101} \right)^2 - \left( \frac{50}{101} \right)^2 - \left( \frac{50}{101} \right)^2$$

$$= 0.5098.$$
One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

Other leaf (with $\hat{y} = 3$) has Gini index

$$u(S) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$

$$= 0.5098.$$ 

Split $S$ with $\{x_1 > t\}$ to $S_L, S_R$: 

\[ 1.6 \ 1.8 \ 2 \ 2.2 \ 2.4 \ 2.6 \ 2.8 \ 3 \]

\[ 0 \ 0.005 \ 0.01 \ 0.015 \ 0.02 \]
One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

Other leaf (with $\hat{y} = 3$) has Gini index

$$u(S) = 1 - \left( \frac{1}{101} \right)^2 - \left( \frac{50}{101} \right)^2 - \left( \frac{50}{101} \right)^2$$

$$= 0.5098.$$ 

Split $S$ with $\{x_2 > t\}$ to $S_L, S_R$: 

$$x_1 > 1.7$$

$$\hat{y} = 1$$

$$\hat{y} = 3$$
Uncertainty reduction

One leaf (with \( \hat{y} = 1 \)) already has zero uncertainty (a pure leaf).

Other leaf (with \( \hat{y} = 3 \)) has Gini index

\[
u(S) = 1 - \left( \frac{1}{101} \right)^2 - \left( \frac{50}{101} \right)^2 - \left( \frac{50}{101} \right)^2 = 0.5098.
\]

Split \( S \) with \( 1\{x_2 > 2.7222\} \) to \( S_L, S_R \):

\[
u(S_L) = 1 - \left( \frac{0}{30} \right)^2 - \left( \frac{1}{30} \right)^2 - \left( \frac{29}{30} \right)^2 = 0.0605,
\]

\[
u(S_R) = 1 - \left( \frac{1}{71} \right)^2 - \left( \frac{49}{71} \right)^2 - \left( \frac{21}{71} \right)^2 = 0.4197.
\]

Reduction in uncertainty:

\[
0.5098 - \left( \frac{30}{101} \cdot 0.0605 + \frac{71}{101} \cdot 0.4197 \right) = 0.2039.
\]
Comparing notions of uncertainty

It is possible to have a splitting rule \( h \) with

- zero reduction in classification error, but
- non-zero reduction in Gini index or entropy.
Comparing notions of uncertainty

It is possible to have a splitting rule $h$ with

- zero reduction in classification error, but
- non-zero reduction in Gini index or entropy.

(data $S$ at leaf $\ell$

$1/3$ of $S$ have $y = 1$

$2/3$ of $S$ have $h(x) = 0$

$1/3$ of $S$ have $h(x) = 1$

$S_L$

$2/5$ of $S_L$ have $y = 1$

$S_R$

$1/5$ of $S_R$ have $y = 1$

$(\mathcal{Y} = \{0, 1\})$
Comparing notions of uncertainty

It is possible to have a splitting rule $h$ with

- zero reduction in classification error, but
- non-zero reduction in Gini index or entropy.

\[
\begin{align*}
\text{data } S & \text{ at leaf } \ell \\
& \frac{1}{3} \text{ of } S \text{ have } y = 1 \\
\frac{2}{3} \text{ of } S & \text{ have } h(x) = 0 \\
\frac{1}{3} \text{ of } S & \text{ have } h(x) = 1 \\
\end{align*}
\]

\[
\begin{align*}
S_L & \text{ have } \frac{2}{5} \text{ of } S_L \text{ have } y = 1 \\
S_R & \text{ have } \frac{1}{5} \text{ of } S_R \text{ have } y = 1 \\
(\mathcal{Y} = \{0, 1\})
\end{align*}
\]

Reduction in classification error:

\[
\frac{1}{3} - \left( \frac{2}{3} \cdot \frac{2}{5} + \frac{1}{3} \cdot \frac{1}{5} \right) = 0
\]

Reduction in Gini index:

\[
\frac{4}{9} - \left( \frac{2}{3} \cdot \frac{12}{25} + \frac{1}{3} \cdot \frac{8}{25} \right) = \frac{4}{225}
\]
ASIDE: REDUCTION IN ENTROPY

The (Shannon) **entropy** of \( \mathcal{Z} \)-valued random variable \( Z \) with is

\[
H(Z) := \sum_{z \in \mathcal{Z}} \Pr[Z = z] \log \frac{1}{\Pr[Z = z]}.
\]

Think of \((X, Y)\) as random pair taking value \((x, y)\) \( \in S \) with probability \( \frac{1}{|S|} \).

Using entropy as uncertainty measure, \( u(S) = H(Y) \).

Reduction in uncertainty after a split \( h: X \rightarrow \{0, 1\} \) is

\[
H(Y) - \left( \Pr[h(X) = 0] \cdot H(Y | h(X) = 0) + \Pr[h(X) = 1] \cdot H(Y | h(X) = 1) \right)
\]

This is called **mutual information** between \( Y \) and \( h(X) \).

More on information theory later in the course.
Aside: Reduction in Entropy

The (Shannon) **entropy** of \( Z \)-valued random variable \( Z \) with is

\[
H(Z) := \sum_{z \in Z} \Pr[Z = z] \log \frac{1}{\Pr[Z = z]}.
\]

The **conditional entropy** of \( Z \) given a \( W \)-valued random variable \( W \) is

\[
H(Z|W) := \sum_{w \in W} \Pr[W = w] \cdot H(Z|W = w).
\]

(Weighted average of entropies of random variables of form \( Z|W = w \).)
Aside: reduction in entropy

The (Shannon) entropy of $Z$-valued random variable $Z$ with is

$$H(Z) := \sum_{z \in Z} \Pr[Z = z] \log \frac{1}{\Pr[Z = z]}.$$  

The conditional entropy of $Z$ given a $W$-valued random variable $W$ is

$$H(Z|W) := \sum_{w \in W} \Pr[W = w] \cdot H(Z|W = w).$$

(Weighted average of entropies of random variables of form $Z|W = w$.)

Think of $(X, Y)$ as random pair taking value $(x, y) \in S$ with probability $1/|S|$. Using entropy as uncertainty measure, $u(S) = H(Y)$. 

---

**Aside: reduction in entropy**

The (Shannon) **entropy** of $\mathcal{Z}$-valued random variable $Z$ with is

$$H(Z) := \sum_{z \in \mathcal{Z}} \Pr[Z = z] \log \frac{1}{\Pr[Z = z]}.$$  

The **conditional entropy** of $Z$ given a $\mathcal{W}$-valued random variable $W$ is

$$H(Z|W) := \sum_{w \in \mathcal{W}} \Pr[W = w] \cdot H(Z|W = w).$$

(Weighted average of entropies of random variables of form $Z|W = w$.)

Think of $(X, Y)$ as random pair taking value $(x, y) \in S$ with probability $1/|S|$. Using entropy as uncertainty measure, $u(S) = H(Y)$. 

---
Aside: reduction in entropy

The (Shannon) entropy of $Z$-valued random variable $Z$ with is

$$H(Z) := \sum_{z \in Z} \Pr[Z = z] \log \frac{1}{\Pr[Z = z]}.$$  

The conditional entropy of $Z$ given a $W$-valued random variable $W$ is

$$H(Z|W) := \sum_{w \in W} \Pr[W = w] \cdot H(Z|W = w).$$

(Weighted average of entropies of random variables of form $Z|W = w$.)

Think of $(X, Y)$ as random pair taking value $(x, y) \in S$ with probability $1/|S|$. Using entropy as uncertainty measure, $u(S) = H(Y)$.

Reduction in uncertainty after a split $h: \mathcal{X} \to \{0, 1\}$ is

$$H(Y) - \left( \Pr[h(X) = 0] \cdot H(Y|h(X) = 0) + \Pr[h(X) = 1] \cdot H(Y|h(X) = 1) \right)$$

$$= H(Y) - H(Y|h(X)).$$

This is called mutual information between $Y$ and $h(X)$. 

More on information theory later in the course.
The (Shannon) **entropy** of $Z$-valued random variable $Z$ with is

$$H(Z) := \sum_{z \in Z} \Pr[Z = z] \log \frac{1}{\Pr[Z = z]}.$$ 

The **conditional entropy** of $Z$ given a $\mathcal{W}$-valued random variable $W$ is

$$H(Z|W) := \sum_{w \in \mathcal{W}} \Pr[W = w] \cdot H(Z|W = w).$$

(Weighted average of entropies of random variables of form $Z|W = w$.)

Think of $(X, Y)$ as random pair taking value $(x, y) \in S$ with probability $1/|S|$. Using entropy as uncertainty measure, $u(S) = H(Y)$.

**Reduction in uncertainty after a split** $h : \mathcal{X} \rightarrow \{0, 1\}$ is

$$H(Y) - \left( \Pr[h(X) = 0] \cdot H(Y|h(X) = 0) + \Pr[h(X) = 1] \cdot H(Y|h(X) = 1) \right)$$

$$= H(Y) - H(Y|h(X)).$$

This is called **mutual information** between $Y$ and $h(X)$.

More on information theory later in the course.
Suppose \( \mathcal{X} = \mathbb{R}^2 \) and \( \mathcal{Y} = \{ \text{red, blue} \} \), and the data is as follows:

Every split of the form \( 1 \{ x_i > t \} \) provides no reduction in uncertainty (whether based on classification error, Gini index, or entropy).
Suppose $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{\text{red, blue}\}$, and the data is as follows:

Every split of the form $1\{x_i > t\}$ provides no reduction in uncertainty (whether based on classification error, Gini index, or entropy).

**Upshot:**
Zero reduction in uncertainty is not necessarily a desirable stopping condition.
Basic decision tree learning algorithm

- Initially, tree is a single leaf node containing all (training) data.
- Loop:
  - Pick the leaf $\ell$ and rule $h$ that maximally reduces uncertainty.
  - Split data in $\ell$ using $h$, and grow tree accordingly.

... until some stopping criterion is satisfied.

[Label of a leaf is the plurality label among the data contained in the leaf.]
Many alternatives; two common choices are:

1. Stop when the tree reaches a pre-specified size.
   - Involves setting additional “tuning parameters”—use hold-out or cross-validation.

2. Stop when every leaf is pure.
   - (More common.)
   - Serious danger of overfitting—spurious structure due to sampling.
Many alternatives; two common choices are:

1. Stop when the \textit{tree reaches a pre-specified size}. 

Serious danger of overfitting spurious structure due to sampling.
Many alternatives; two common choices are:

1. Stop when the tree reaches a pre-specified size.
   Involves setting additional “tuning parameters”—use hold-out or cross-validation.
Many alternatives; two common choices are:

1. Stop when the tree reaches a pre-specified size.
   
   Involves setting additional “tuning parameters”—use hold-out or cross-validation.

2. Stop when every leaf is pure. (More common.)
Stopping criterion

Many alternatives; two common choices are:

1. Stop when the tree reaches a pre-specified size.
   
   Involves setting additional “tuning parameters”—use hold-out or cross-validation.

2. Stop when every leaf is pure. (More common.)
   
   Serious danger of overfitting spurious structure due to sampling.
Many alternatives; two common choices are:

1. **Stop when the tree reaches a pre-specified size.**
   
   Involves setting additional “tuning parameters”—use hold-out or cross-validation.

2. **Stop when every leaf is pure.** (More common.)
   
   Serious danger of **overfitting** spurious structure due to sampling.
Many alternatives; two common choices are:

1. **Stop when the tree reaches a pre-specified size.**
   
   Involves setting additional “tuning parameters”—use hold-out or cross-validation.

2. **Stop when every leaf is pure.** (More common.)

   Serious danger of overfitting spurious structure due to sampling.
Many alternatives; two common choices are:

1. **Stop when the tree reaches a pre-specified size.**
   
   Involves setting additional “tuning parameters”—use hold-out or cross-validation.

2. **Stop when every leaf is pure.** (More common.)
   
   Serious danger of **overfitting** spurious structure due to sampling.
Overfitting

- **Training error** goes to zero as the number of nodes in the tree increases.
- **True error** decreases initially, but eventually **increases due to overfitting**.
Preventing overfitting

Split training data \( S \) into two parts, \( S' \) and \( S'' \):

- Use first part \( S' \) to **grow the tree until all leaves are pure**.
- Use second part \( S'' \) to **choose a good pruning of the tree**.
What can be done about overfitting?

Preventing overfitting
Split training data $S$ into two parts, $S'$ and $S''$:

- Use first part $S'$ to **grow the tree until all leaves are pure**.
- Use second part $S''$ to **choose a good pruning of the tree**.

Pruning algorithm
Loop:

- Replace any tree node by a leaf node if it improves the error on $S''$. 

... until no more such improvements possible.
What can be done about overfitting?

Preventing overfitting
Split training data $S$ into two parts, $S'$ and $S''$:
- Use first part $S'$ to **grow the tree until all leaves are pure**.
- Use second part $S''$ to **choose a good pruning of the tree**.

Pruning algorithm
Loop:
- Replace any tree node by a leaf node if it improves the error on $S''$.
...until no more such improvements possible.

This can be done efficiently using **dynamic programming** (bottom-up traversal of the tree).
What can be done about overfitting?

Preventing overfitting
Split training data $S$ into two parts, $S'$ and $S''$:
- Use first part $S'$ to **grow the tree until all leaves are pure**.
- Use second part $S''$ to **choose a good pruning of the tree**.

Pruning algorithm
Loop:
- Replace any tree node by a leaf node if it improves the error on $S''$.

... until no more such improvements possible.

This can be done efficiently using **dynamic programming** (bottom-up traversal of the tree).

**Independence of $S'$ and $S''$ make it unlikely for spurious structures in each to perfectly align.**
Example: Spam filtering

Data

- 4601 e-mail messages, 39.4% are spam.
- \( Y = \{ \text{spam}, \text{not spam} \} \)
- E-mails represented by 57 features:
  - 48: percentage of e-mail words that is specific word (e.g., “free”, “business”)
  - 6: percentage of e-mail characters that is specific character (e.g., “!”).
  - 3: other features (e.g., average length of ALL-CAPS words).

Results

Using variant of greedy algorithm to grow tree; prune tree using validation set.

Chosen tree has just 17 leaves. Test error is 9.3%.

<table>
<thead>
<tr>
<th>( y = ) not spam</th>
<th>( \hat{y} = ) not spam</th>
<th>( \hat{y} = ) spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>57.3%</td>
<td>4.0%</td>
<td></td>
</tr>
<tr>
<td>5.3%</td>
<td>33.4%</td>
<td></td>
</tr>
</tbody>
</table>
Example: Spam filtering

FIGURE 9.5. The pruned tree for the spam example. The split variables are shown in blue on the branches, and the classification is shown in every node. The numbers under the terminal nodes indicate misclassification rates on the test data.
Recap and final notes

- Decision trees are very flexible classifiers (like NN).
  - Certain greedy strategies for training decision trees are consistent: \( \text{err}(\hat{f}) \to \text{err}(f^*) \) as \( n \to \infty \).
  - But also very prone to overfitting in most basic form.
  - (NP-hard to find smallest decision tree consistent with data.)

Next time: how to find simple rules that use several features at a time?
Decision trees are very flexible classifiers (like NN).

- Certain greedy strategies for training decision trees are consistent: 
  \[ \text{err}(\hat{f}) \rightarrow \text{err}(f^*) \text{ as } n \rightarrow \infty. \]

- But also very prone to overfitting in most basic form.

- (NP-hard to find smallest decision tree consistent with data.)

Current theoretical understanding of (greedy) decision tree learning:

- As fitting a non-parametric model (like NN).

- As meta-algorithm for combining classifiers (splitting rules).
Recap and final notes

- Decision trees are very flexible classifiers (like NN).
  - Certain greedy strategies for training decision trees are consistent: \( \text{err}(\hat{f}) \rightarrow \text{err}(f^*) \) as \( n \rightarrow \infty \).
  - But also very prone to overfitting in most basic form.
  - (NP-hard to find smallest decision tree consistent with data.)
- Current theoretical understanding of (greedy) decision tree learning:
  - As fitting a non-parametric model (like NN).
  - As meta-algorithm for combining classifiers (splitting rules).
- Basic form of decision trees only uses splitting rules that only look at a single feature—easy to deal with computationally.
Recap and final notes

- Decision trees are very flexible classifiers (like NN).
  - Certain greedy strategies for training decision trees are consistent: $\text{err}(\hat{f}) \to \text{err}(f^*)$ as $n \to \infty$.
  - But also very prone to overfitting in most basic form.
  - (NP-hard to find smallest decision tree consistent with data.)
- Current theoretical understanding of (greedy) decision tree learning:
  - As fitting a non-parametric model (like NN).
  - As meta-algorithm for combining classifiers (splitting rules).
- Basic form of decision trees only uses splitting rules that only look at a single feature—easy to deal with computationally.

Next time: how to find simple rules that use several features at a time?