COMS 4771 Lecture 4

1. K-D trees for nearest neighbor search.
2. Decision trees.

K-D trees for nearest neighbor search

Tree structures for one-dimensional data

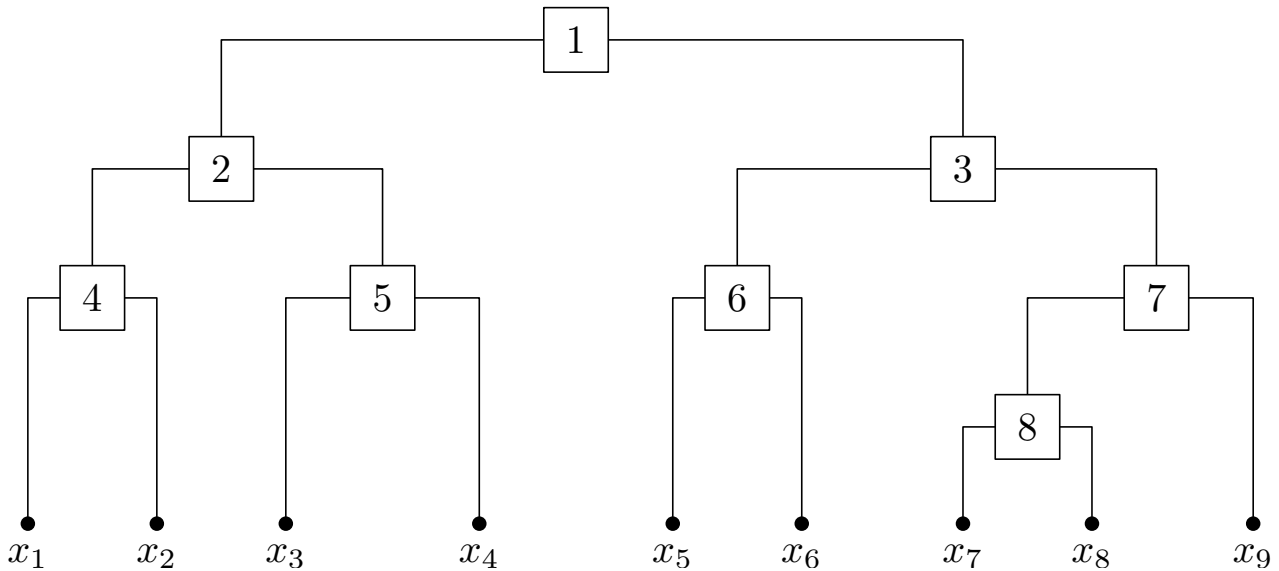
A data structure for fast NN search in $\mathbb{R}^1$

Sort training data so that $x_1 \leq x_2 \leq \cdots \leq x_n$, then construct binary tree.

Tree structures for one-dimensional data

A data structure for fast NN search in $\mathbb{R}^1$

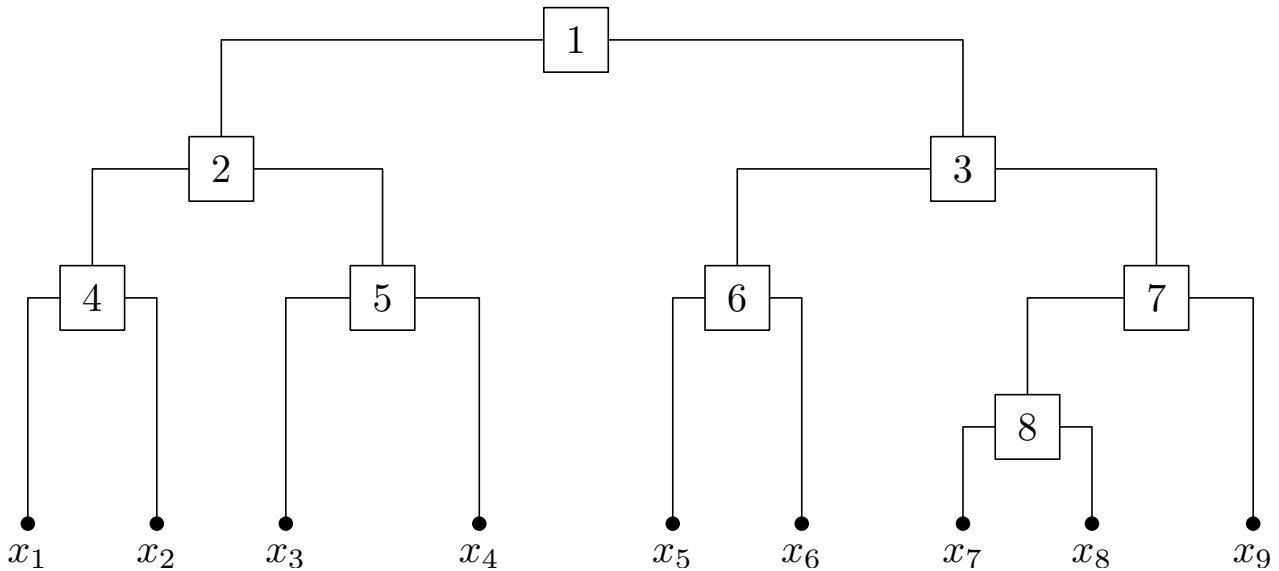
Sort training data so that $x_1 \leq x_2 \leq \cdots \leq x_n$, then construct binary tree.



Tree structures for one-dimensional data

A data structure for fast NN search in $\mathbb{R}^1$

Sort training data so that $x_1 \leq x_2 \leq \cdots \leq x_n$, then construct binary tree.

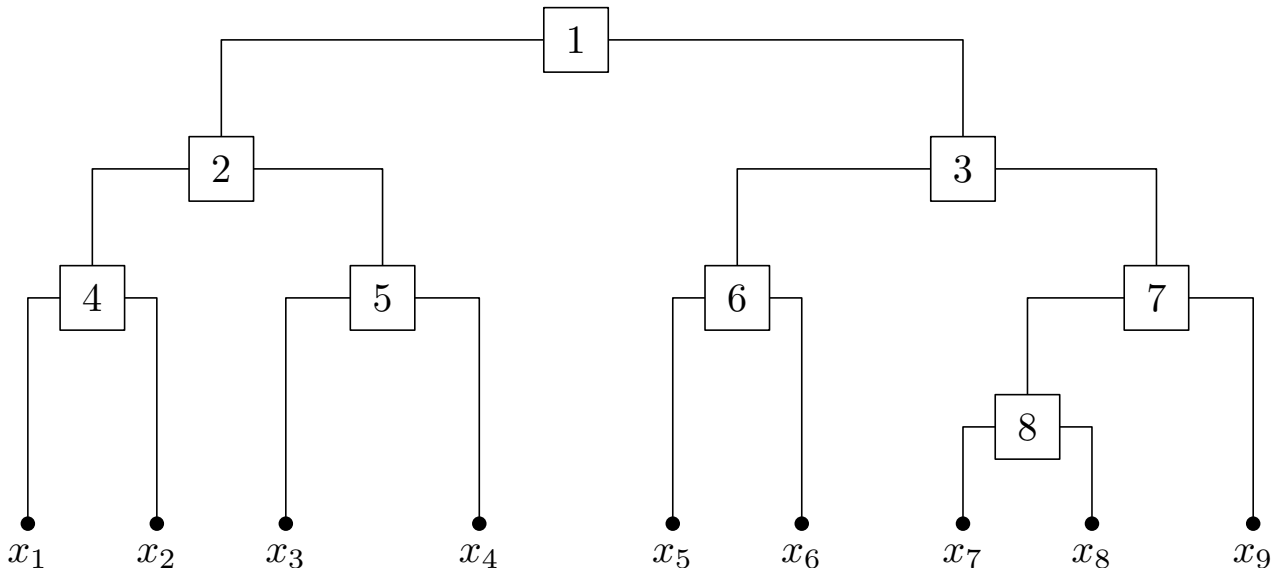


With each tree node, remember **midpoint** between rightmost point in left child, and leftmost point in right child.

Tree structures for one-dimensional data

A data structure for fast NN search in $\mathbb{R}^1$

Sort training data so that $x_1 \leq x_2 \leq \cdots \leq x_n$, then construct binary tree.



With each tree node, remember **midpoint** between rightmost point in left child, and leftmost point in right child. **This permits very efficient NN search.**

If tree is (approximately) balanced, then $O(\log(n))$ time to find NN!

Tree structures for multi-dimensional data

A data structure for fast NN search in $\mathbb{R}^d$, $d > 1$
Many options, but a popular one is the **K-D tree**.

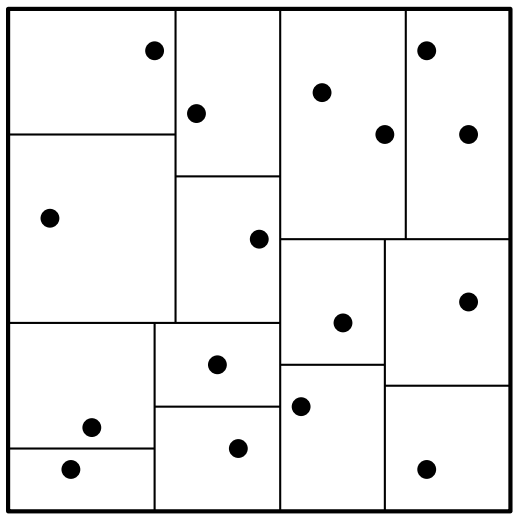
A data structure for fast NN search in $\mathbb{R}^d$, $d > 1$
Many options, but a popular one is the **K-D tree**.

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 $\{x_j : \boldsymbol{x} \in S\}$.
3. Split points into halves:

$$L := \{\boldsymbol{x} \in S : x_j < m\},$$
$$R := \{\boldsymbol{x} \in S : x_j \geq m\}.$$

4. Recurse on L and R.



Tree structures for multi-dimensional data

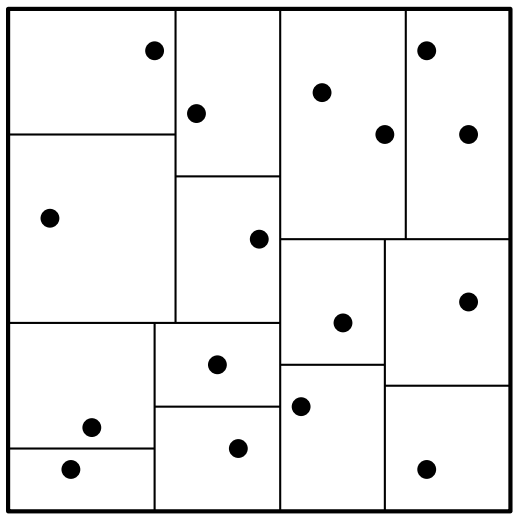
A data structure for fast NN search in $\mathbb{R}^d$, $d > 1$
Many options, but a popular one is the **K-D tree**.

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 $\{x_j : \boldsymbol{x} \in S\}$.
3. Split points into halves:

$$L := \{\boldsymbol{x} \in S : x_j < m\},$$
$$R := \{\boldsymbol{x} \in S : x_j \geq m\}.$$

4. Recurse on L and R.



Easy to lookup points in S (in $O(\log(n))$ time), but how about NN search?

Tree structures for multi-dimensional data

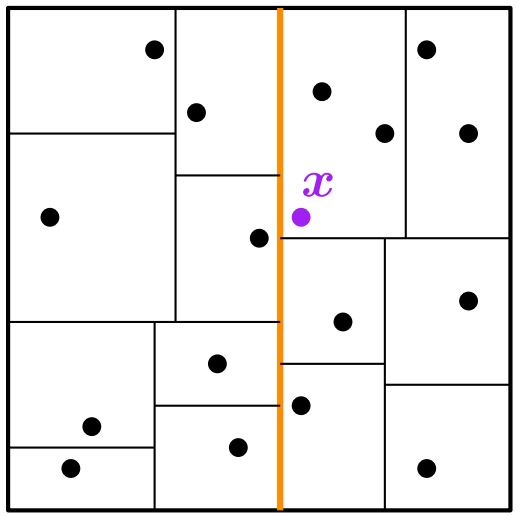
A data structure for fast NN search in $\mathbb{R}^d$, $d > 1$
Many options, but a popular one is the **K-D tree**.

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 $\{x_j : \boldsymbol{x} \in S\}$.
3. Split points into halves:

$$L := \{\boldsymbol{x} \in S : x_j < m\},$$
$$R := \{\boldsymbol{x} \in S : x_j \geq m\}.$$

4. Recurse on L and R.



Easy to lookup points in S (in $O(\log(n))$ time), but how about NN search?

Same $O(\log(n))$-time routing of a test point $\boldsymbol{x} \in \mathbb{R}^d$ is **overly optimistic**:
might not yield the NN!

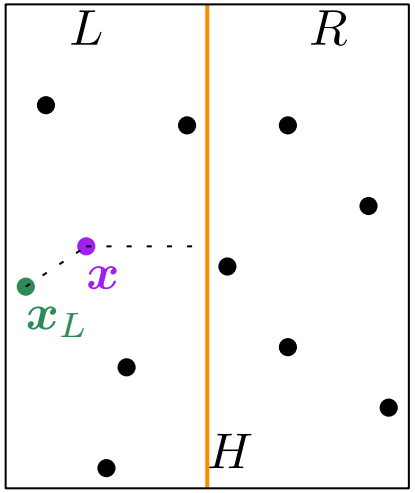
Searching K-D trees using geometry

For K-D trees:
L and R are separated by a hyperplane $H = \left\{ \boldsymbol{z} \in \mathbb{R}^d : z_j = m \right\}$.

Searching K-D trees using geometry

For K-D trees:
L and R are separated by a hyperplane $H = \{\boldsymbol{z} \in \mathbb{R}^d : z_j = m\}$.

Suppose test point $\boldsymbol{x}$ is in L, and the NN of $\boldsymbol{x}$ in L is $\boldsymbol{x}_L$.



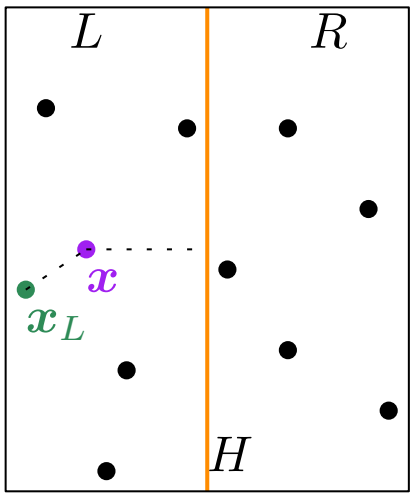
Searching K-D trees using geometry

For K-D trees:
L and R are separated by a hyperplane $H = \{\boldsymbol{z} \in \mathbb{R}^d : z_j = m\}$.

Suppose test point $\boldsymbol{x}$ is in L, and the NN of $\boldsymbol{x}$ in L is $\boldsymbol{x}_L$.

By geometry,

$$\min_{\boldsymbol{x}' \in R} \|\boldsymbol{x} - \boldsymbol{x}'\|_2 \geq \text{distance from } \boldsymbol{x} \text{ to } H$$
$$= |x_j - m|.$$



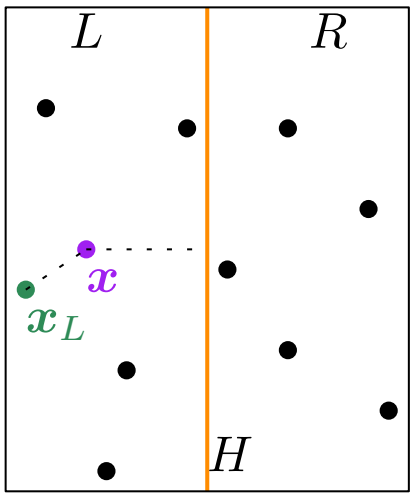
Searching K-D trees using geometry

For K-D trees:
L and R are separated by a hyperplane $H = \{\boldsymbol{z} \in \mathbb{R}^d : z_j = m\}$.

Suppose test point $\boldsymbol{x}$ is in L, and the NN of $\boldsymbol{x}$ in L is $\boldsymbol{x}_L$.

By geometry,

$$\min_{\boldsymbol{x}' \in R} \|\boldsymbol{x} - \boldsymbol{x}'\|_2 \geq \text{distance from } \boldsymbol{x} \text{ to } H$$
$$= |x_j - m|.$$



A valid check: if $\|\boldsymbol{x} - \boldsymbol{x}_L\|_2 < |x_j - m|$, then

$$\|\boldsymbol{x} - \boldsymbol{x}_L\|_2 < \min_{\boldsymbol{x}' \in R} \|\boldsymbol{x} - \boldsymbol{x}'\|_2.$$

In this case, we can skip searching R and immediately return $\boldsymbol{x}_L$.

Efficient NN search?

For certain kinds of binary space partition trees (similar to K-D trees), NN search typically completes in $O(2^d \log(n))$ time.

- Very fast in low dimensions.
- But can be slow in high dimensions.

Efficient NN search?

For certain kinds of binary space partition trees (similar to K-D trees), NN search typically completes in $O(2^d \log(n))$ time.

- Very fast in low dimensions.
- But can be slow in high dimensions.

But NN search is only means to an end—ultimate goal is good classification.
K-D tree construction doesn't even look at the labels!

Efficient NN search?

For certain kinds of binary space partition trees (similar to K-D trees), NN search typically completes in $O(2^d \log(n))$ time.

- Very fast in low dimensions.
- But can be slow in high dimensions.

But NN search is only means to an end—ultimate goal is good classification.
K-D tree construction doesn't even look at the labels!

Question: Can we use trees to directly build good classifiers?

Decision trees

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 $y \in \mathcal{Y}$.

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:

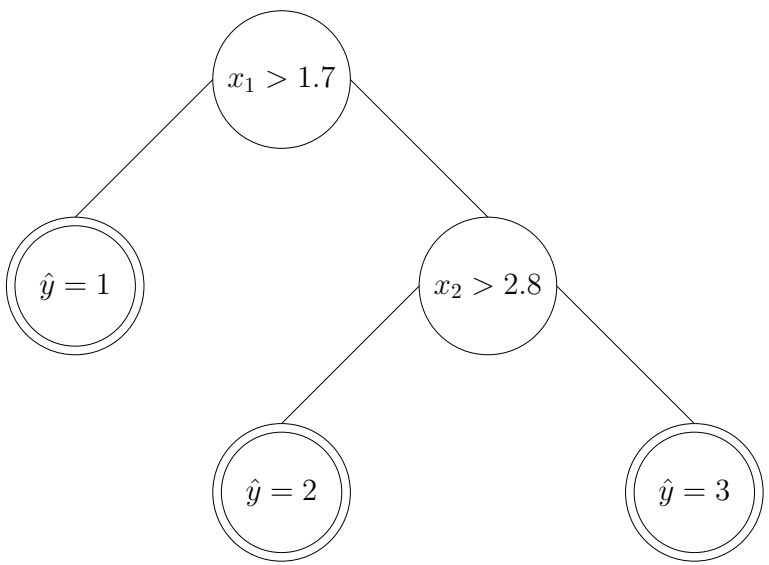
- Each **tree node** is associated with a splitting rule $h : \mathcal{X} \to \{0, 1\}$.
- Each **leaf node** is associated with a label $y \in \mathcal{Y}$.

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

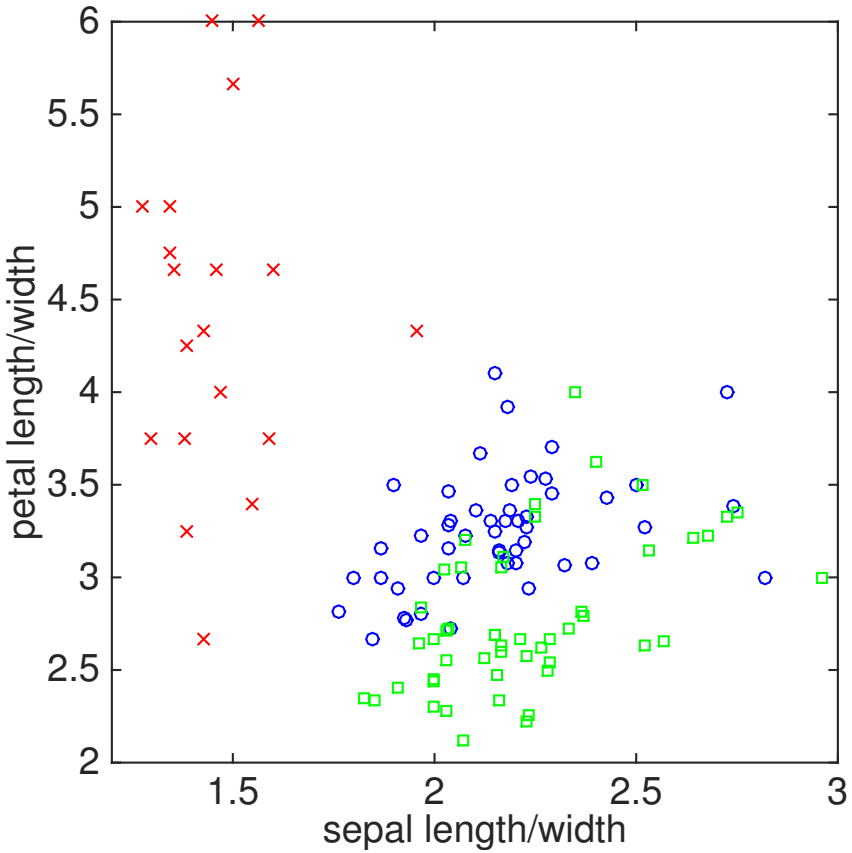
$$h(\boldsymbol{x}) = \mathbb{1}\{x_i > t\}$$

for some $i \in [d]$ and $t \in \mathbb{R}$.
Called *axis-aligned*, *coordinate*, or *Stoller* splits.

(Notation: $[d] := \{1, 2, \ldots, d\}$)



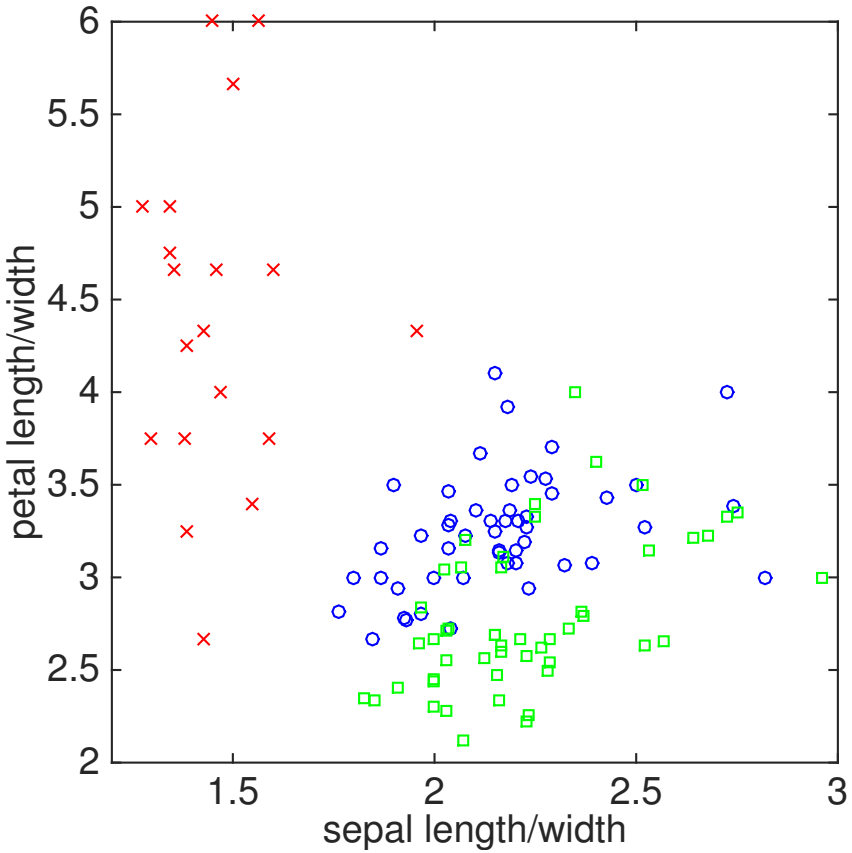
Decision tree example



Classifying irises by sepal and petal measurements

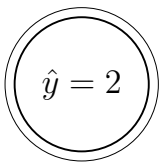
- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

Decision tree example

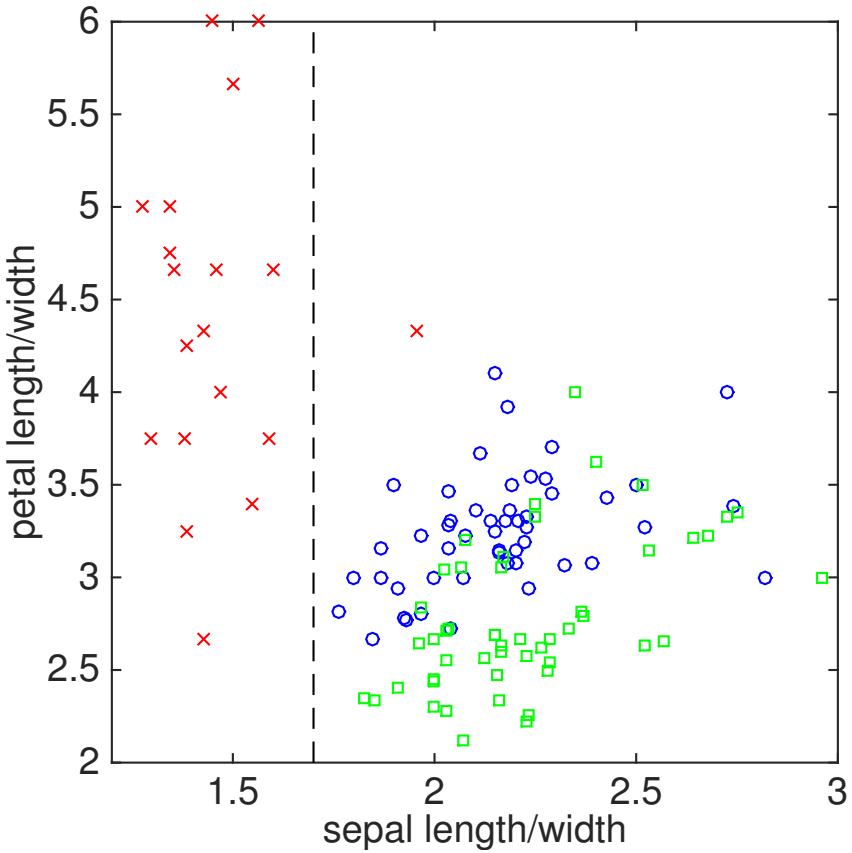


Classifying irises by sepal and petal measurements

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

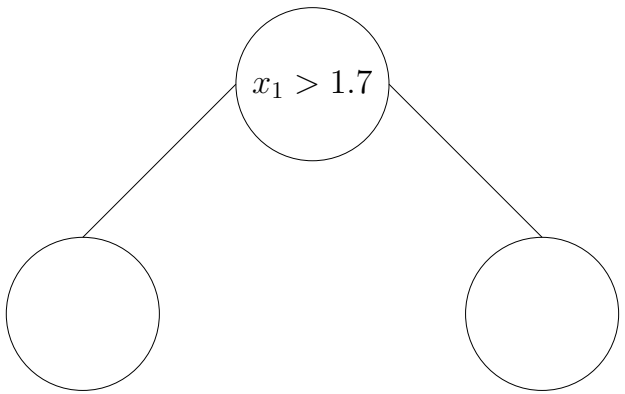


Decision tree example

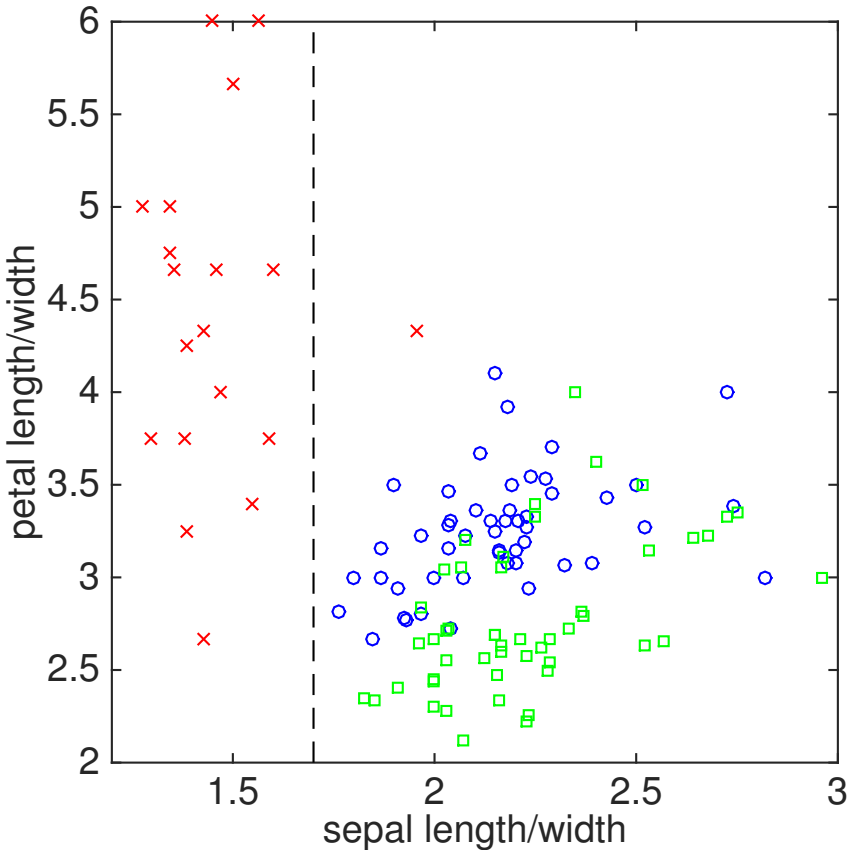


Classifying irises by sepal and petal measurements

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

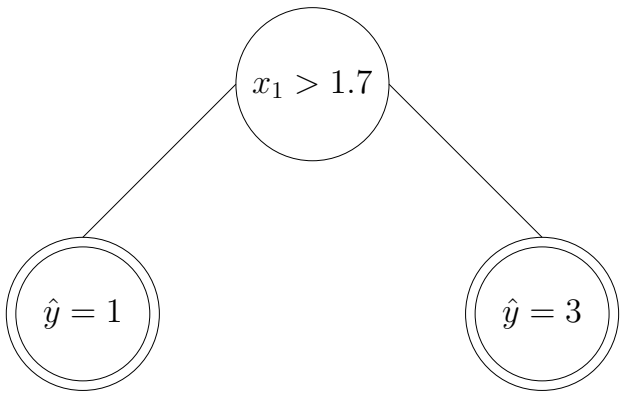


Decision tree example

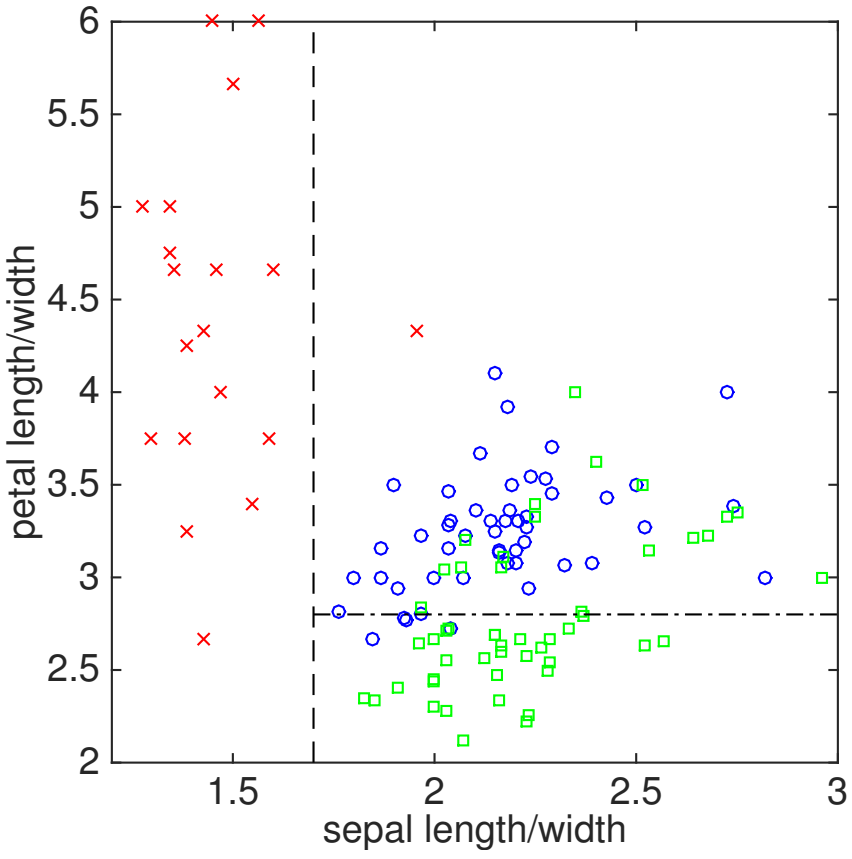


Classifying irises by sepal and petal measurements

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

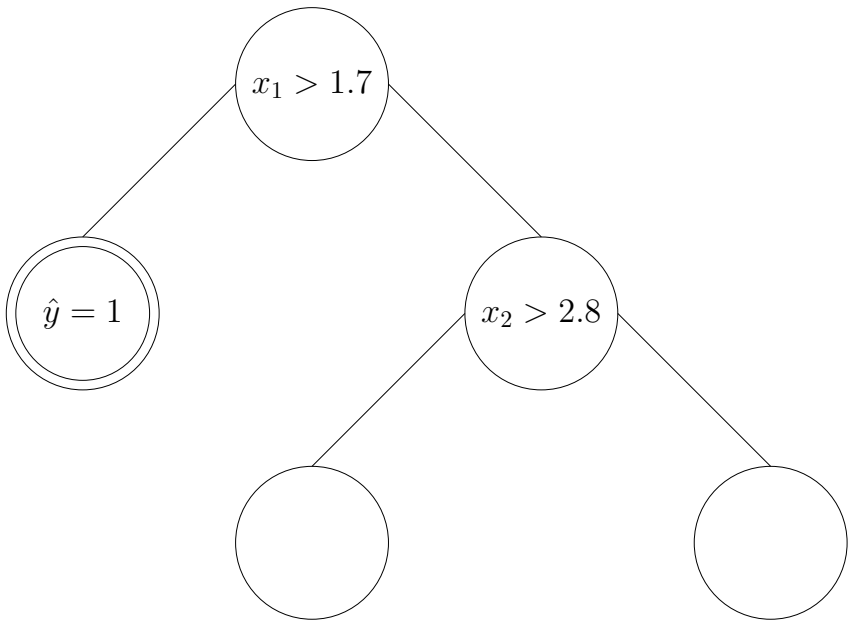


Decision tree example

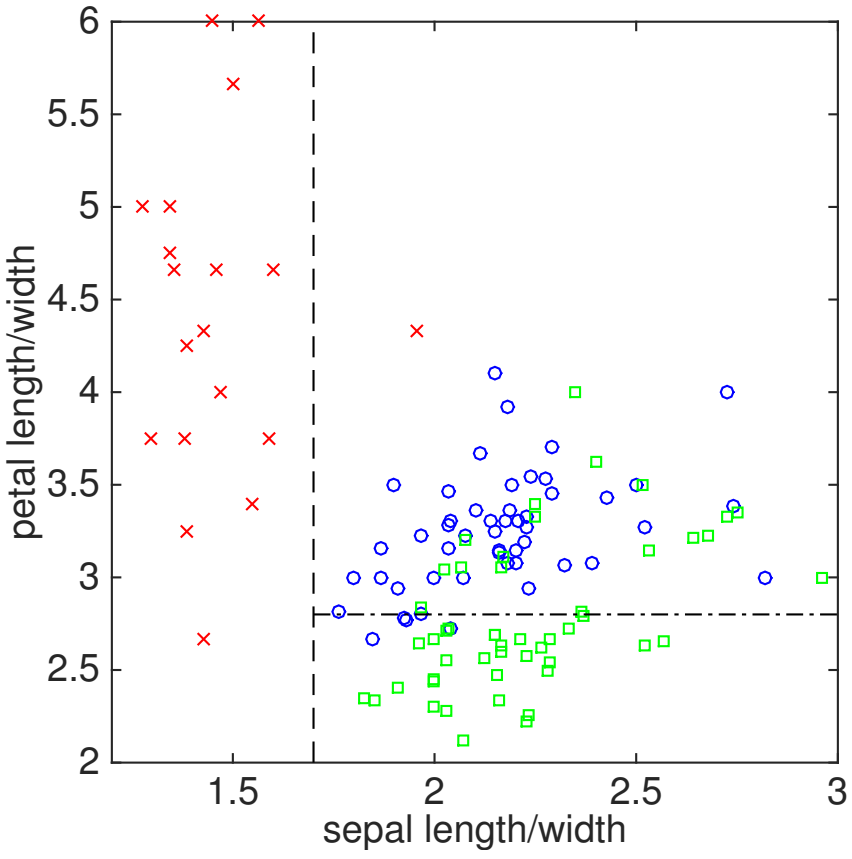


Classifying irises by sepal and petal measurements

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

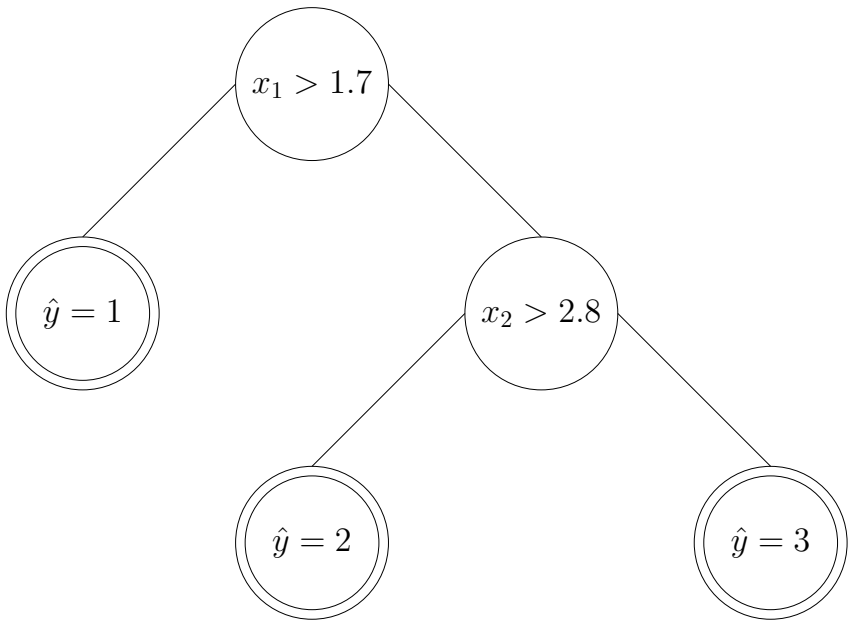


Decision tree example

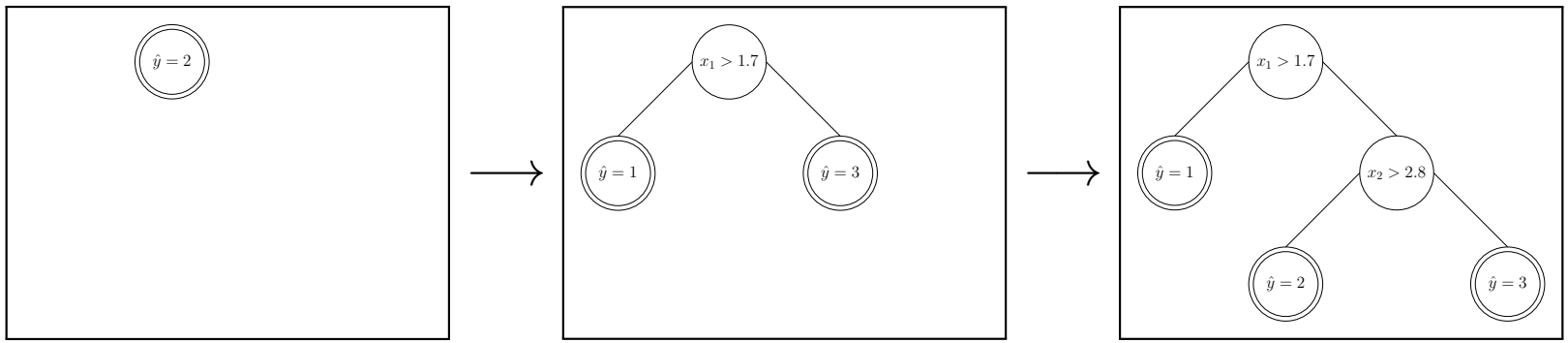


Classifying irises by sepal and petal measurements

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



Basic decision tree learning algorithm



Basic "top-down" greedy algorithm

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

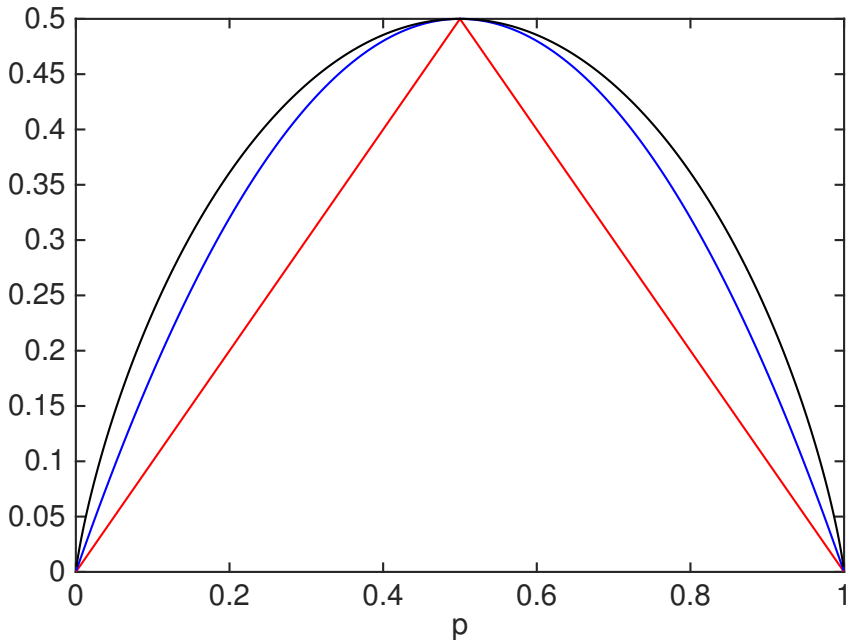
$$u(S) := \min\{p,\, 1 - p\}$$

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

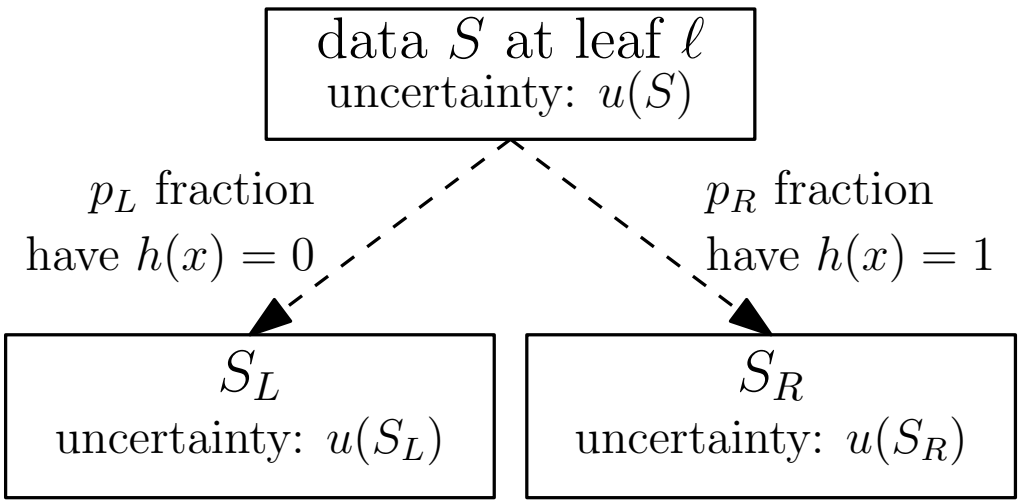
1. **Classification error**:
$$u(S) := 1 - \max_{k \in \mathcal{Y}} p_k$$
2. **Gini index**:
$$u(S) := 1 - \sum_{k \in \mathcal{Y}} p_k^2$$
3. **Entropy**:
$$u(S) := \sum_{k \in \mathcal{Y}} p_k \log \frac{1}{p_k}$$

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

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

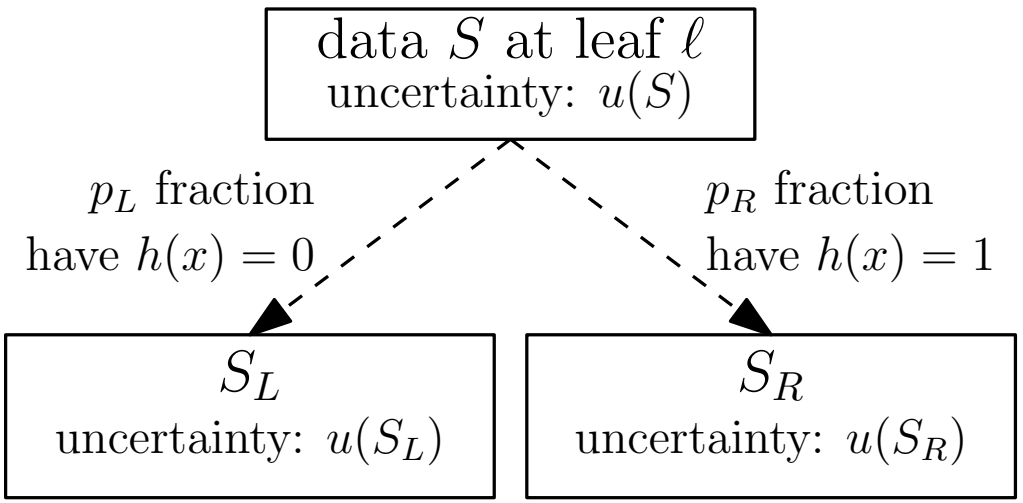
Uncertainty reduction

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



Uncertainty reduction

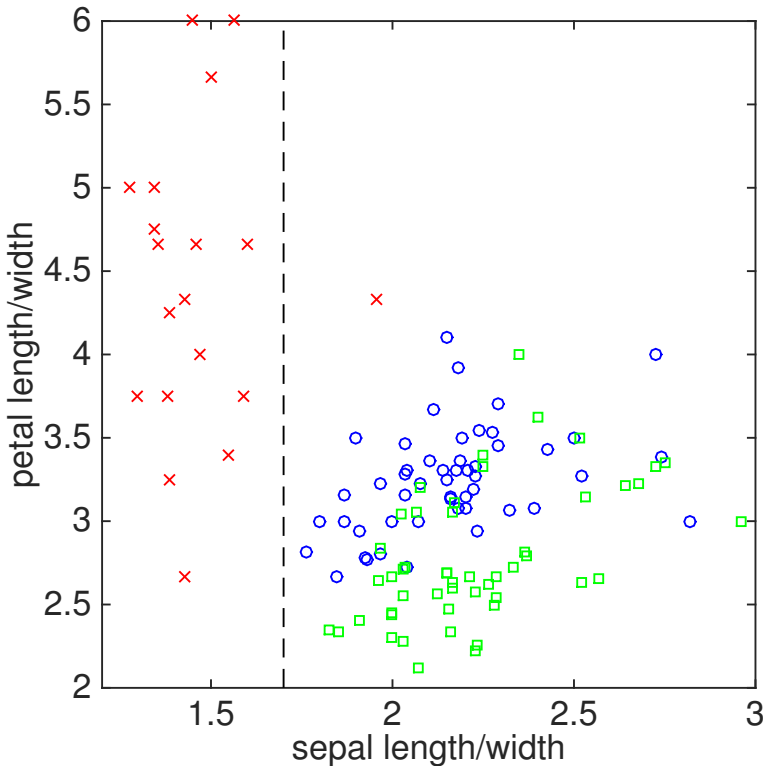
Suppose the data S at a leaf ℓ is split by a rule h into S_L and S_R, where $p_L := |S_L|/|S|$ and $p_R := |S_R|/|S|$.

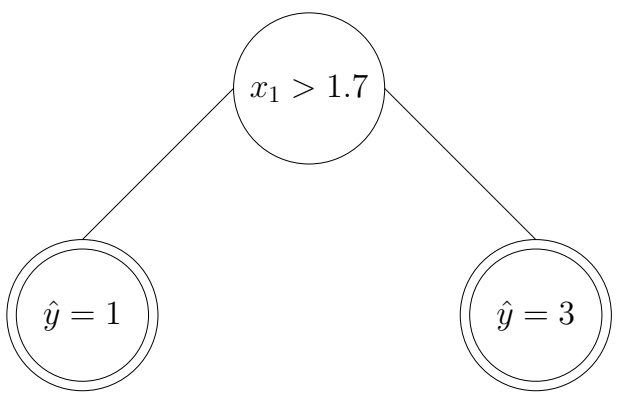


The **reduction in uncertainty** from using rule h at leaf ℓ is

$$u(S) - \Big(p_L \cdot u(S_L) + p_R \cdot u(S_R)\Big).$$

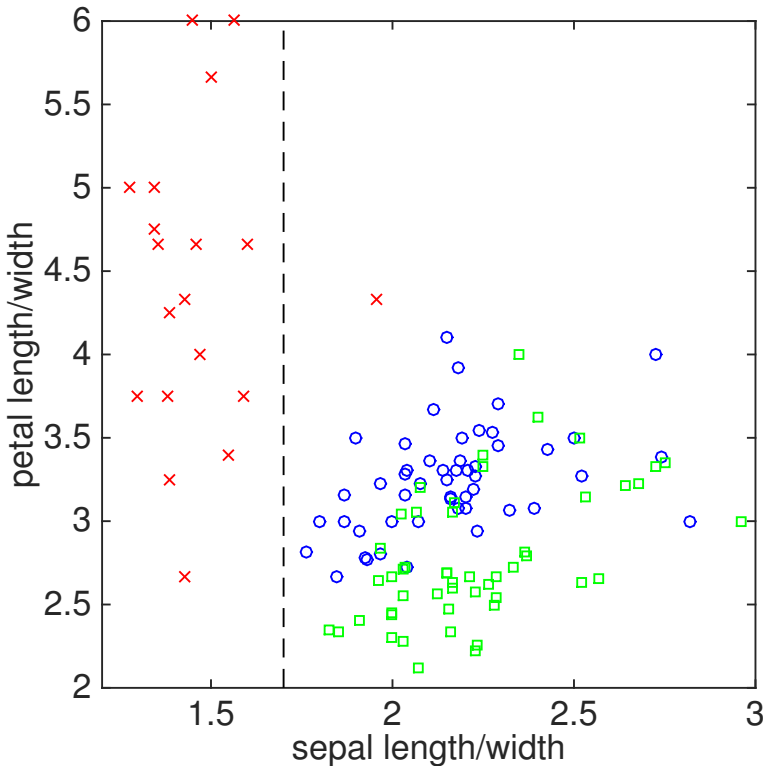
Uncertainty reduction

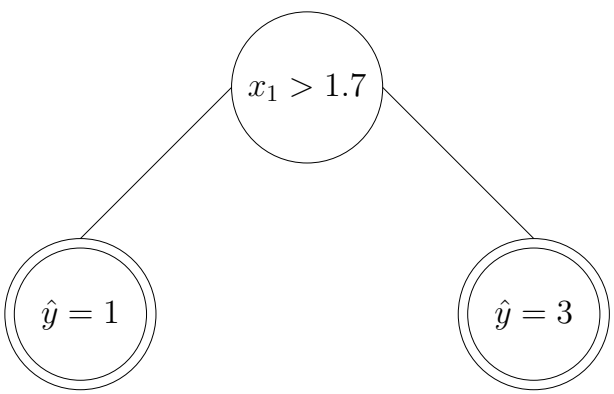




One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

Uncertainty reduction



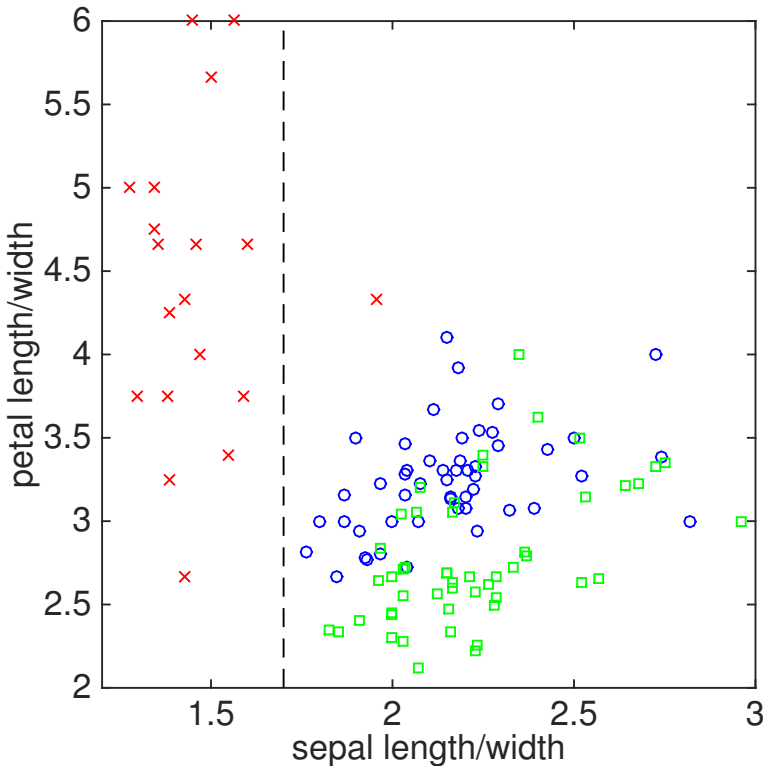


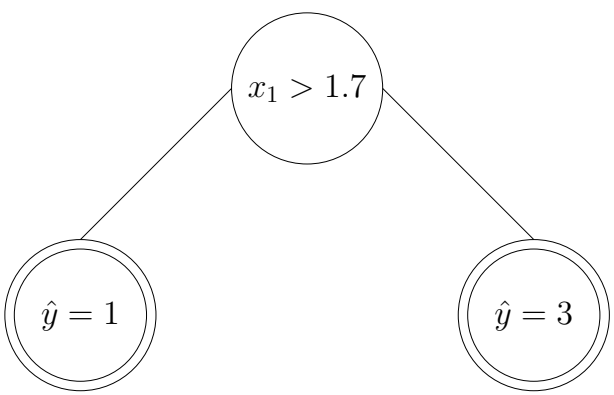
One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

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

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

Uncertainty reduction



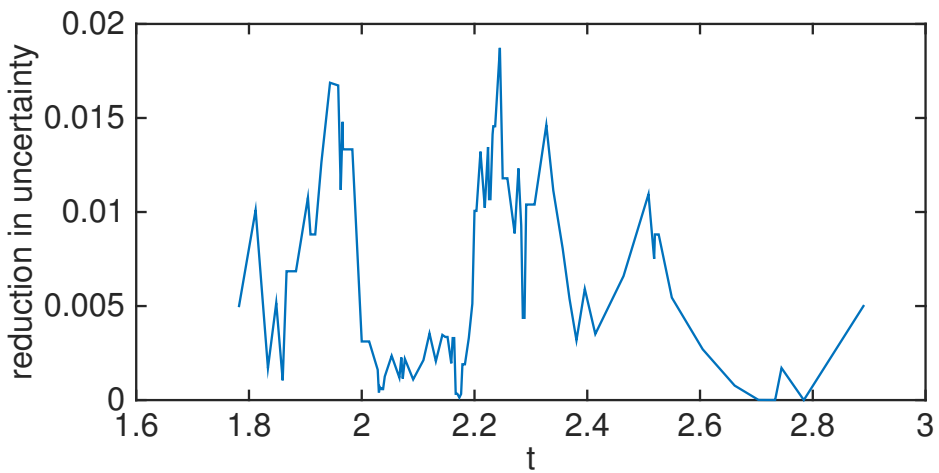


One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

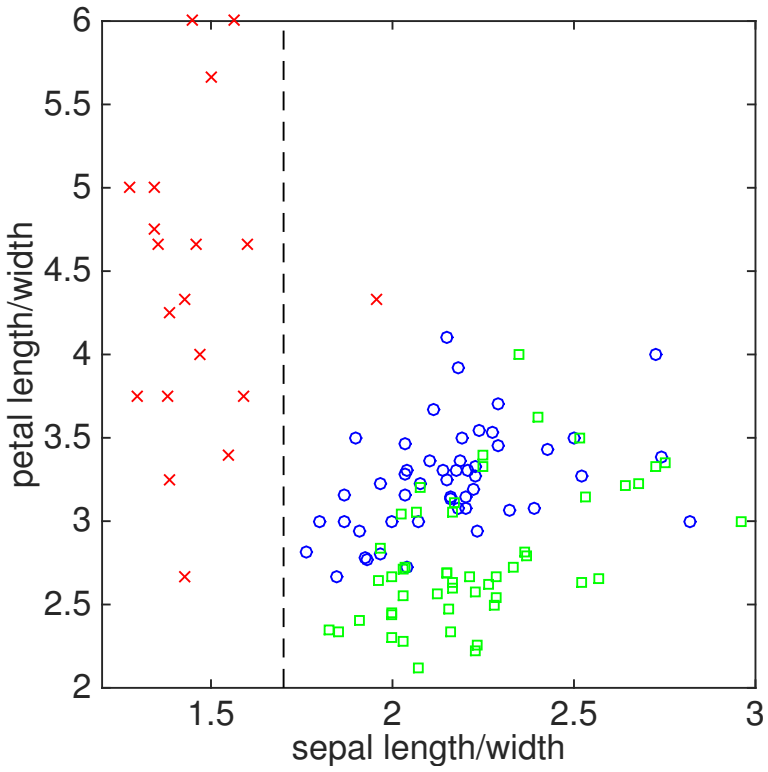
Other leaf (with $\hat{y} = 3$) has Gini index

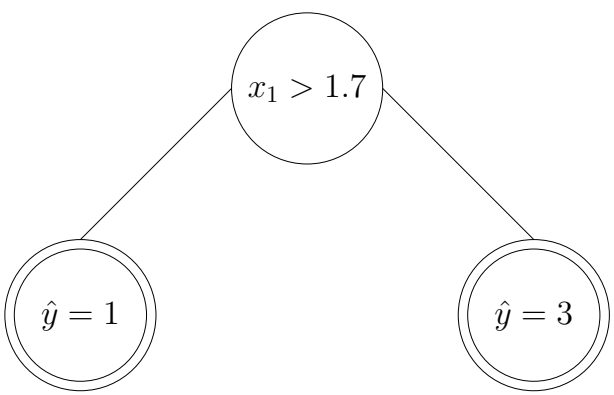
$$u(S) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$
$$= 0.5098.$$

Split S with $\mathbb{1}\{x_1 > t\}$ to S_L, S_R:



Uncertainty reduction



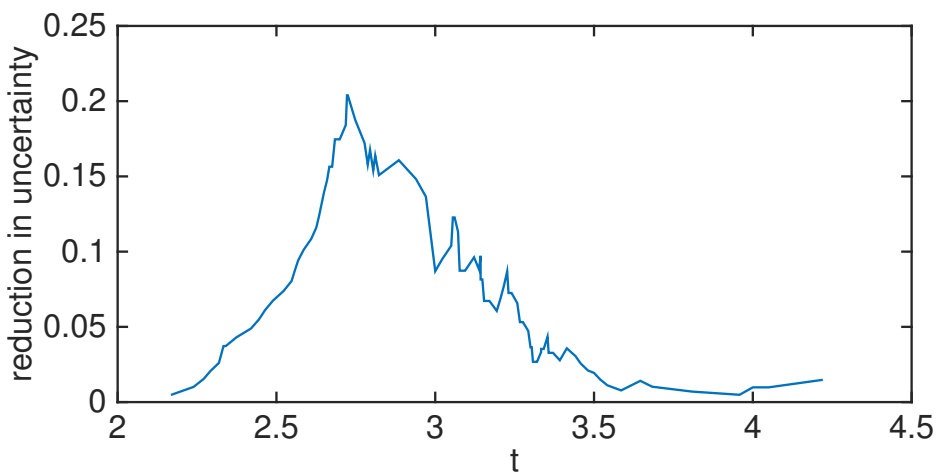


One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

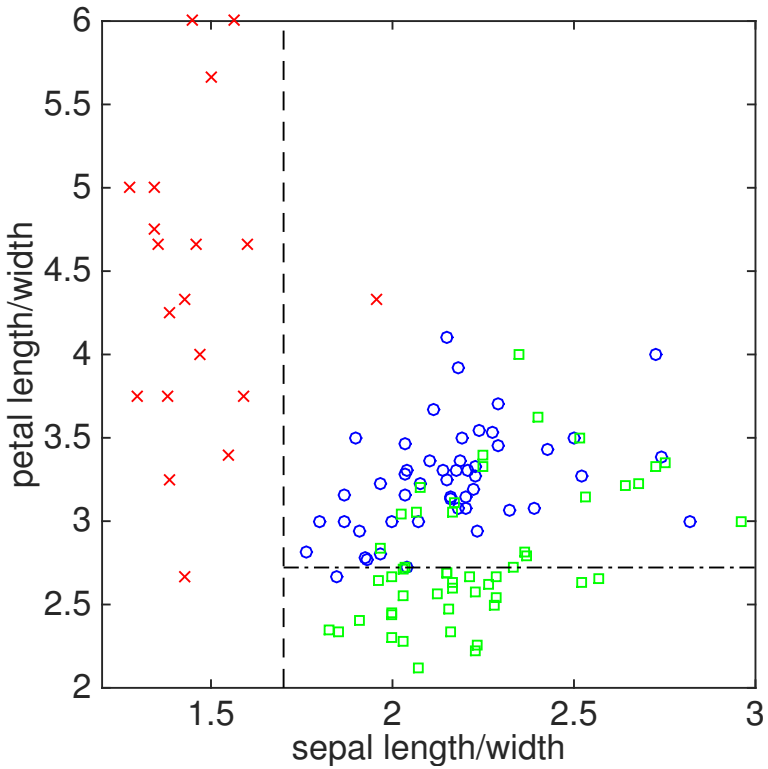
Other leaf (with $\hat{y} = 3$) has Gini index

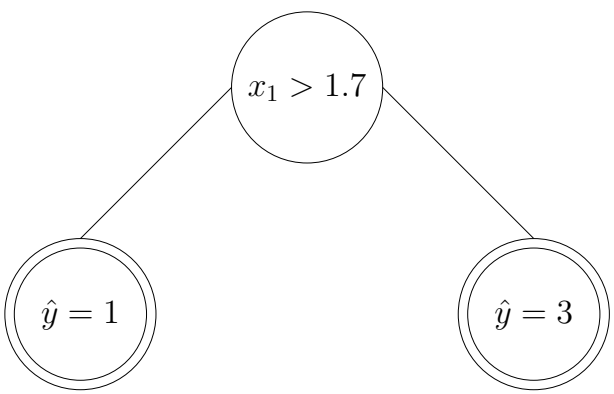
$$u(S) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$
$$= 0.5098.$$

Split S with $\mathbb{1}\{x_2 > t\}$ to S_L, S_R:



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 $\mathbb{1}\{x_2 > 2.7222\}$ 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

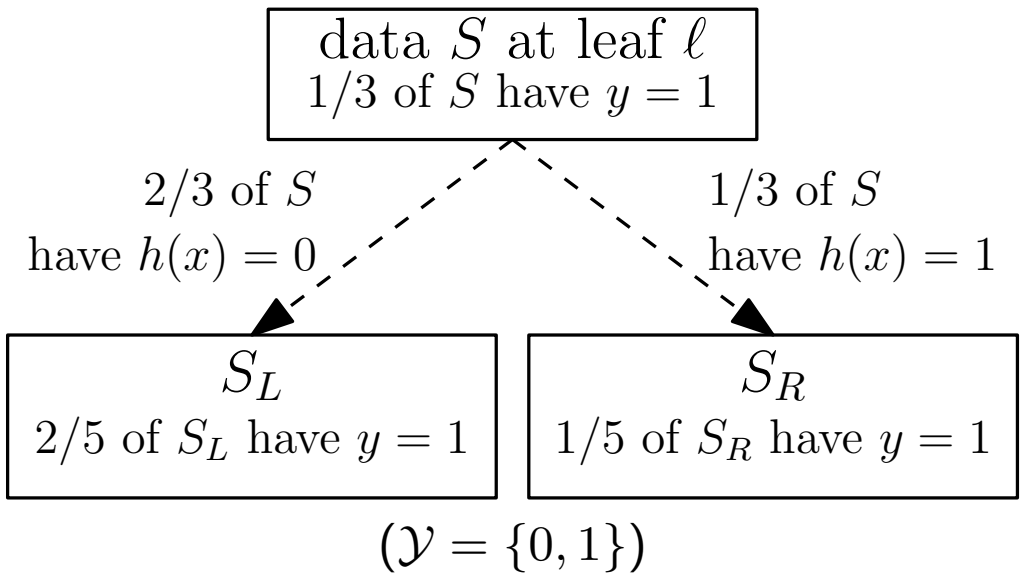
It is possible to have a splitting rule h with

- 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

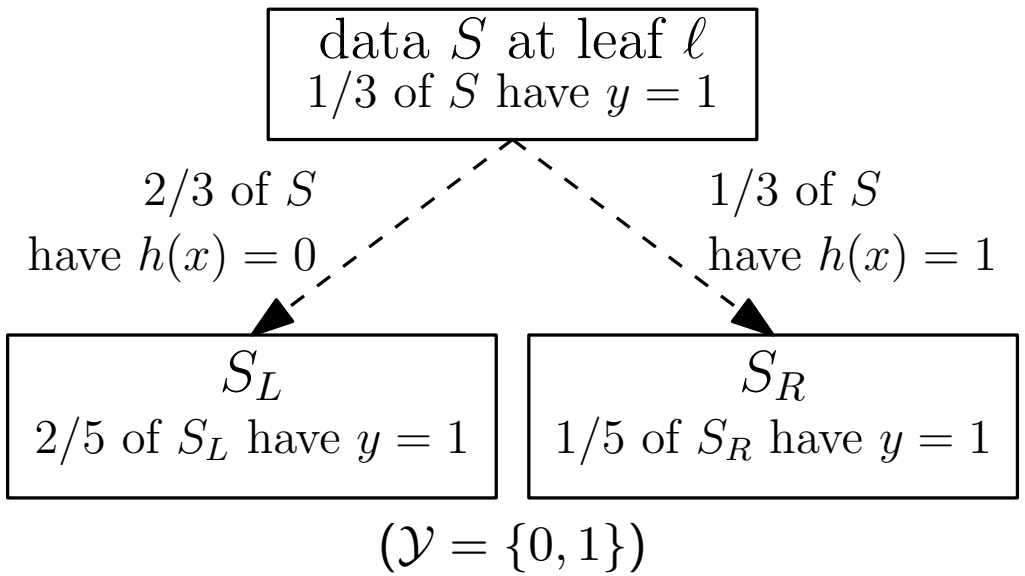
- zero reduction in classification error, but
- non-zero reduction in Gini index or entropy.



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.

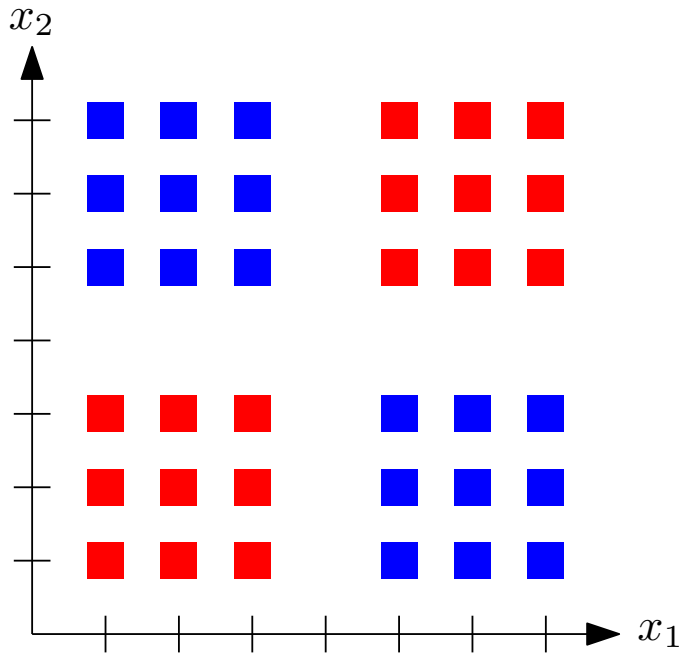


Reduction in classification error: $\frac{1}{3} - \left(\frac{2}{3} \cdot \frac{2}{5} + \frac{1}{3} \cdot \frac{1}{5}\right) = 0$

Reduction in Gini index: $\frac{4}{9} - \left(\frac{2}{3} \cdot \frac{12}{25} + \frac{1}{3} \cdot \frac{8}{25}\right) = \frac{4}{225}$

Limitations of uncertainty notions

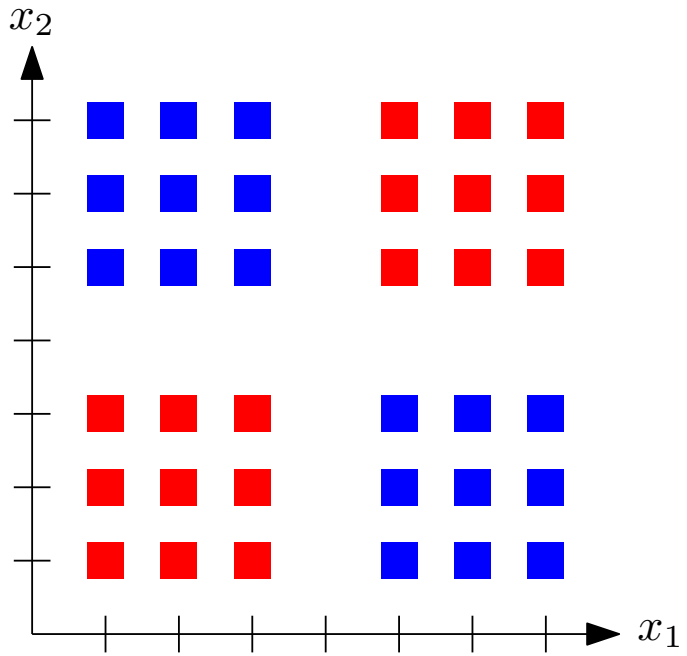
Suppose $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{\text{red}, \text{blue}\}$, and the data is as follows:



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

Limitations of uncertainty notions

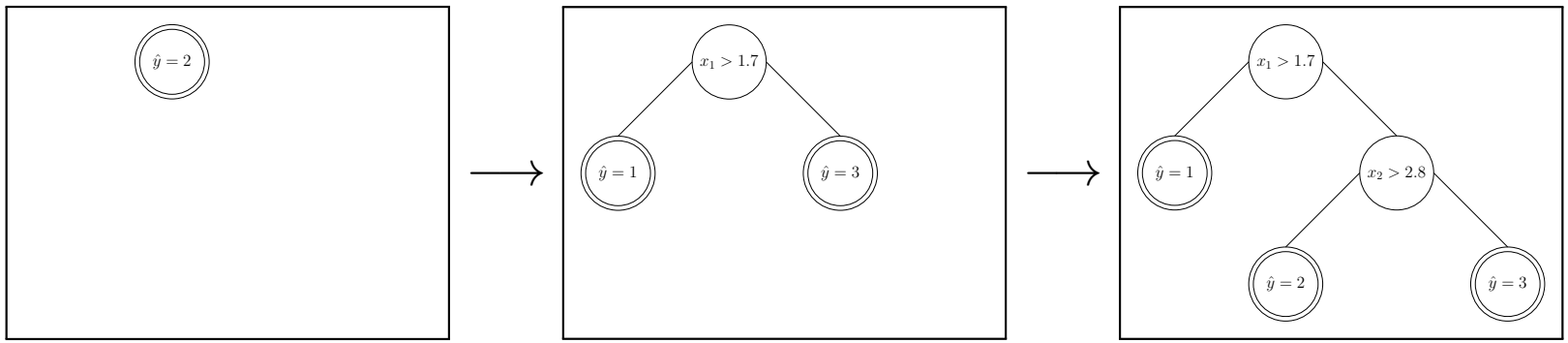
Suppose $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{\text{red}, \text{blue}\}$, and the data is as follows:



Every split of the form $\mathbb{1}\{x_i > t\}$ 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



Basic "top-down" greedy algorithm

- Initially, tree is a single leaf node containing all (training) data.
- Loop:
  - Pick the leaf ℓ and rule h that maximally reduces uncertainty.
  - Split data in ℓ 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.]

Stopping criterion

Many alternatives; two common choices are:

Stopping criterion

Many alternatives; two common choices are:

1. Stop when the **tree reaches a pre-specified size**.

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.

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.)

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.

Stopping criterion

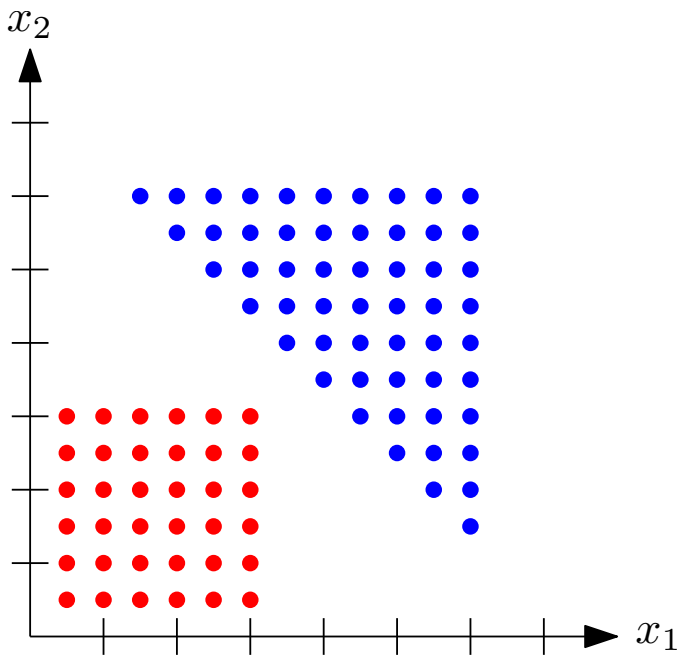
Many alternatives; two common choices are:

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.



Stopping criterion

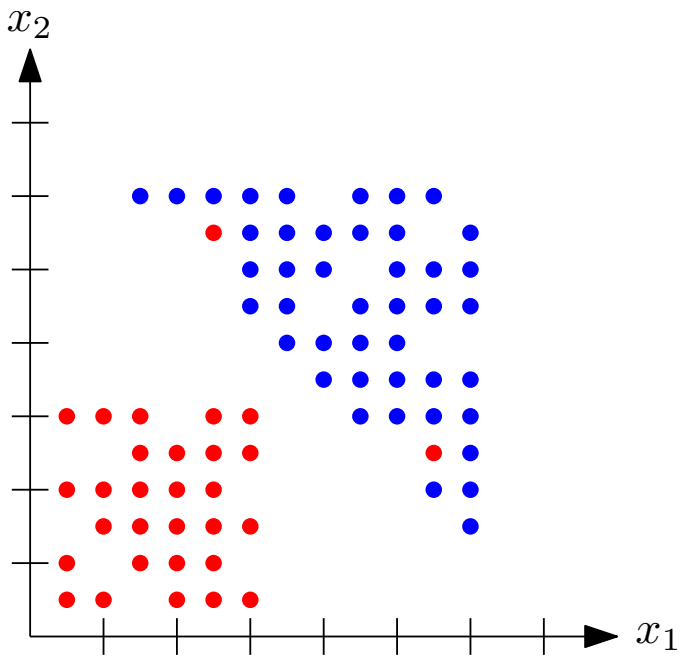
Many alternatives; two common choices are:

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.



Stopping criterion

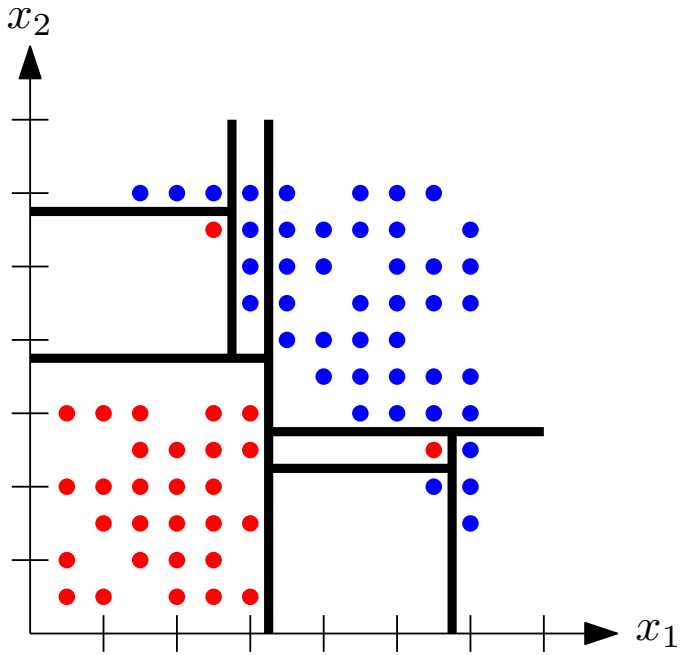
Many alternatives; two common choices are:

1. Stop when the **tree reaches a pre-specified size**.

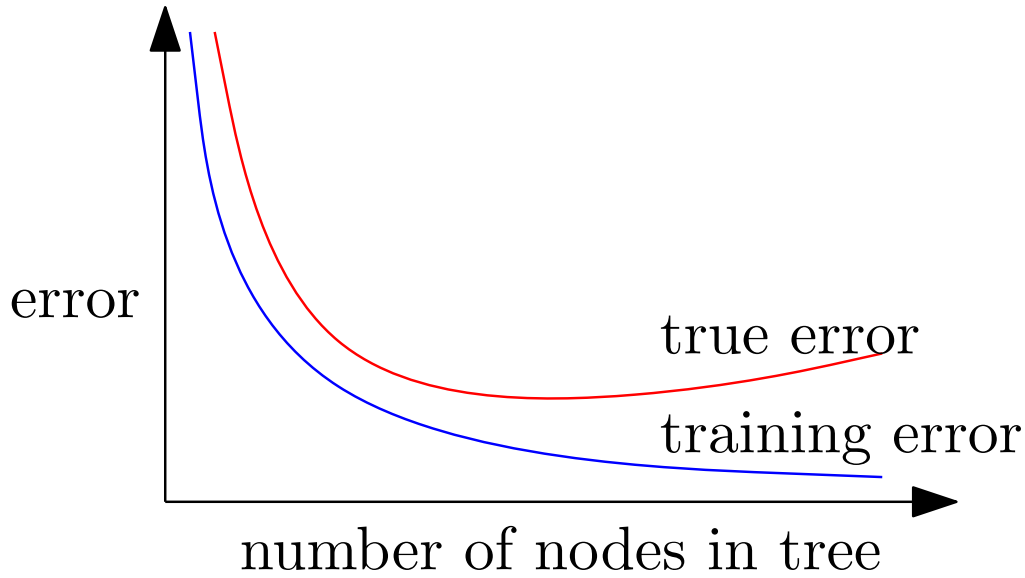
   Involves setting additional "tuning parameters"—use hold-out or cross-validation.

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

   Serious danger of **overfitting** spurious structure due to sampling.



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**.

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.

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).

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.
- $\mathcal{Y} = \{\text{spam}, \text{not spam}\}$
- E-mails represented by 57 features:
  - 48: percentange 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%.

	$\hat{y} = \text{not spam}$	$\hat{y} = \text{spam}$
$y = \text{not spam}$	57.3%	4.0%
$y = \text{spam}$	5.3%	33.4%

Example: Spam filtering