COMS 4771 Lecture 4

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 $\mathcal{X} = \mathbb{R}^d$).

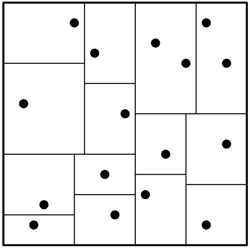
Construction procedure

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

1. Pick a coordinate $j \in \{1, 2, \ldots, d\}$.
2. Let m be the median of $\{x_j : \boldsymbol{x} \in S\}$.
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 fast NN search in $\mathbb{R}^d$

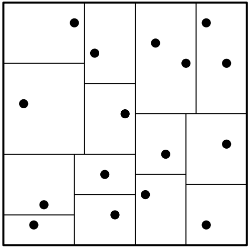
K-D tree: data structure for supporting NN search in $\mathbb{R}^d$ (and hence for implementing NN classifiers where $\mathcal{X} = \mathbb{R}^d$).

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 $\{x_j : \boldsymbol{x} \in S\}$.
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.

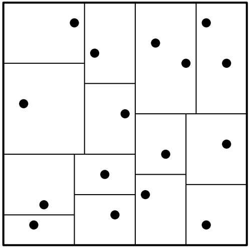


Searching a K-D tree for a **NN**: requires some *backtracking*, still fast when d is small, but can be slow when d is large.

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 $\{x_j : \boldsymbol{x} \in S\}$.
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.

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.

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 $\mathcal{X} = \mathbb{R}^d$).

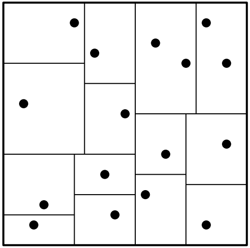
Construction procedure

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

1. Pick a coordinate $j \in \{1, 2, \ldots, d\}$.
2. Let m be the median of $\{x_j : \boldsymbol{x} \in S\}$.
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.



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 $f\colon \mathcal{X} \to \mathcal{Y}$, represented by a binary tree in which:

- Each **tree node** is associated with a splitting rule $h\colon \mathcal{X} \to \{0, 1\}$.
- 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 $f\colon \mathcal{X} \to \mathcal{Y}$, represented by a binary tree in which:

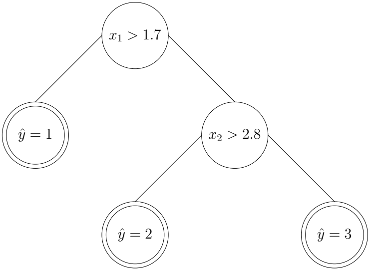
- Each **tree node** is associated with a splitting rule $h\colon \mathcal{X} \to \{0,1\}$.
- Each **leaf node** is associated with a label $y \in \mathcal{Y}$.

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

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

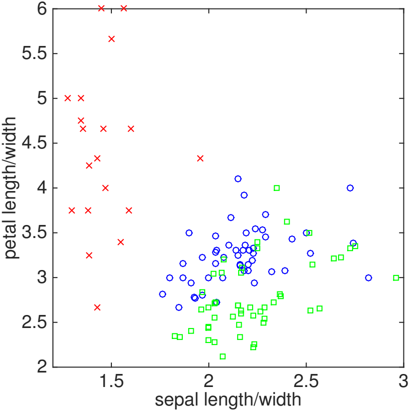
for some $i \in [d]$ and $t \in \mathbb{R}$.
Called *axis-aligned*, *coordinate*, or *Stoller* splits.

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



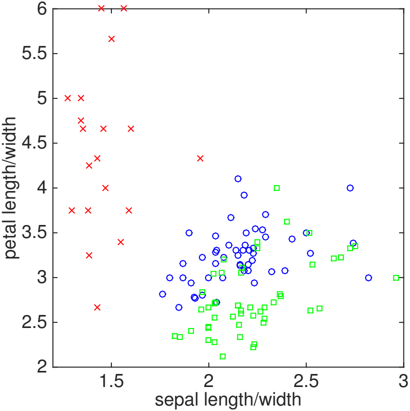
base on single dimension

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

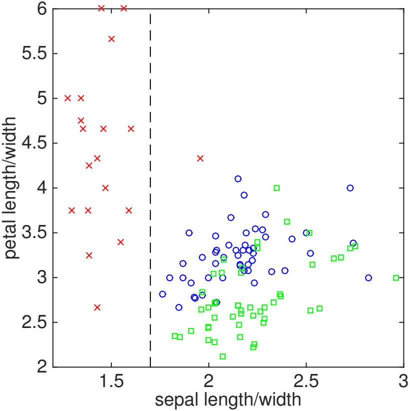


Classifying irises by sepal and petal measurements

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

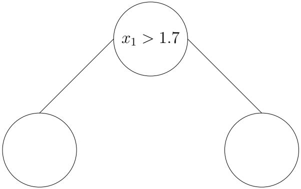
$\hat{y} = 2$

Decision tree example

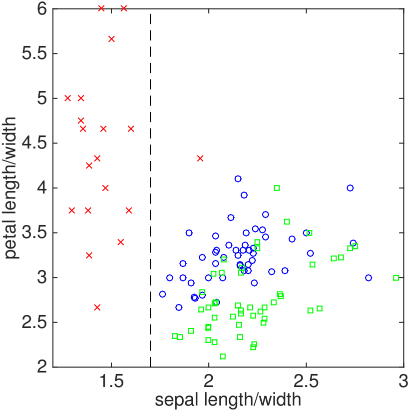


Classifying irises by sepal and petal measurements

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

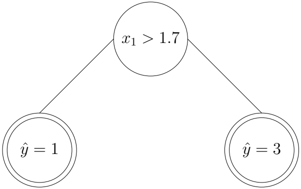


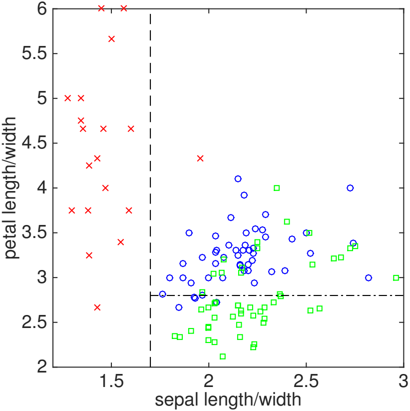
Decision tree example



Classifying irises by sepal and petal measurements

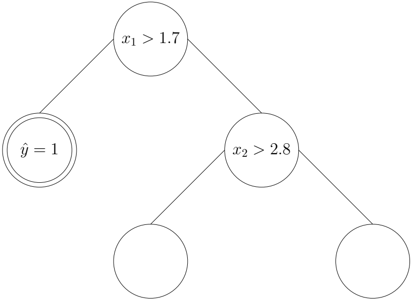
- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- x_1 = ratio of sepal length to width
- x_2 = ratio of petal length to width



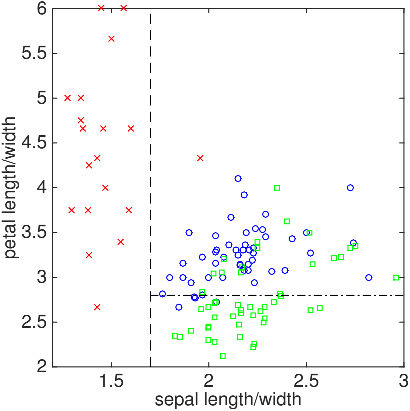


Classifying irises by sepal and petal measurements

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

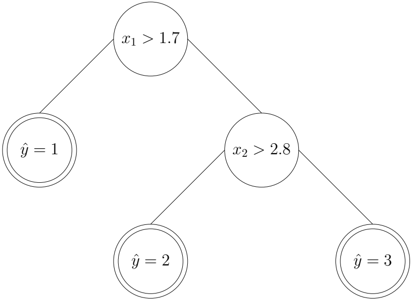


Decision tree example

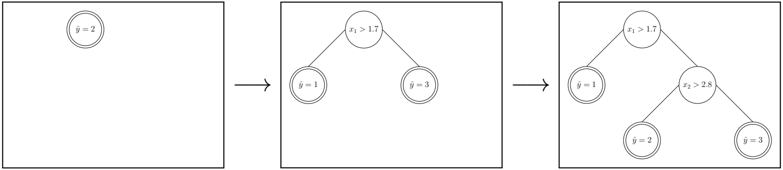


Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- 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: how to measure uncertainty.
  - 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**:

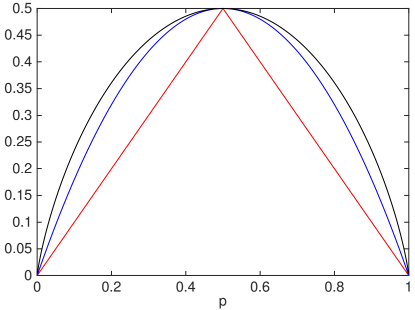
$$u(S) := \min\{p,\, 1 - p\}$$

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 $k \in \mathcal{Y}$).

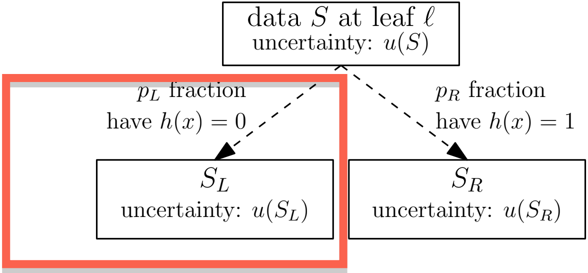
1. **Classification error**:
$$u(S) := 1 - \max_{k \in \mathcal{Y}} p_k$$
2. **Gini index**:
$$u(S) := 1 - \sum_{k \in \mathcal{Y}} p_k^2$$
3. **Entropy**:
$$u(S) := \sum_{k \in \mathcal{Y}} p_k \log \frac{1}{p_k}$$

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

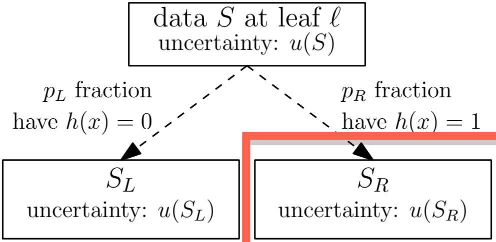
Suppose the data S at a leaf ℓ is split by a rule h into S_L and S_R, where $p_L := |S_L|/|S|$ and $p_R := |S_R|/|S|$.



should try to minimize the uncertainty. thus
push the same class into the same side

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 $p_R := |S_R|/|S|$.

note: how to compute a node's uncertaintity

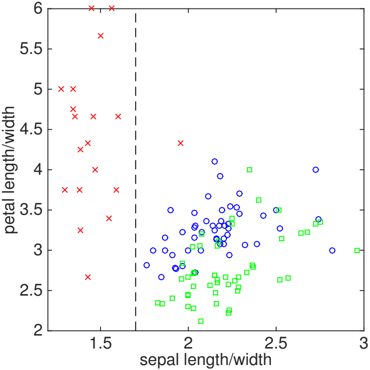


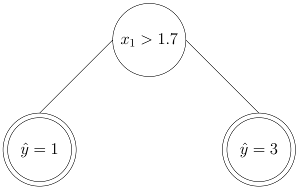
This part's uncertainty!

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

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

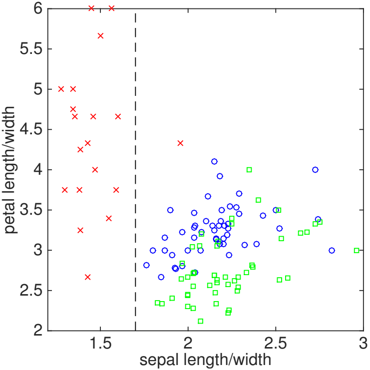
Uncertainty reduction

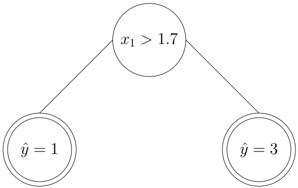




One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

Uncertainty reduction



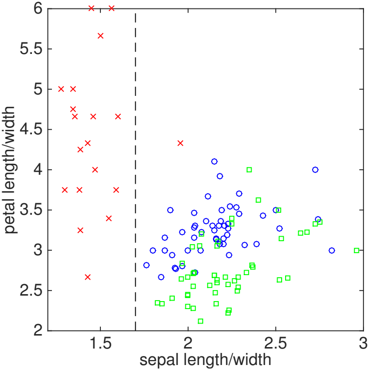


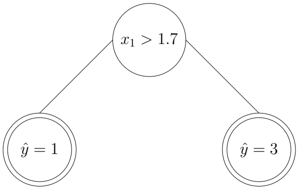
One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

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



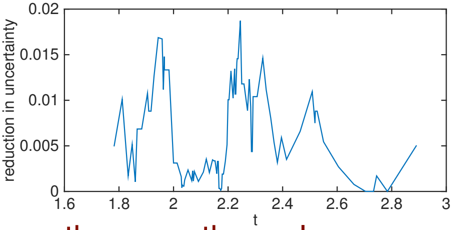


One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

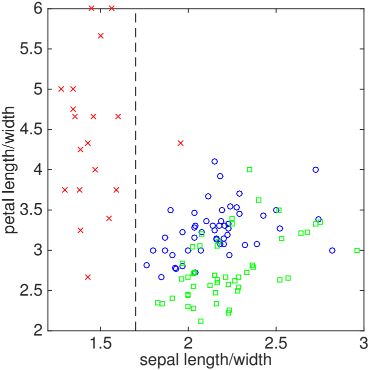
Other leaf (with $\hat{y} = 3$) has Gini index

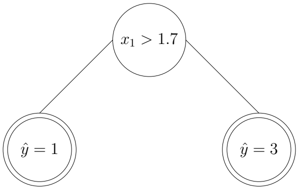
$$u(S) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$
$$= 0.5098.$$

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



cause there are three classes, we can't only pick the pick one



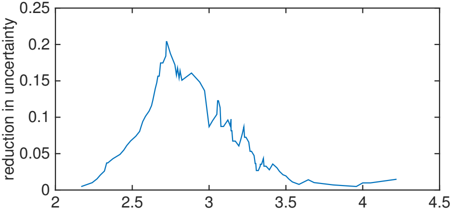


One leaf (with $\hat{y} = 1$) already has zero uncertainty (a pure leaf).

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

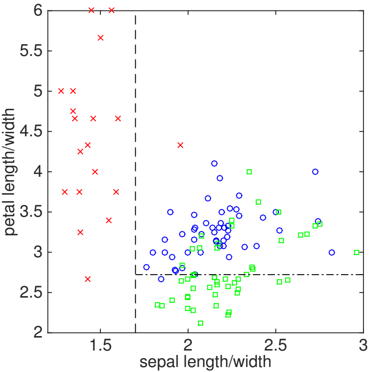
$$u(S) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$
$$= 0.5098.$$

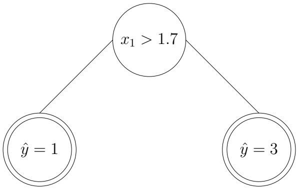
Split S with $\mathbb{1}\{x_2 > t\}$ to S_L, S_R:



only two classes left, pick the peak one!

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

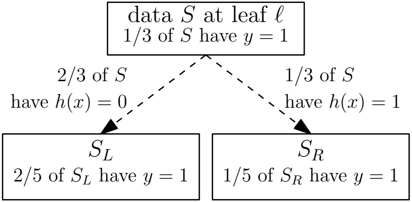
It is possible to have a splitting rule h with

- 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.

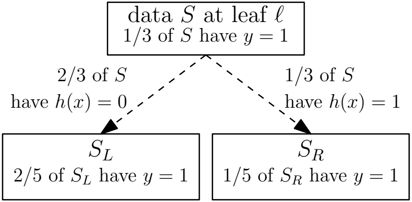


$(\mathcal{Y} = \{0, 1\})$

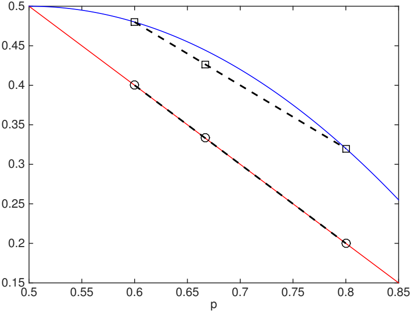
Comparing notions of uncertainty

It is possible to have a splitting rule h with

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



$(\mathcal{Y} = \{0, 1\})$



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

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

Aside: reduction in entropy

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

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

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 $Z|W = w$.)

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 $\mathcal{Z}$-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, $u(S) = H(Y)$.

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

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

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

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

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

(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, $u(S) = H(Y)$.

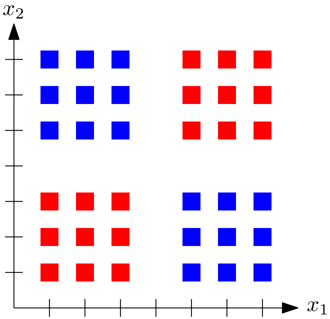
Reduction in uncertainty after a split $h \colon \mathcal{X} \to \{0, 1\}$ is

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

This is called **mutual information** between Y and $h(X)$.
More on information theory later in the course.

Limitations of uncertainty notions

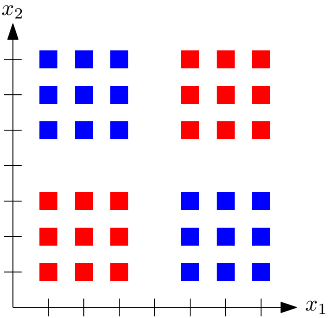
Suppose $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{\text{red}, \text{blue}\}$, and the data is as follows:



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

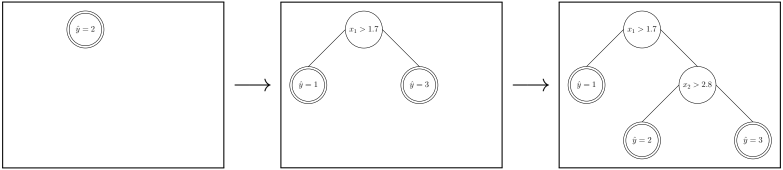
Suppose $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{\text{red}, \text{blue}\}$, and the data is as follows:



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

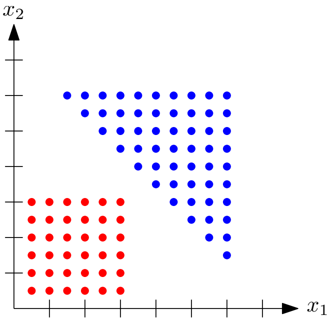
Many alternatives; two common choices are:

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

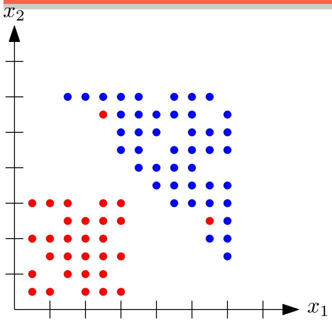
Many alternatives; two common choices are:

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

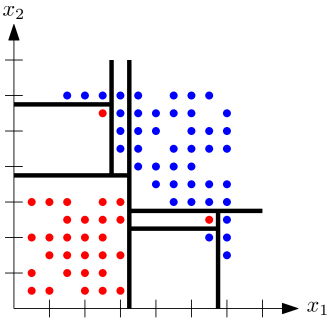
Many alternatives; two common choices are:

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

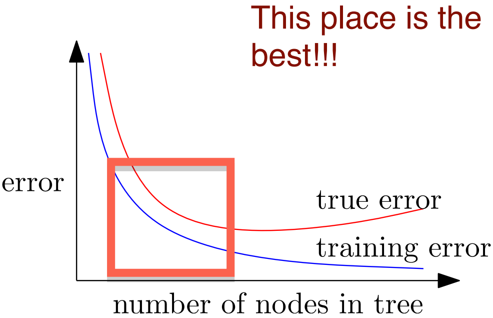
   Involves setting additional "tuning parameters"—use hold-out or cross-validation.

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**.

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**.

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.
- $\mathcal{Y} = \{\text{spam}, \text{not spam}\}$
- 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).

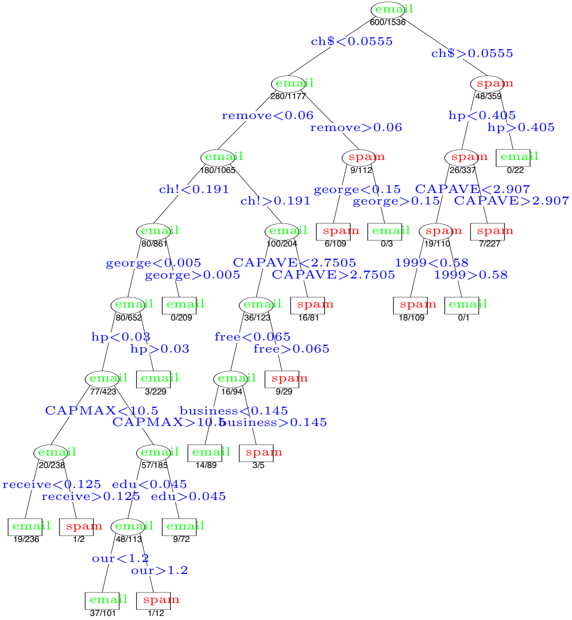
Results

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

Chosen tree has just **17 leaves**. Test error is 4.51%.

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

Example: Spam filtering



Recap and final notes

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

Recap and final notes

- Decision trees are very flexible classifiers (like NN).
  - Certain greedy strategies for training decision trees are **consistent**: $\mathrm{err}(\hat{f}) \to \mathrm{err}(f^\star)$ 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).

Recap and final notes

- Decision trees are very flexible classifiers (like NN).
  - Certain greedy strategies for training decision trees are **consistent**: $\mathrm{err}(\hat{f}) \to \mathrm{err}(f^\star)$ 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.

Recap and final notes

- Decision trees are very flexible classifiers (like NN).
  - Certain greedy strategies for training decision trees are **consistent**: $\text{err}(\hat{f}) \to \text{err}(f^\star)$ 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?