CS 760: Machine Learning
Supervised Learning I

Josiah Hanna

University of Wisconsin — Madison

9/14/2023

Announcements

Announcements

- Enrollment:
  - Email me today if you’re still on waitlist AND have a reason for additional priority.
  - It will be offered next semester if you don’t get in.

Announcements

- Enrollment:
  - Email me today if you’re still on waitlist AND have a reason for additional priority.
  - It will be offered next semester if you don’t get in.
- Background Knowledge:
  - Please look at homework 1 before add/drop deadline.

Announcements

- Enrollment:
  - Email me today if you’re still on waitlist AND have a reason for additional priority.
  - It will be offered next semester if you don’t get in.
- Background Knowledge:
  - Please look at homework 1 before add/drop deadline.
  - Please take background survey on Piazza.
- Homework 1 is due at 9:30 AM on Tuesday, September 19.
- Sign-up for Piazza (link on webpage)
  - Passcode: mlfall23

Today's Learning Outcomes

Today’s Learning Outcomes

- **After today’s lecture:**

Today’s Learning Outcomes

- **After today’s lecture:**
  - You will be able to explain how the k-nearest neighbor’s algorithm classifies unseen instances.

Today’s Learning Outcomes

- **After today’s lecture:**
  - You will be able to explain how the k-nearest neighbor’s algorithm classifies unseen instances.
  - You will be able to explain the concept of an inductive bias.

Today’s Learning Outcomes

- **After today’s lecture:**
  - You will be able to explain how the k-nearest neighbor’s algorithm classifies unseen instances.
  - You will be able to explain the concept of an inductive bias.
  - You will be able to explain how a decision tree classifies instances.

Outline

- **Review from last time**
  - Features, labels, hypothesis class, training, generalization
- **Instance-based learning**
  - k-NN classification/regression, locally weighted regression, strengths & weaknesses, inductive bias
- **Decision trees**
  - Setup, splits, learning, information gain, strengths and weaknesses

Outline

- **Review from last time**
  - Features, labels, hypothesis class, training, generalization
- **Instance-based learning**
  - k-NN classification/regression, locally weighted regression, strengths & weaknesses, inductive bias
- **Decision trees**
  - Setup, splits, learning, information gain, strengths and weaknesses

Supervised Learning: Formal Setup

Problem setting

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function*

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \to \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*):

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$

Given

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$

Given

- Training set of instances for unknown target function,

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$

Given

- Training set of instances for unknown target function, where

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$

Given

- Training set of instances for unknown target function, where $y^{(i)} \approx f(x^{(i)})$

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \to \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \to \mathcal{Y}\}$

Given

- Training set of instances for unknown target function, where $y^{(i)} \approx f(x^{(i)})$

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$$

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$

Given

- Training set of instances for unknown target function, where $y^{(i)} \approx f(x^{(i)})$

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$$

safe

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$

Given

- Training set of instances for unknown target function, where $y^{(i)} \approx f(x^{(i)})$

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$$

safe

poisonous

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$

Given

- Training set of instances for unknown target function, where $y^{(i)} \approx f(x^{(i)})$

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$$


safe

poisonous

safe

Supervised Learning: Objects

Three types of sets

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$
  - Output space:

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$

  - Output space:
    - **Discrete/Nominal**

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$

  - Output space:
    - **Discrete/Nominal** $\mathcal{Y} = \{-1, +1\}$

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$
  - Output space:
    - **Discrete/Nominal** $\mathcal{Y} = \{-1, +1\}$

safe poisonous

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$

  - Output space:
    - **Discrete/Nominal** $\mathcal{Y} = \{-1, +1\}$

safe poisonous

    - **Continuous**

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$

  - Output space:
    - **Discrete/Nominal** $\mathcal{Y} = \{-1, +1\}$ safe poisonous
    - **Continuous** $\mathcal{Y} \subseteq \mathbb{R}$

Supervised Learning: Objects

Three types of sets

- Input space, output space, hypothesis class

$$\mathcal{X}, \mathcal{Y}, \mathcal{H}$$

- **Examples**:
  - Input space: feature vectors $\mathcal{X} \subseteq \mathbb{R}^d$

  - Output space:
    - **Discrete/Nominal** $\mathcal{Y} = \{-1, +1\}$ safe poisonous
    - **Continuous** $\mathcal{Y} \subseteq \mathbb{R}$ 13.23°

Output space: Classification vs. Regression

Depending on the choice of $\mathcal{Y}$, we have special names:

Output space: Classification vs. Regression

Depending on the choice of $\mathcal{Y}$, we have special names:

- Discrete: "**classification**". The elements of $\mathcal{Y}$ are **classes**

Output space: Classification vs. Regression

Depending on the choice of $\mathcal{Y}$, we have special names:

- Discrete: "**classification**". The elements of $\mathcal{Y}$ are **classes**

Output space: Classification vs. Regression

Depending on the choice of $\mathcal{Y}$, we have special names:

- Discrete: "**classification**". The elements of $\mathcal{Y}$ are **classes**

- Continuous: "**regression**"

Output space: Classification vs. Regression

Depending on the choice of $\mathcal{Y}$, we have special names:

- Discrete: "**classification**". The elements of $\mathcal{Y}$ are **classes**

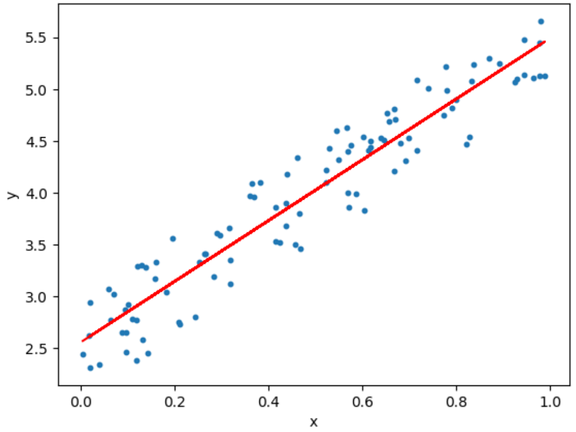
- Continuous: "**regression**"
  - Example: linear regression

Output space: Classification vs. Regression

Depending on the choice of $\mathcal{Y}$, we have special names:

- Discrete: "**classification**". The elements of $\mathcal{Y}$ are **classes**

- Continuous: "**regression**"
  - Example: linear regression

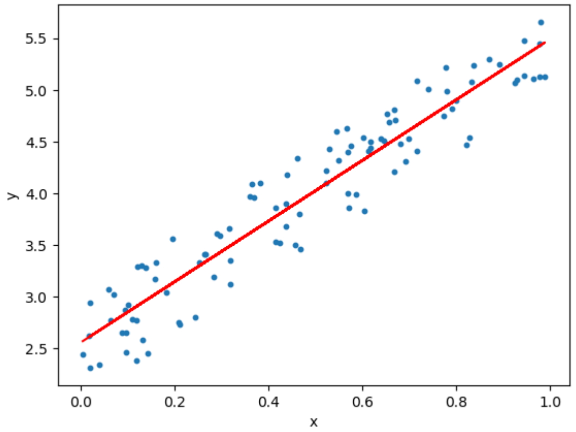


Output space: Classification vs. Regression

Depending on the choice of $\mathcal{Y}$, we have special names:

- Discrete: "**classification**". The elements of $\mathcal{Y}$ are **classes**

- Continuous: "**regression**"
  - Example: linear regression



- There are other types…

Hypothesis class

Hypothesis class

- Pick specific class of models. Ex: **linear models**:

Hypothesis class

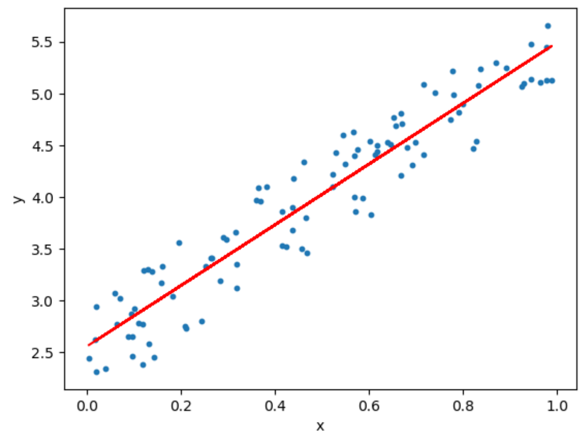
- Pick specific class of models. Ex: **linear models**:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d$$

Hypothesis class

- Pick specific class of models. Ex: **linear models**:

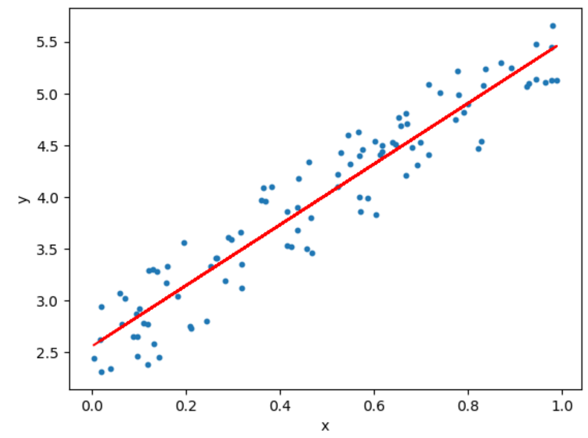
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d$$



Hypothesis class

- Pick specific class of models. Ex: **linear models**:

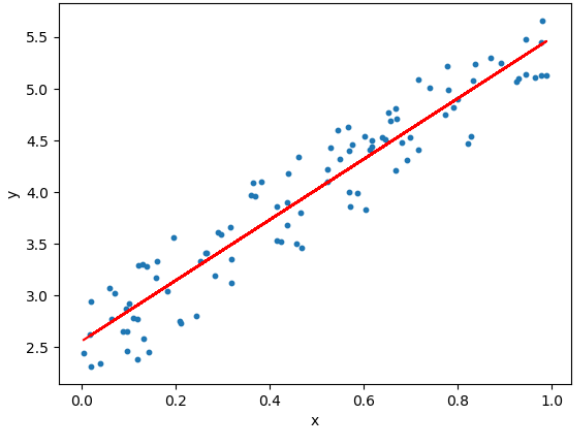
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d$$

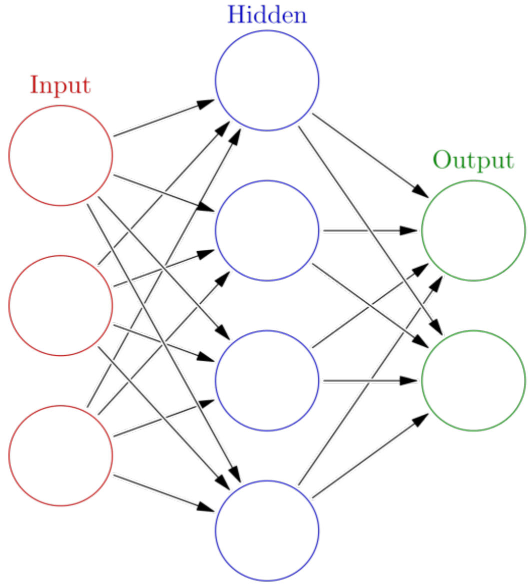


Hypothesis class

- Pick specific class of models. Ex: **linear models**:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d$$



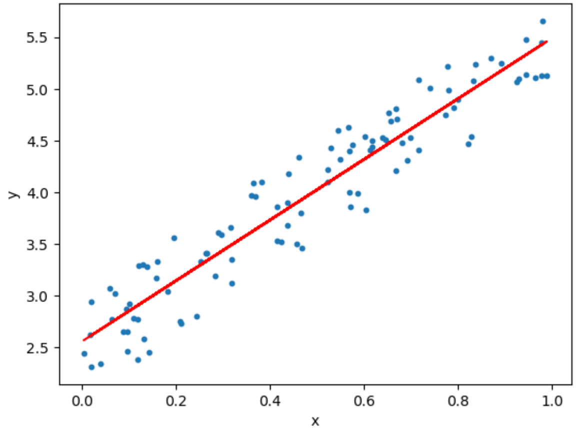


Wikipedia

Hypothesis class

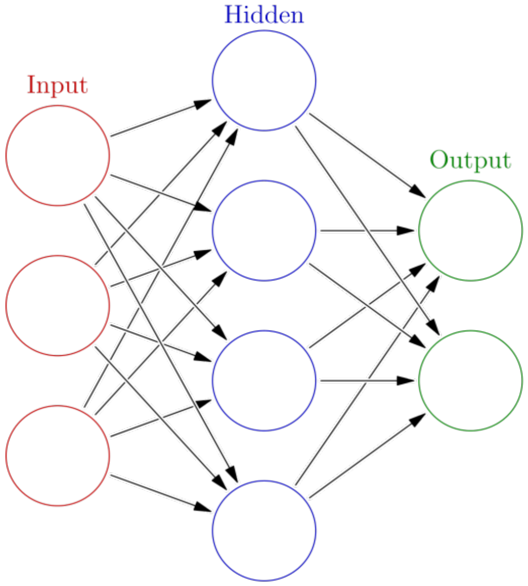
- Pick specific class of models. Ex: **linear models**:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d$$



$$f^{(0)}(x) = x$$

$$f^{(k)}(x) = \sigma(W_k^T f^{(k-1)}(x)))$$



Wikipedia

Supervised Learning: Training & Generalization

Goal: model *h* that best approximates *f*

Supervised Learning: Training & Generalization

Goal: model *h* that best approximates *f*

- One way: empirical risk minimization (ERM) on training data.

Supervised Learning: Training & Generalization

Goal: model *h* that best approximates *f*

- One way: empirical risk minimization (ERM) on training data.

$$\hat{f} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)}))$$

Supervised Learning: Training & Generalization

Goal: model *h* that best approximates *f*

- One way: empirical risk minimization (ERM) on training data.

$$\hat{f} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)})$$

↑ Hypothesis Class

Supervised Learning: Training & Generalization

Goal: model *h* that best approximates *f*

- One way: empirical risk minimization (ERM) on training data.

$$\hat{f} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)}))$$

Model prediction

Hypothesis Class

Supervised Learning: Training & Generalization

Goal: model *h* that best approximates *f*

- One way: empirical risk minimization (ERM) on training data.

$$\hat{f} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)}))$$

Model prediction

Hypothesis Class

Loss function (how far are we)?

Supervised Learning: Training & Generalization

Goal: model *h* that best approximates *f*

- One way: empirical risk minimization (ERM) on training data.

$$\hat{f} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)}))$$

Model prediction

Hypothesis Class

Loss function (how far are we)?

- Recall: we want to *generalize*.

Supervised Learning: Training & Generalization

Goal: model *h* that best approximates *f*

- One way: empirical risk minimization (ERM) on training data.

$$\hat{f} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)}))$$

Hypothesis Class

Loss function (how far are we)?

Model prediction

- Recall: we want to *generalize*.
  - Do well on future (test) data points, not just on training data.

Outline

- **Review from last time**
  - Features, labels, hypothesis class, training, generalization
- **Instance-based learning**
  - k-NN classification/regression, locally weighted regression, strengths & weaknesses, inductive bias
- **Decision trees**
  - Setup, splits, learning, information gain, strengths and weaknesses

Nearest Neighbors: Idea

Nearest Neighbors: Idea

Basic idea: “nearby” feature vectors more likely have the same label

Nearest Neighbors: Idea

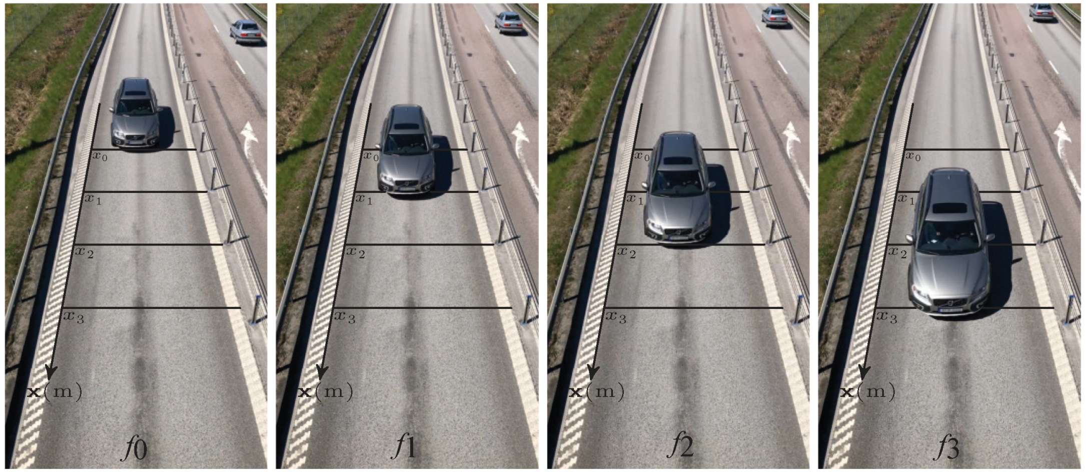
Basic idea: “nearby” feature vectors more likely have the same label

- **Example**: classify car/no car
  - Everything is similar, except the location of car

Nearest Neighbors: Idea

Basic idea: "nearby" feature vectors more likely have the same label

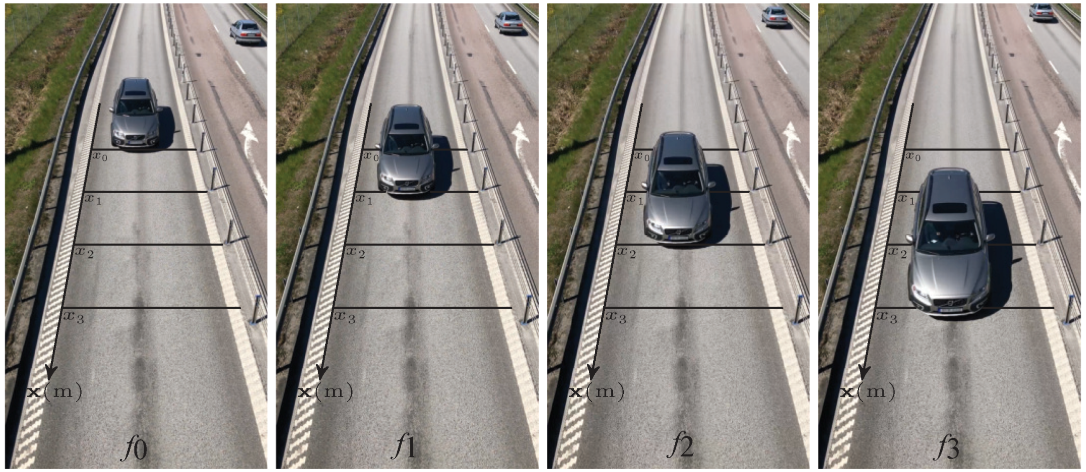
- **Example**: classify car/no car
  - Everything is similar, except the location of car



Nearest Neighbors: Idea

Basic idea: “nearby” feature vectors more likely have the same label

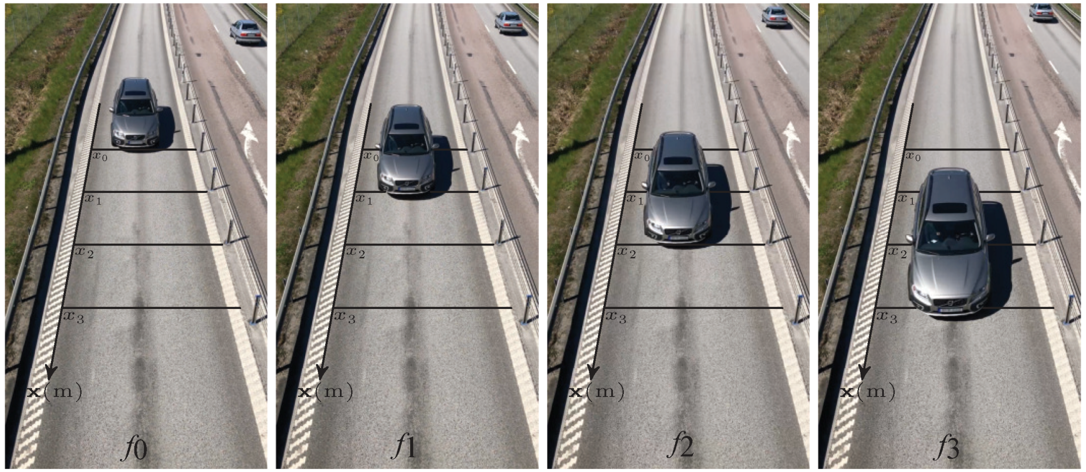
- **Example**: classify car/no car
  - Everything is similar, except the location of car



Nearest Neighbors: Idea

Basic idea: "nearby" feature vectors more likely have the same label

- **Example**: classify car/no car
  - Everything is similar, except the location of car
- What does "nearby" mean?



1-Nearest Neighbors: Algorithm

1-Nearest Neighbors: Algorithm

Training/learning: given

1-Nearest Neighbors: Algorithm

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

1-Nearest Neighbors: Algorithm

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x , find nearest training point $x^{(j)}$

1-Nearest Neighbors: Algorithm

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x , find nearest training point $x^{(j)}$
Return $y^{(j)}$

1-Nearest Neighbors: Algorithm

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x, find nearest training point $x^{(j)}$
Return $y^{(j)}$

1-Nearest Neighbors: Algorithm

1-Nearest Neighbors: Algorithm

Training/learning: given

1-Nearest Neighbors: Algorithm

Training/learning: given



1-Nearest Neighbors: Algorithm

Training/learning: given

safe

1-Nearest Neighbors: Algorithm

Training/learning: given

safe

1-Nearest Neighbors: Algorithm

Training/learning: given



safe




poisonous

Prediction: for x, find nearest training point $x^{(j)}$

1-Nearest Neighbors: Algorithm

Training/learning: given

safe

poisonous

Prediction: for x , find nearest training point $x^{(j)}$

Return $y^{(j)}$

1-Nearest Neighbors: Algorithm

Training/learning: given

safe

poisonous

x

Prediction: for x , find nearest training point $x^{(j)}$

Return $y^{(j)}$

1-Nearest Neighbors: Algorithm

Training/learning: given

safe

poisonous

Prediction: for x , find nearest training point $x^{(j)}$

Return $y^{(j)}$

x

1-Nearest Neighbors: Algorithm

Training/learning: given

safe

poisonous

Prediction: for x , find nearest training point $x^{(j)}$

Return $y^{(j)}$ poisonous

x

1NN: Decision Regions

1NN: Decision Regions

Defined by “**Voronoi Diagram**”

1NN: Decision Regions

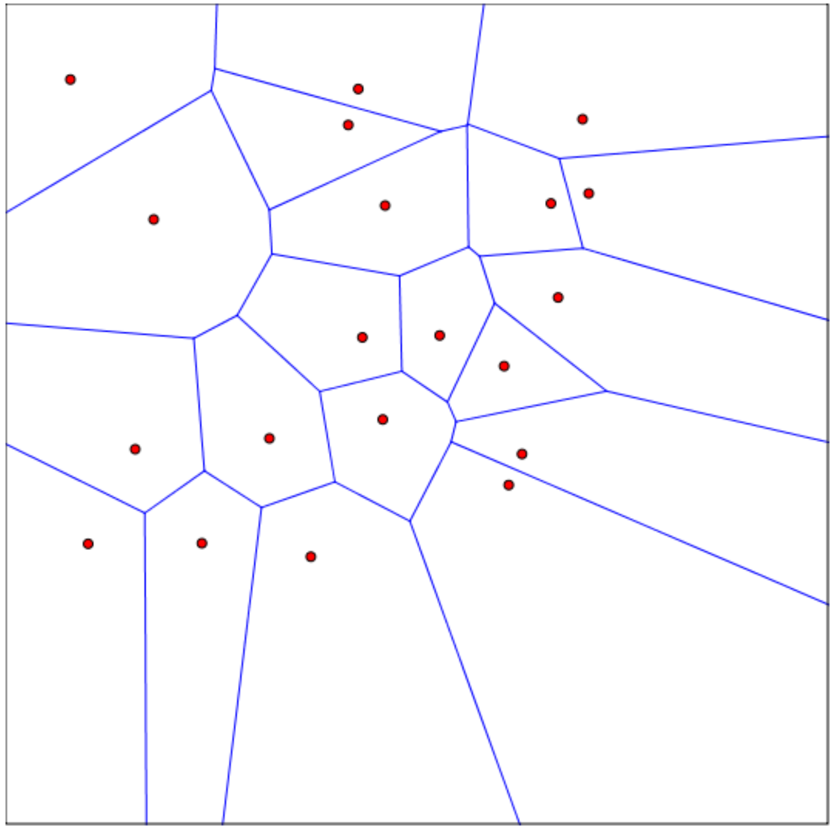
Defined by “**Voronoi Diagram**”

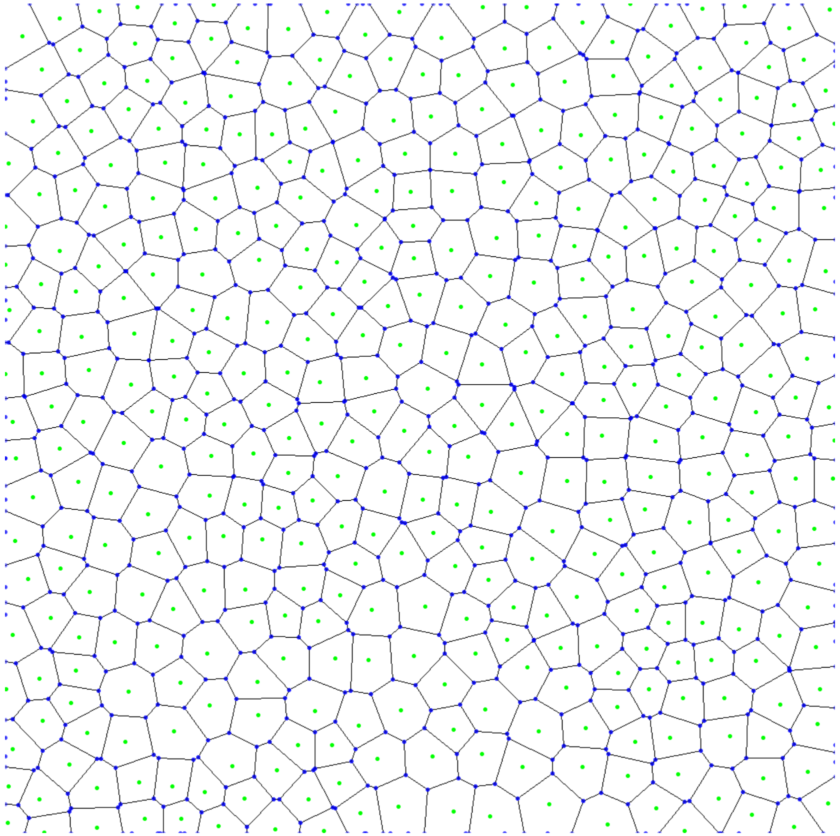
- Each cell contains points closer to a particular training point

1NN: Decision Regions

Defined by "**Voronoi Diagram**"

- Each cell contains points closer to a particular training point





k-Nearest Neighbors: Classification

Training/learning: given

k-Nearest Neighbors: Classification

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

k-Nearest Neighbors: Classification

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for given x, find ***k*** most similar training points

k-Nearest Neighbors: Classification

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for given x, find ***k*** most similar training points

Return plurality class

k-Nearest Neighbors: Classification

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for given x, find ***k*** most similar training points

Return plurality class

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{k} \mathbb{1}(y = y^{(i)})$$

k-Nearest Neighbors: Classification

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for given x, find ***k*** most similar training points

Return plurality class

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{k} \mathbb{1}(y = y^{(i)})$$

- I.e., among the ***k*** most similar points, output most popular class.

k-Nearest Neighbors: Distances

k-Nearest Neighbors: Distances

Discrete features: Hamming distance

Ex: d([‘a’, ‘b’, ‘c’], [‘d’, ‘b’, ‘e’]) = 2

$$d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} 1\{x_a^{(i)} \neq x_a^{(j)}\}$$

k-Nearest Neighbors: Distances

Discrete features: Hamming distance

Ex: d([‘a’, ‘b’, ‘c’], [‘d’, ‘b’, ‘e’]) = 2

$$d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} 1\{x_a^{(i)} \neq x_a^{(j)}\}$$

Continuous features:

k-Nearest Neighbors: Distances

Discrete features: Hamming distance

Ex: d([‘a’, ‘b’, ‘c’], [‘d’, ‘b’, ‘e’]) = 2

$$d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} 1\{x_a^{(i)} \neq x_a^{(j)}\}$$

Continuous features:

- Euclidean distance:

Ex: d([0, 0], [4, 4]) = $\sqrt{32}$

k-Nearest Neighbors: Distances

Discrete features: Hamming distance

Ex: d([‘a’, ‘b’, ‘c’], [‘d’, ‘b’, ‘e’]) = 2

$$d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} 1\{x_a^{(i)} \neq x_a^{(j)}\}$$

Continuous features:

- Euclidean distance:

$$d(x^{(i)}, x^{(j)}) = \left(\sum_{a=1}^{d} (x_a^{(i)} - x_a^{(j)})^2 \right)^{\frac{1}{2}}$$

Ex: d([0, 0], [4, 4]) = $\sqrt{32}$

k-Nearest Neighbors: Distances

Discrete features: Hamming distance

Ex: d([‘a’, ‘b’, ‘c’], [‘d’, ‘b’, ‘e’]) = 2

$$d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} 1\{x_a^{(i)} \neq x_a^{(j)}\}$$

Continuous features:

- Euclidean distance:

$$d(x^{(i)}, x^{(j)}) = \left(\sum_{a=1}^{d} (x_a^{(i)} - x_a^{(j)})^2\right)^{\frac{1}{2}}$$

Ex: d([0, 0], [4, 4]) = $\sqrt{32}$

- L1 (Manhattan) dist.:

k-Nearest Neighbors: Distances

Discrete features: Hamming distance

Ex: d([‘a’, ‘b’, ‘c’], [‘d’, ‘b’, ‘e’]) = 2

$$d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} 1\{x_a^{(i)} \neq x_a^{(j)}\}$$

Continuous features:

- Euclidean distance:

$$d(x^{(i)}, x^{(j)}) = \left(\sum_{a=1}^{d} (x_a^{(i)} - x_a^{(j)})^2 \right)^{\frac{1}{2}}$$

Ex: d([0, 0], [4, 4]) = $\sqrt{32}$

- L1 (Manhattan) dist.:

$$d(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} |x_a^{(i)} - x_a^{(j)}|$$

Ex: d([0, 0], [4, 4]) = 8

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

- Compute empirical mean/stddev for a feature (in train set)

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

- Compute empirical mean/stddev for a feature (in train set)

$$\mu_a = \frac{1}{n} \sum_{i=1}^{n} x_a^{(i)}$$

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

- Compute empirical mean/stddev for a feature (in train set)

$$\mu_a = \frac{1}{n}\sum_{i=1}^{n} x_a^{(i)} \qquad \sigma_a = \left(\frac{1}{n}\sum_{i=1}^{n}(x_a^{(i)} - \mu_i)^2\right)^{\frac{1}{2}}$$

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

- Compute empirical mean/stddev for a feature (in train set)

$$\mu_a = \frac{1}{n}\sum_{i=1}^{n} x_a^{(i)} \qquad \sigma_a = \left(\frac{1}{n}\sum_{i=1}^{n}(x_a^{(i)} - \mu_i)^2\right)^{\frac{1}{2}}$$

- Standardize features:

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

- Compute empirical mean/stddev for a feature (in train set)

$$\mu_a = \frac{1}{n}\sum_{i=1}^{n} x_a^{(i)} \qquad \sigma_a = \left(\frac{1}{n}\sum_{i=1}^{n}(x_a^{(i)} - \mu_i)^2\right)^{\frac{1}{2}}$$

- Standardize features:

$$\tilde{x}_a^{(j)} = \frac{x_a^{(j)} - \mu_a}{\sigma_a}$$

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

- Compute empirical mean/stddev for a feature (in train set)

$$\mu_a = \frac{1}{n}\sum_{i=1}^{n} x_a^{(i)} \qquad \sigma_a = \left(\frac{1}{n}\sum_{i=1}^{n}(x_a^{(i)} - \mu_i)^2\right)^{\frac{1}{2}}$$

- Standardize features:
  - Do the same for test points!

$$\tilde{x}_a^{(j)} = \frac{x_a^{(j)} - \mu_a}{\sigma_a}$$

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

- Compute empirical mean/stddev for a feature (in train set)

$$\mu_a = \frac{1}{n}\sum_{i=1}^{n} x_a^{(i)} \qquad \sigma_a = \left(\frac{1}{n}\sum_{i=1}^{n}(x_a^{(i)} - \mu_i)^2\right)^{\frac{1}{2}}$$

- Standardize features:
  - Do the same for test points!

$$\tilde{x}_a^{(j)} = \frac{x_a^{(j)} - \mu_a}{\sigma_a}$$

What problem does this solve?

k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:

- Compute empirical mean/stddev for a feature (in train set)

$$\mu_a = \frac{1}{n}\sum_{i=1}^{n} x_a^{(i)} \qquad \sigma_a = \left(\frac{1}{n}\sum_{i=1}^{n}(x_a^{(i)} - \mu_i)^2\right)^{\frac{1}{2}}$$

- Standardize features:
  - Do the same for test points!

$$\tilde{x}_a^{(j)} = \frac{x_a^{(j)} - \mu_a}{\sigma_a}$$

What problem does this solve?

Prevents high magnitude / variance features from dominating distance calculation.

k-Nearest Neighbors: Mixed Distances

Might have both discrete and continuous features:

k-Nearest Neighbors: Mixed Distances

Might have both discrete and continuous features:

- Sum two types of distances component (or sum squared etc)

k-Nearest Neighbors: Mixed Distances

Might have both discrete and continuous features:

- Sum two types of distances component (or sum squared etc)

- Might need **normalization**, (e.g. normalize individual distances to maximum value of 1)

k-Nearest Neighbors: Regression

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

k-Nearest Neighbors: Regression

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x, find ***k*** most similar training points

k-Nearest Neighbors: Regression

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x, find ***k*** most similar training points

Return

k-Nearest Neighbors: Regression

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x, find ***k*** most similar training points

Return

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y^{(i)}$$

k-Nearest Neighbors: Regression

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x, find ***k*** most similar training points

Return

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y^{(i)}$$

- I.e., among the ***k*** points, output mean label.

k-Nearest Neighbors: Locally Weighted k-NN

Could contribute to predictions via a weighted distance

k-Nearest Neighbors: Locally Weighted k-NN

Could contribute to predictions via a weighted distance

- All k no longer equally contribute

k-Nearest Neighbors: Locally Weighted k-NN

Could contribute to predictions via a weighted distance

- All k no longer equally contribute
- Classification:

$$\hat{y} \leftarrow \arg\max_{v \in \mathcal{Y}} \sum_{i=1}^{k} \frac{1}{d(x, x^{(i)})^2} \delta(v, y^{(i)})$$

k-Nearest Neighbors: Locally Weighted k-NN

Could contribute to predictions via a weighted distance

- All k no longer equally contribute
- Classification:

$$\hat{y} \leftarrow \arg\max_{v \in \mathcal{Y}} \sum_{i=1}^{k} \frac{1}{d(x, x^{(i)})^2} \delta(v, y^{(i)})$$

- Regression

$$\hat{y} \leftarrow \frac{\sum_{i=1}^{k} y^{(i)} / d(x, x^{(i)})^2}{\sum_{i=1}^{k} 1 / d(x, x^{(i)})^2}$$

Dealing with **Irrelevant Features**

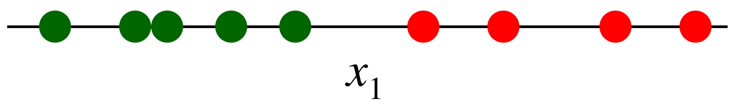
One relevant feature x_1

1-NN rule classifies each instance correctly

Dealing with **Irrelevant Features**

One relevant feature x_1

1-NN rule classifies each instance correctly

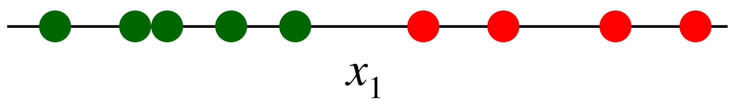


Dealing with **Irrelevant Features**

One relevant feature x_1

1-NN rule classifies each instance correctly

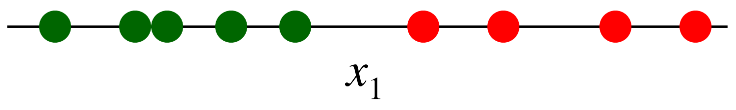
Effect of an irrelevant feature x_2 on distances and nearest neighbors



Dealing with **Irrelevant Features**

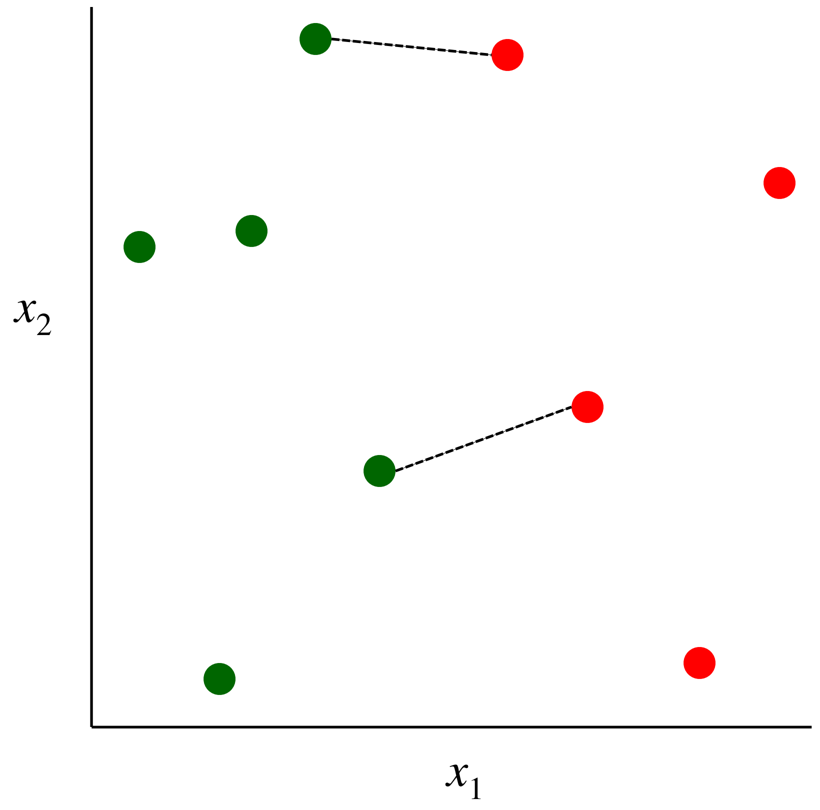
One relevant feature x_1

1-NN rule classifies each instance correctly



Effect of an irrelevant feature x_2

on distances and nearest neighbors



kNN: Strengths & Weaknesses

kNN: Strengths & Weaknesses

Strengths

kNN: Strengths & Weaknesses

Strengths

- Easy to explain predictions

kNN: Strengths & Weaknesses

Strengths

- Easy to explain predictions
- Simple to implement and conceptualize.

kNN: Strengths & Weaknesses

Strengths

- Easy to explain predictions
- Simple to implement and conceptualize.
- No training!

kNN: Strengths & Weaknesses

Strengths

- Easy to explain predictions
- Simple to implement and conceptualize.
- No training!
- Often good in practice

kNN: Strengths & Weaknesses

Strengths

- Easy to explain predictions
- Simple to implement and conceptualize.
- No training!
- Often good in practice

Weaknesses

kNN: Strengths & Weaknesses

Strengths

- Easy to explain predictions
- Simple to implement and conceptualize.
- No training!
- Often good in practice

Weaknesses

- Sensitive to irrelevant + correlated features
  - Can try to solve via variations.

kNN: Strengths & Weaknesses

Strengths

- Easy to explain predictions
- Simple to implement and conceptualize.
- No training!
- Often good in practice

Weaknesses

- Sensitive to irrelevant + correlated features
  - Can try to solve via variations.
- Prediction stage can be expensive

kNN: Strengths & Weaknesses

Strengths

- Easy to explain predictions
- Simple to implement and conceptualize.
- No training!
- Often good in practice

Weaknesses

- Sensitive to irrelevant + correlated features
  - Can try to solve via variations.
- Prediction stage can be expensive
- No “model” to interpret

Inductive Bias

Inductive Bias

- ***Inductive bias*:** assumptions a learner uses to predict y_i for a previously unseen instance $\boldsymbol{x}_i$

Inductive Bias

- ***Inductive bias*:** assumptions a learner uses to predict y_i for a previously unseen instance $\boldsymbol{x}_i$
- Two components (mostly)
  - *hypothesis space bias*: determines the models that can be represented
  - *preference bias*: specifies a preference ordering within the space of models

Inductive Bias

- ***Inductive bias***: assumptions a learner uses to predict y_i for a previously unseen instance $\boldsymbol{x}_i$
- Two components (mostly)
  - *hypothesis space bias*: determines the models that can be represented
  - *preference bias*: specifies a preference ordering within the space of models

learner	hypothesis space bias	preference bias
k-NN	Decomposition of space determined by nearest neighbors	Instances in neighborhood belong to same class

Break & Quiz

Q2-1: Table shows all the training points in 2D space and their labels. Assume a 3-NN classifier and Euclidean distance. What should be the labels of the points A: (1, 1) and B(2, 1)?

1. A: +, B: -
2. A: -, B: +
3. A: -, B: -
4. A: +, B: +

x	y	label
0	0	+
1	0	+
2	0	+
2	2	+
0	1	-
0	2	-
1	2	-
3	1	-

Q2-1: Table shows all the training points in 2D space and their labels. Assume 3NN classifier and Euclidean distance. What should be the labels of the points A: (1, 1) and B(2, 1)?

1. A: +, B: -
2. A: -, B: + ←
3. A: -, B: -
4. A: +, B: +

x	y	label
0	0	+
1	0	+
2	0	+
2	2	+
0	1	-
0	2	-
1	2	-
3	1	-

3 nearest neighbors to point A are (0, 1) [-], (1, 0) [+], (1, 2) [-]. Hence, the label should be **[-]**

3 nearest neighbors to point B are (2, 0) [+], (2, 2) [+], (3, 1) [-]. Hence, the label should be **[+]**

Q2-2: In a distance-weighted nearest neighbor, which of the following weight is **NOT** appropriate? Let p be the test data point and x_i {i = 1: N} be training data points.

1. $w_i = d(p, x_i)^{1/2}$
2. $w_i = d(p, x_i)^{-2}$
3. $w_i = \exp(-d(p, x_i))$
4. $w_i = 1$

Q2-2: In a distance-weighted nearest neighbor, which of the following weights is **NOT** appropriate? Let p be the test data point and x_i {i = 1: N} be training data points.

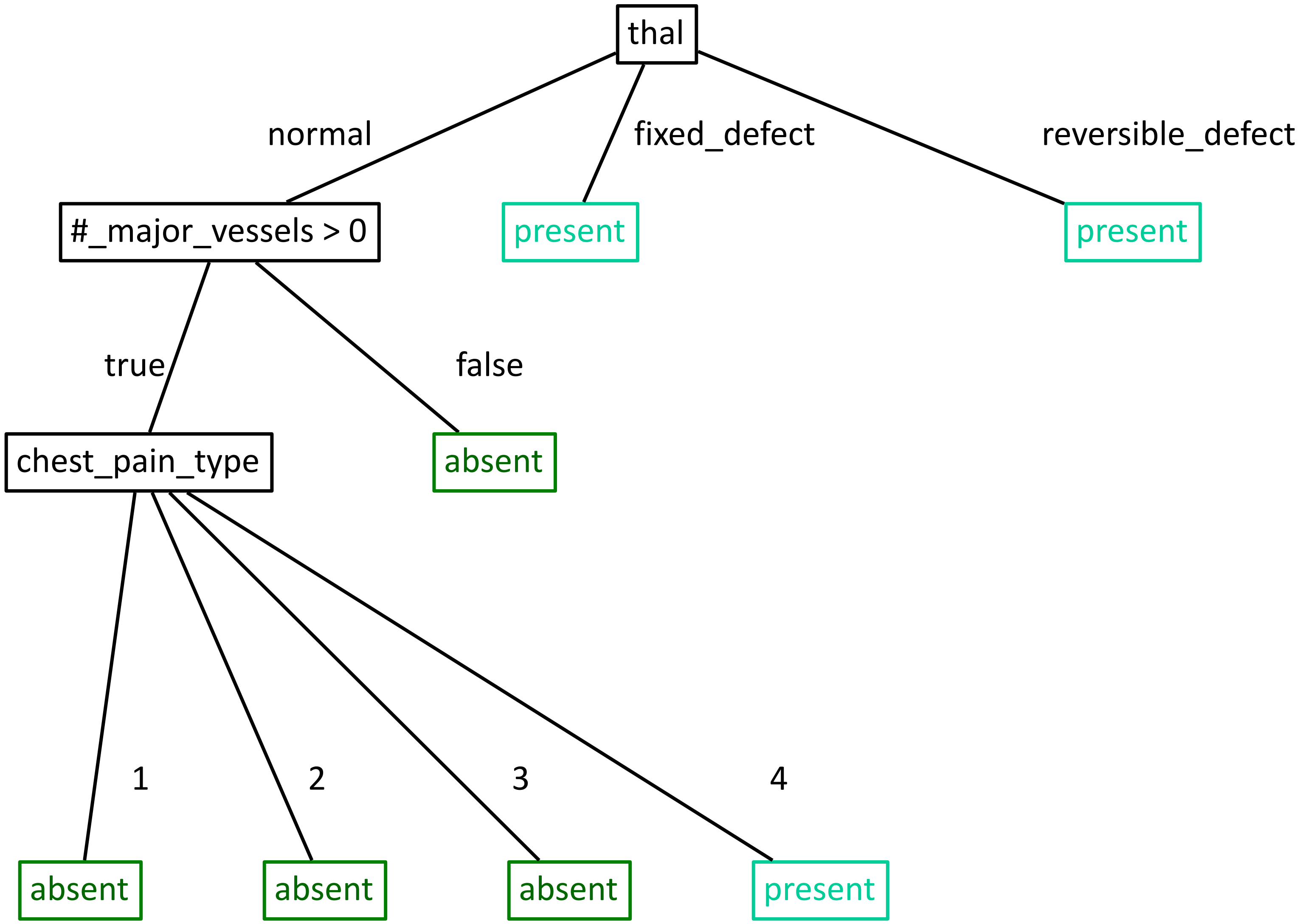
1. $w_i = d(p, x_i)^{½}$
2. $w_i = d(p, x_i)^{-2}$
3. $w_i = \exp(-d(p, x_i))$
4. $w_i = 1$

The intuition behind weighted kNN, is to give more weight to the points which are nearby and less weight to the points which are farther away. Any function whose value decreases as the distance increases can be used as a function for the weighted knn classifier. w = 1 is also **OK** as it reduces to our traditional nearest-neighbor algorithm.

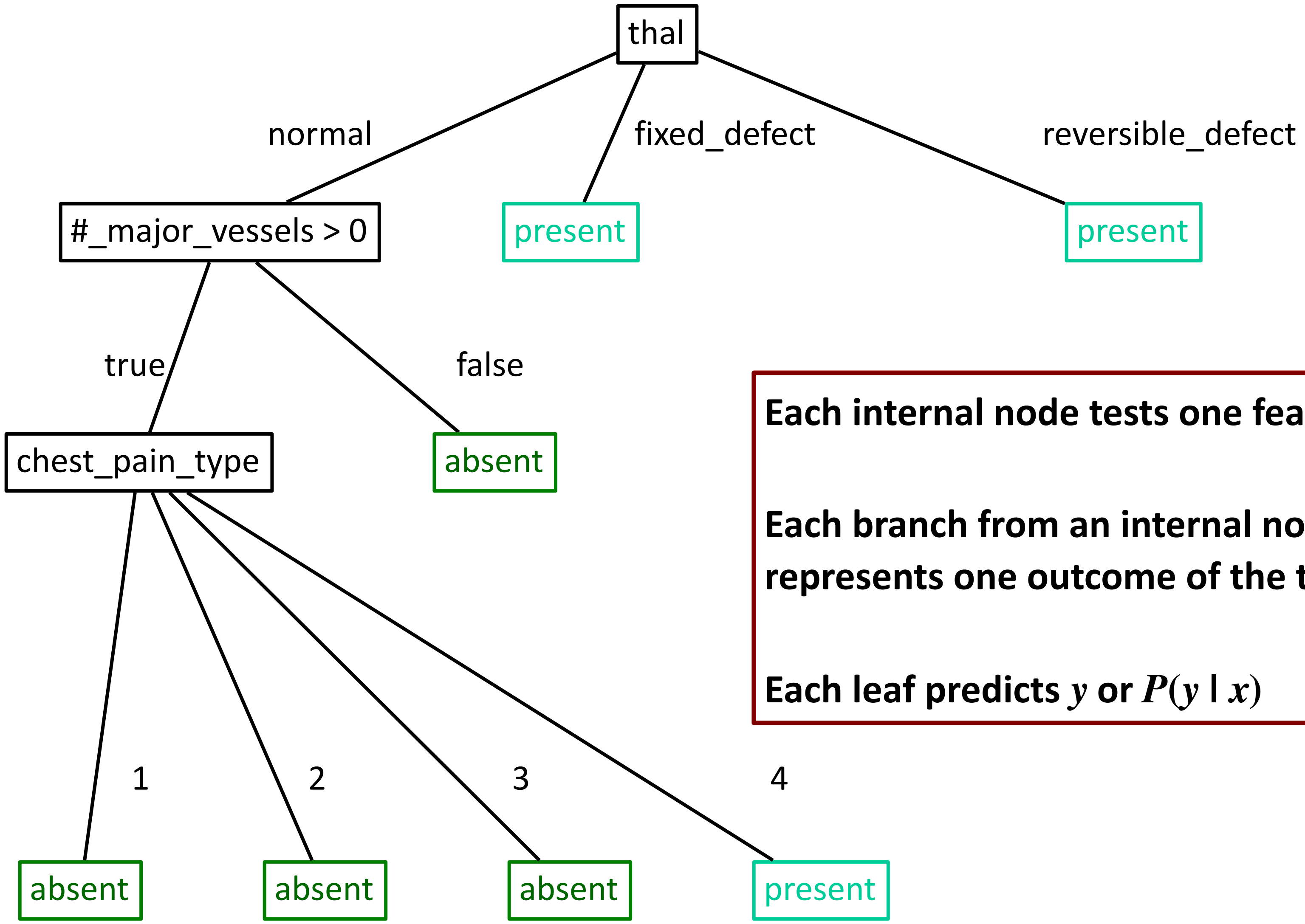
Outline

- **Review from last time**
  - Features, labels, hypothesis class, training, generalization
- **Instance-based learning**
  - k-NN classification/regression, locally weighted regression, strengths & weaknesses, inductive bias
- **Decision trees**
  - Setup, splits, learning, information gain, strengths and weaknesses

Decision Trees: Heart Disease Example



Decision Trees: Heart Disease Example



Each internal node tests one feature x_i

Each branch from an internal node represents one outcome of the test

Each leaf predicts y or $P(y \mid x)$

Decision Trees: Logical Formulas

Decision Trees: Logical Formulas

- Suppose $X_1 \ldots X_5$ are Boolean features, and Y is also Boolean
  - How would you represent the following with decision trees?

Decision Trees: Logical Formulas

- Suppose $X_1 \ldots X_5$ are Boolean features, and Y is also Boolean
    - How would you represent the following with decision trees?

$$Y = X_2 X_5 \quad (\text{i.e.}, Y = X_2 \wedge X_5)$$

Decision Trees: Logical Formulas

- Suppose $X_1 \ldots X_5$ are Boolean features, and Y is also Boolean
  - How would you represent the following with decision trees?

$$Y = X_2 X_5 \quad (\text{i.e.}, Y = X_2 \wedge X_5)$$

$$Y = X_2 \vee X_5$$

Decision Trees: Logical Formulas

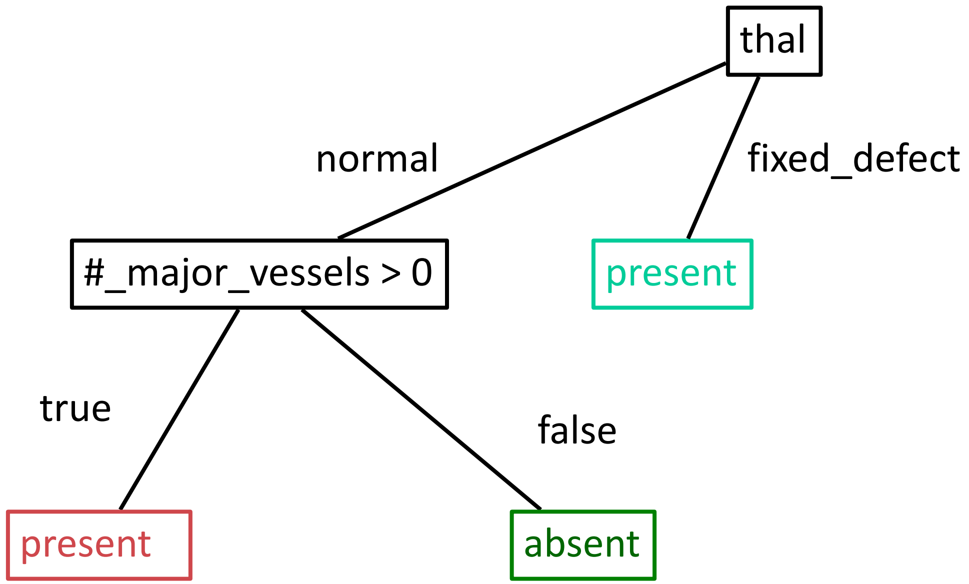
- Suppose $X_1 \ldots X_5$ are Boolean features, and Y is also Boolean
  - How would you represent the following with decision trees?

$$Y = X_2 X_5 \quad (\text{i.e.,} Y = X_2 \wedge X_5)$$

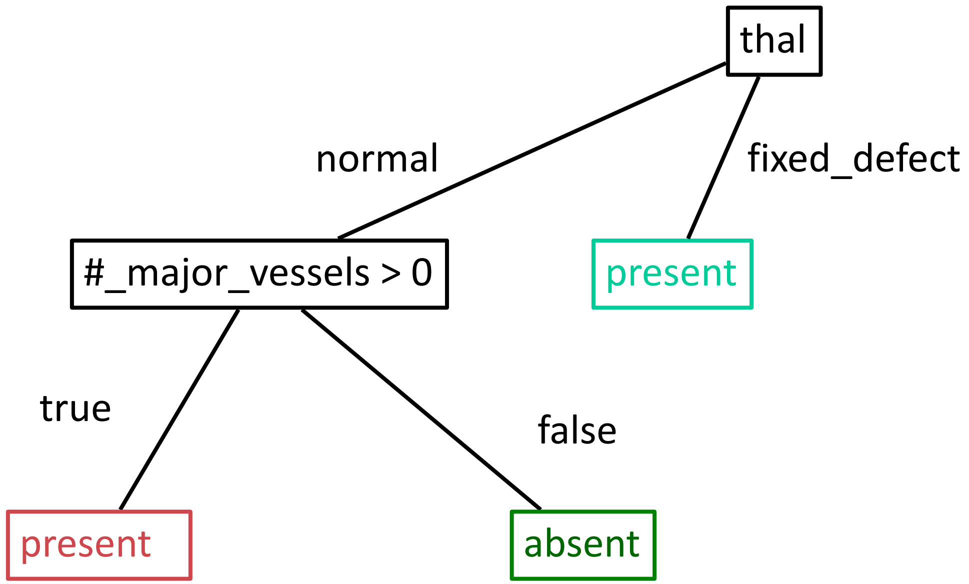
$$Y = X_2 \vee X_5$$

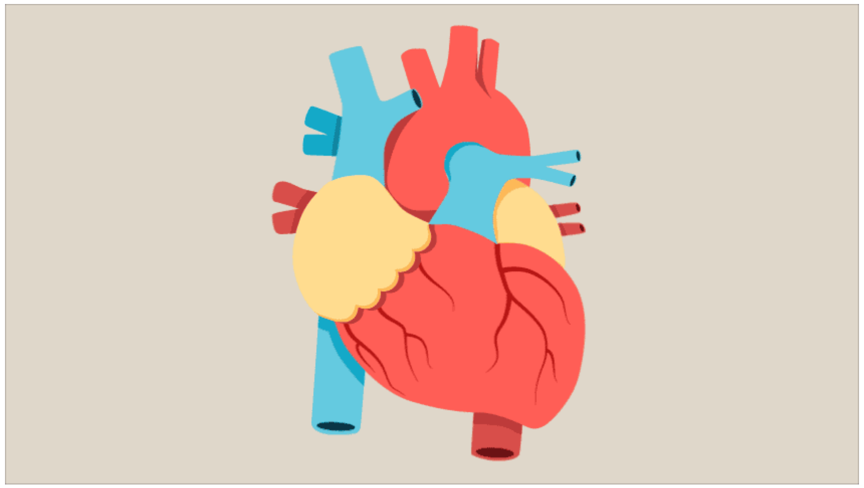
$$Y = X_2 X_5 \vee X_3 \neg X_1$$

Decision Trees: Textual Description

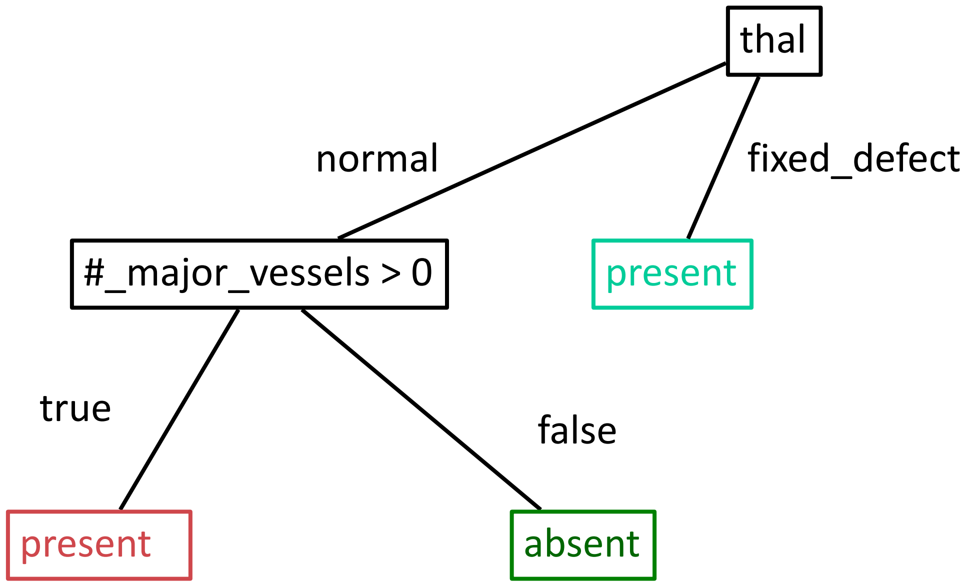


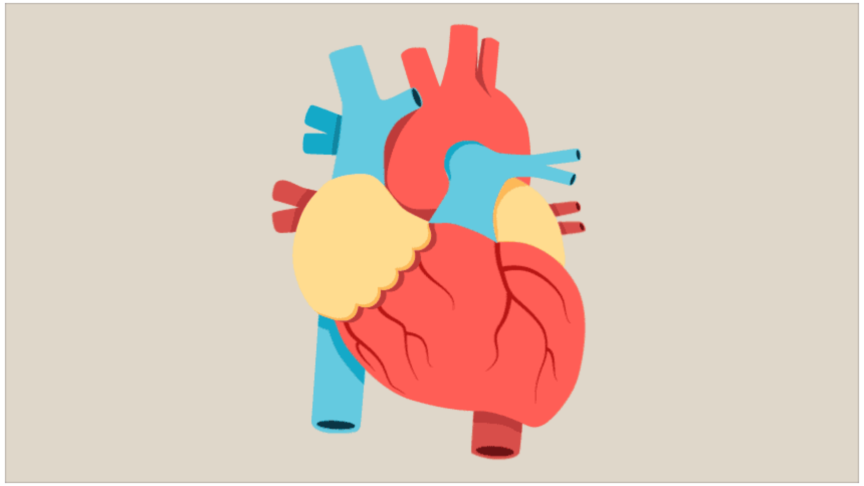
Decision Trees: Textual Description





Decision Trees: Textual Description





thal = normal

[#_major_vessels > 0] = true: present

[#_major_vessels > 0] = false: absent

thal = fixed_defect: present

Decision Trees: Mushrooms Example

```
odor = a: e (400.0)
odor = c: p (192.0)
odor = f: p (2160.0)
odor = l: e (400.0)
odor = m: p (36.0)
odor = n
|   spore-print-color = b: e (48.0)
|   spore-print-color = h: e (48.0)
|   spore-print-color = k: e (1296.0)
|   spore-print-color = n: e (1344.0)
|   spore-print-color = o: e (48.0)
|   spore-print-color = r: p (72.0)
|   spore-print-color = u: e (0.0)
|   spore-print-color = w
|   |   gill-size = b: e (528.0)
|   |   gill-size = n
|   |   |   gill-spacing = c: p (32.0)
|   |   |   gill-spacing = d: e (0.0)
|   |   |   gill-spacing = w
|   |   |   |   population = a: e (0.0)
|   |   |   |   population = c: p (16.0)
|   |   |   |   population = n: e (0.0)
|   |   |   |   population = s: e (0.0)
|   |   |   |   population = v: e (48.0)
|   |   |   |   population = y: e (0.0)
|   spore-print-color = y: e (48.0)
odor = p: p (256.0)
odor = s: p (576.0)
odor = y: p (576.0)
```

if odor=almond, predict edible

if odor=none ∧
spore-print-color=white ∧
gill-size=narrow ∧
gill-spacing=crowded,
predict poisonous

Decision Trees: Learning

- **Learning Algorithm**:

Decision Trees: Learning

- **Learning Algorithm**:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Decision Trees: Learning

- **Learning Algorithm**:

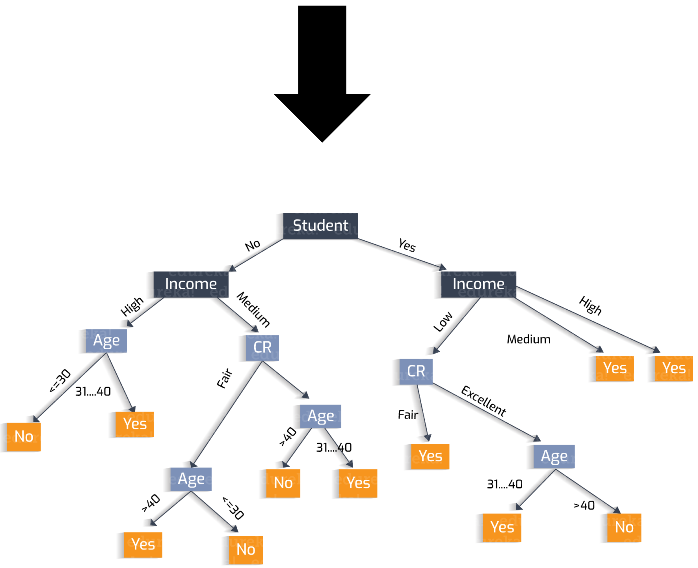
$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

⬇

Decision Trees: Learning

- **Learning Algorithm:**

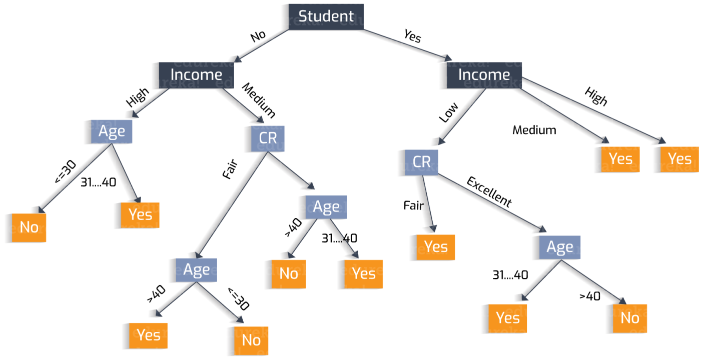
$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$



Decision Trees: Learning

- **Learning Algorithm:**

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$



MakeSubtree(set of training instances D)

C = DetermineCandidateSplits(D)

if stopping criteria is met

  make a leaf node N

  determine class label for N

else

  make an internal node N

  S = FindBestSplit(D, C)

  for each group k of S

    D_k = subset of training data in group k

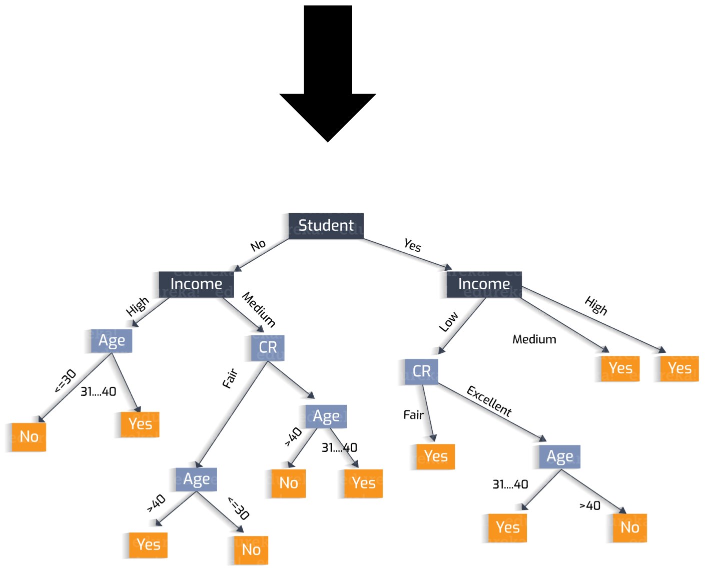
    k^{th} child of N = MakeSubtree(D_k)

return subtree rooted at N

Decision Trees: Learning

- **Learning Algorithm:**

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$



MakeSubtree(set of training instances D)

C = **DetermineCandidateSplits**(D)

if **stopping criteria** is met

  make a leaf node N

  determine class label for N

else

  make an internal node N

  S = **FindBestSplit**(D, C)

  for each group k of S

    D_k = subset of training data in group k

    k^{th} child of N = MakeSubtree(D_k)

return subtree rooted at N

1. DT Learning: Candidate Splits

First, need to determine how to **split features**

1. DT Learning: Candidate Splits

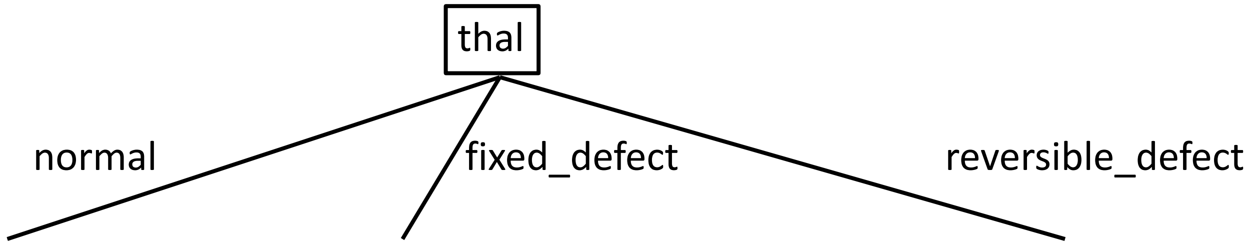
First, need to determine how to **split features**

- Splits on nominal features have one branch per value

1. DT Learning: Candidate Splits

First, need to determine how to **split features**

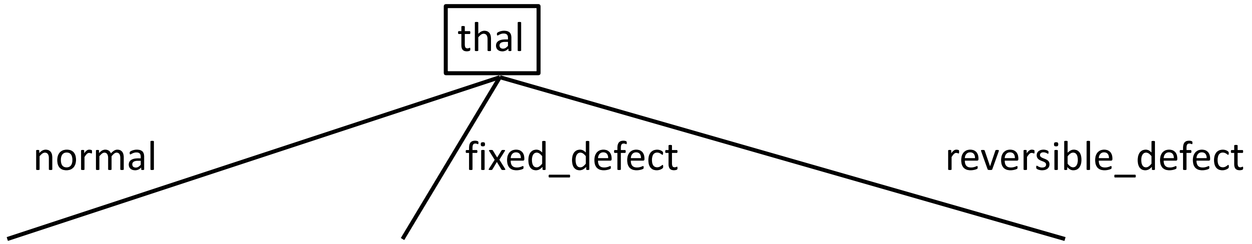
- Splits on nominal features have one branch per value



1. DT Learning: Candidate Splits

First, need to determine how to **split features**

- Splits on nominal features have one branch per value

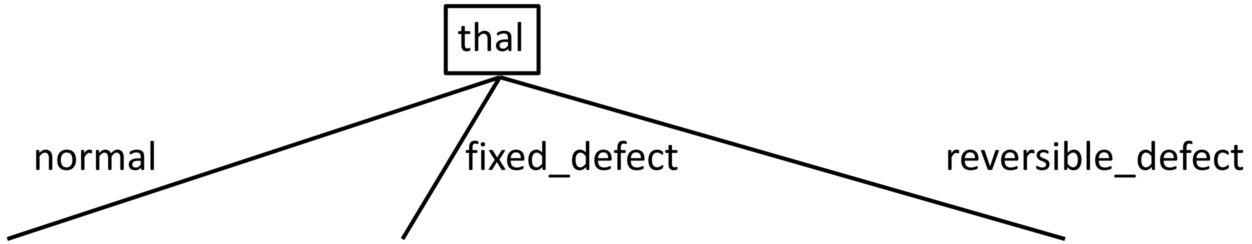


- Splits on numeric features use a threshold/interval

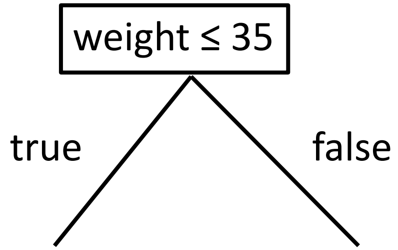
1. DT Learning: Candidate Splits

First, need to determine how to **split features**

- Splits on nominal features have one branch per value



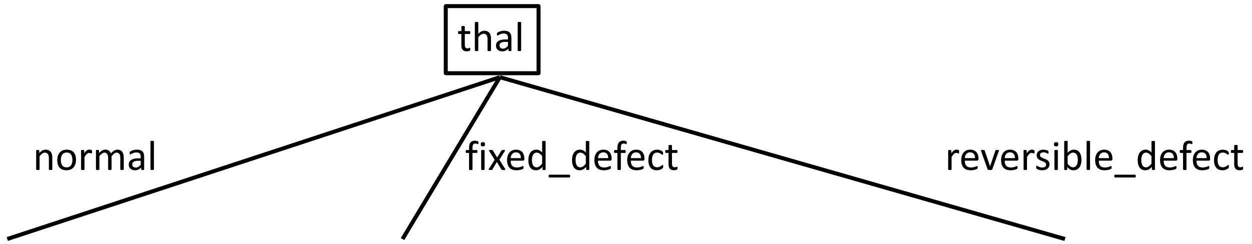
- Splits on numeric features use a threshold/interval



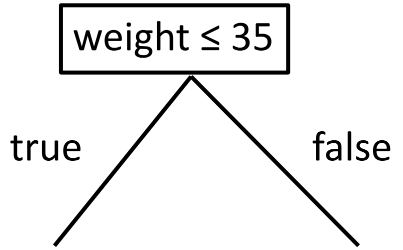
1. DT Learning: Candidate Splits

First, need to determine how to **split features**

- Splits on nominal features have one branch per value



- Splits on numeric features use a threshold/interval



ID3, C4.5

DT Learning: Numeric Feature Splits

Given a set of training instances D and a specific feature X_i

DT Learning: Numeric Feature Splits

Given a set of training instances D and a specific feature X_i

- Sort the values of X_i in D

DT Learning: Numeric Feature Splits

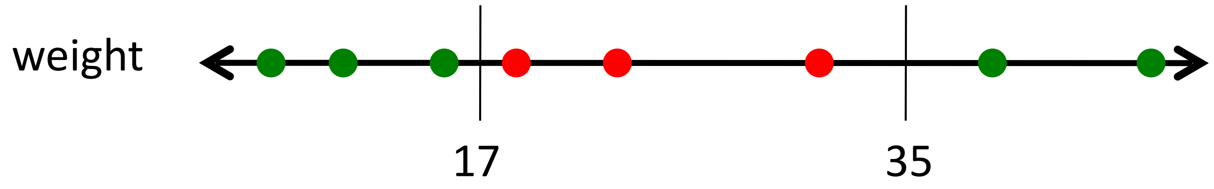
Given a set of training instances D and a specific feature X_i

- Sort the values of X_i in D
- Evaluate split thresholds in intervals between instances of different classes

DT Learning: Numeric Feature Splits

Given a set of training instances D and a specific feature X_i

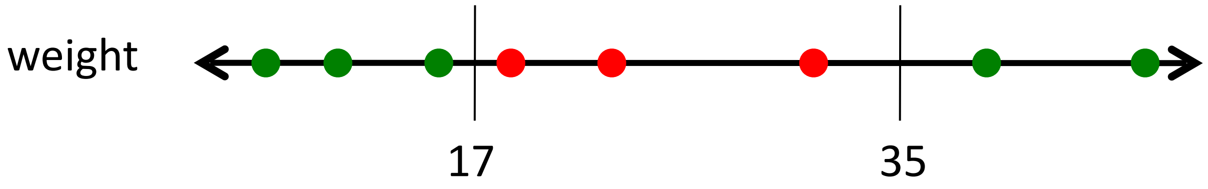
- Sort the values of X_i in D
- Evaluate split thresholds in intervals between instances of different classes

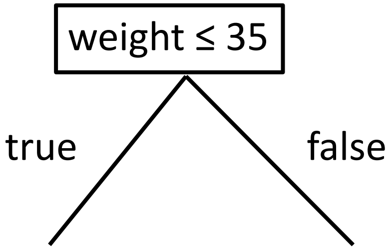


DT Learning: Numeric Feature Splits

Given a set of training instances D and a specific feature X_i

- Sort the values of X_i in D
- Evaluate split thresholds in intervals between instances of different classes





Numeric Feature Splits Algorithm

// Run this subroutine for each numeric feature at each node of DT induction

DetermineCandidateNumericSplits(set of training instances D, feature X_i)

$C = \{\}$ // initialize set of candidate splits for feature X_i

let v_j denote the value of X_i for the j^{th} data point

sort the dataset using v_j as the key for each data point

for each pair of adjacent v_j, v_{j+1} in the sorted order

if the corresponding class labels are different

add candidate split $X_i \leq (v_j + v_{j+1})/2$ to C

return C

DT: Splits on Nominal Features

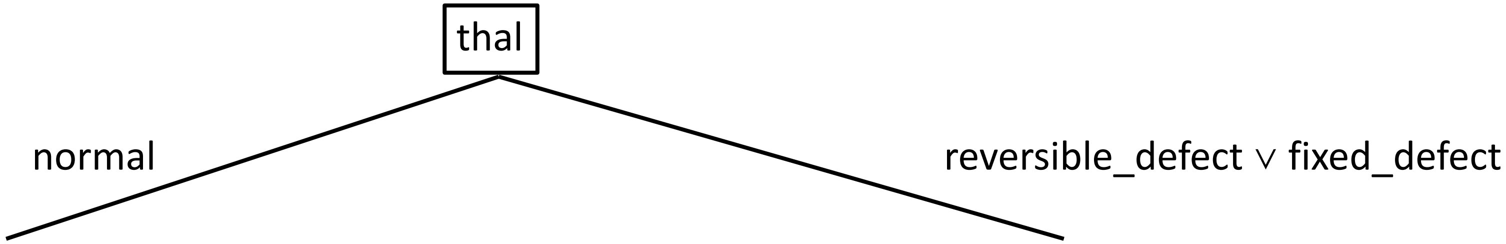
Instead of using k-way splits for k-valued features, could require binary splits on all nominal features.

- CART algorithm (popular DT algorithm) does this.

DT: Splits on Nominal Features

Instead of using k-way splits for k-valued features, could require binary splits on all nominal features.

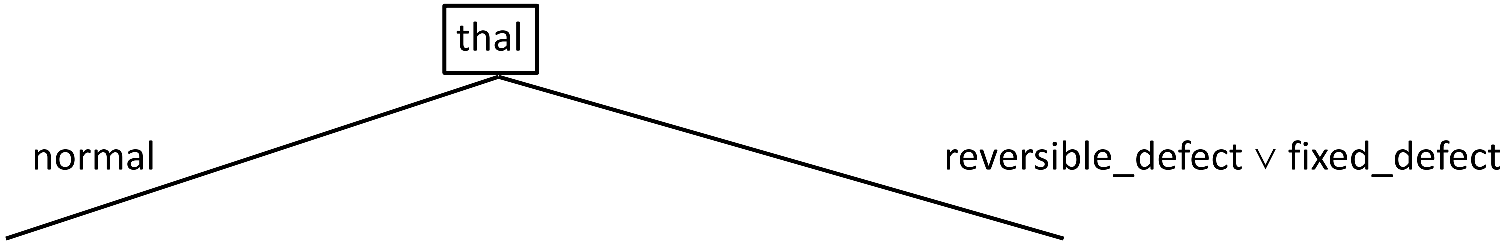
- CART algorithm (popular DT algorithm) does this.

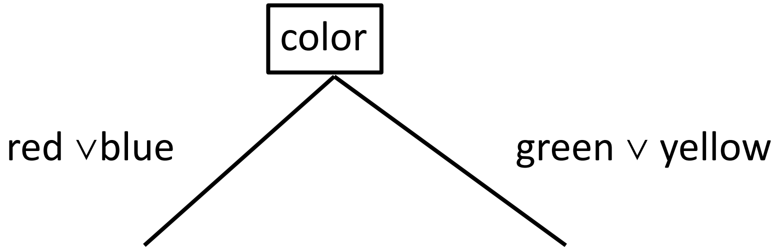


DT: Splits on Nominal Features

Instead of using k-way splits for k-valued features, could require binary splits on all nominal features.

- CART algorithm (popular DT algorithm) does this.





Decision Trees: Learning

- **Learning Algorithm**:

Decision Trees: Learning

- **Learning Algorithm**:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Decision Trees: Learning

- **Learning Algorithm**:

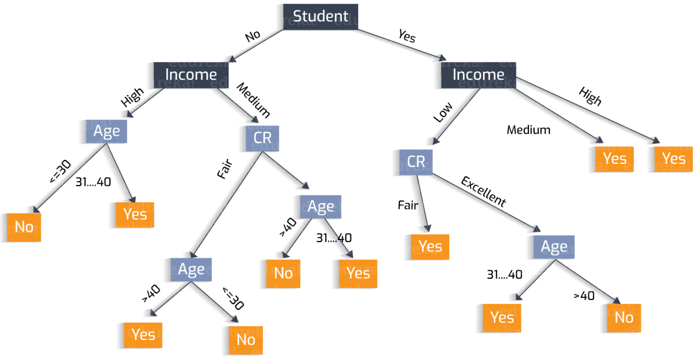
$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

⬇

Decision Trees: Learning

- **Learning Algorithm:**

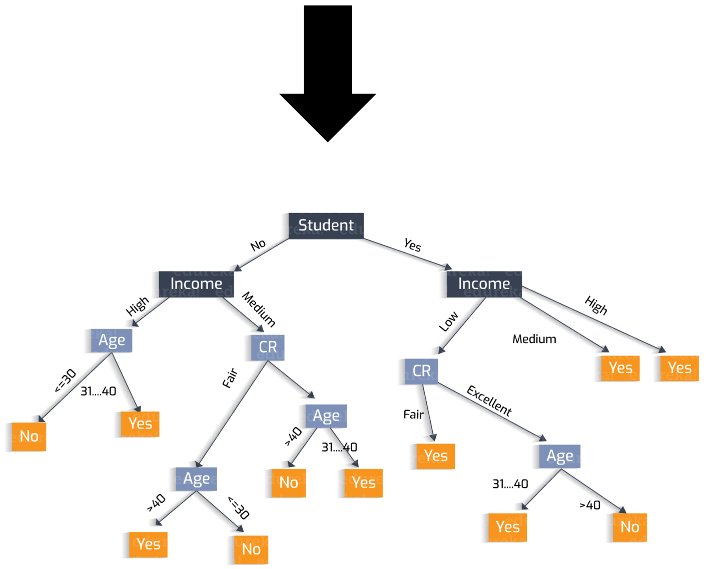
$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$



Decision Trees: Learning

- **Learning Algorithm:**

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$



MakeSubtree(set of training instances D)

C = **DetermineCandidateSplits**(D)

if **stopping criteria** is met

make a leaf node N

determine class label for N

else

make an internal node N

S = **FindBestSplit**(D, C)

for each group k of S

D_k = subset of training data in group k

k^{th} child of N = MakeSubtree(D_k)

return subtree rooted at N

Decision tree Learning: Finding the Best Splits

Decision tree Learning: Finding the Best Splits

How to we select the best feature to split on at each step?

Decision tree Learning: Finding the Best Splits

How to we select the best feature to split on at each step?

- **Hypothesis**: simplest tree that classifies the training instances accurately will generalize.

Decision tree Learning: Finding the Best Splits

How to we select the best feature to split on at each step?

- **Hypothesis**: simplest tree that classifies the training instances accurately will generalize.

Occam's razor

- "when you have two competing theories that make the same predictions, the simpler one is the better"

Decision tree Learning: Finding the Best Splits

How to we select the best feature to split on at each step?

- **Hypothesis**: simplest tree that classifies the training instances accurately will generalize.

Occam's razor

- "when you have two competing theories that make the same predictions, the simpler one is the better"

DT Learning: Finding the Best Splits

How to we select the best feature to split on at each step?

- **Hypothesis**: simplest tree that classifies the training instances accurately will generalize

DT Learning: Finding the Best Splits

How to we select the best feature to split on at each step?

- **Hypothesis**: simplest tree that classifies the training instances accurately will generalize

Why is Occam's razor a **reasonable heuristic?**

DT Learning: Finding the Best Splits

How to we select the best feature to split on at each step?

- **Hypothesis**: simplest tree that classifies the training instances accurately will generalize

Why is Occam's razor a **reasonable heuristic?**

- There are fewer short models (i.e. small trees) than long ones

DT Learning: Finding the Best Splits

How to we select the best feature to split on at each step?

- **Hypothesis**: simplest tree that classifies the training instances accurately will generalize

Why is Occam’s razor a **reasonable heuristic?**

- There are fewer short models (i.e. small trees) than long ones
- A short model is unlikely to fit the training data well by chance

DT Learning: Finding the Best Splits

How to we select the best feature to split on at each step?

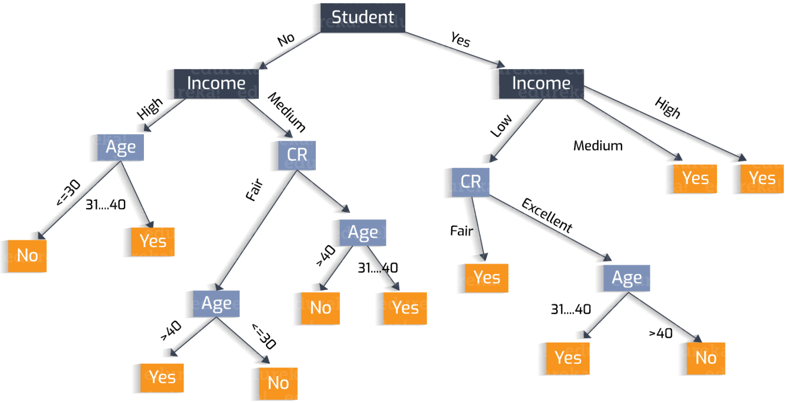
- **Hypothesis**: simplest tree that classifies the training instances accurately will generalize

Why is Occam's razor a **reasonable heuristic?**

- There are fewer short models (i.e. small trees) than long ones
- A short model is unlikely to fit the training data well by chance
- A long model is more likely to fit the training data well coincidentally

DT Learning: Finding Optimal Splits?

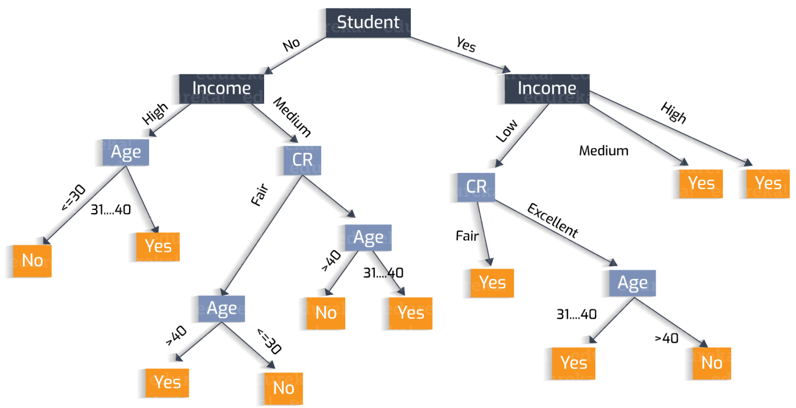
Can we find and return the smallest possible decision tree that accurately classifies the training set?



DT Learning: Finding Optimal Splits?

Can we find and return the smallest possible decision tree that accurately classifies the training set?

- **NO! This is an NP-hard problem**

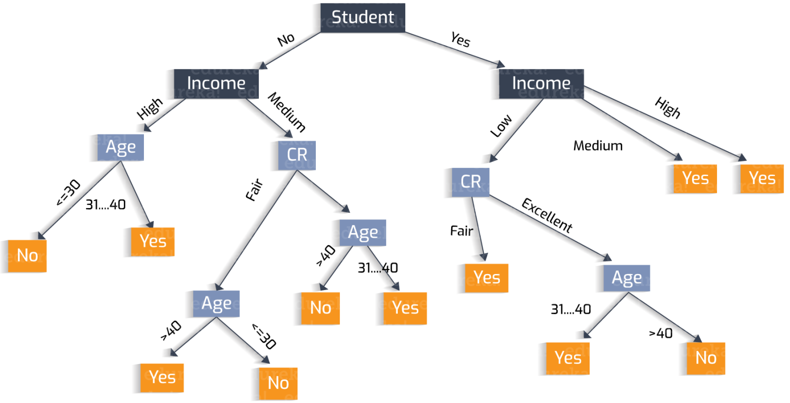


DT Learning: Finding Optimal Splits?

Can we find and return the smallest possible decision tree that accurately classifies the training set?

- **NO! This is an NP-hard problem**

  [Hyafil & Rivest, *Information Processing Letters, 1976*]



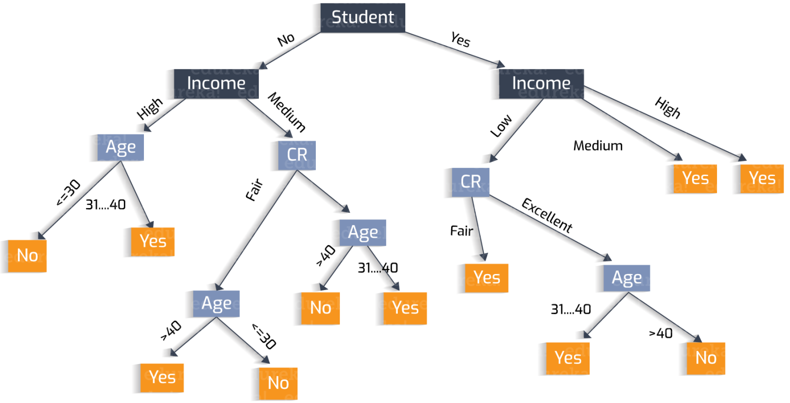
DT Learning: Finding Optimal Splits?

Can we find and return the smallest possible decision tree that accurately classifies the training set?

- **NO! This is an NP-hard problem**

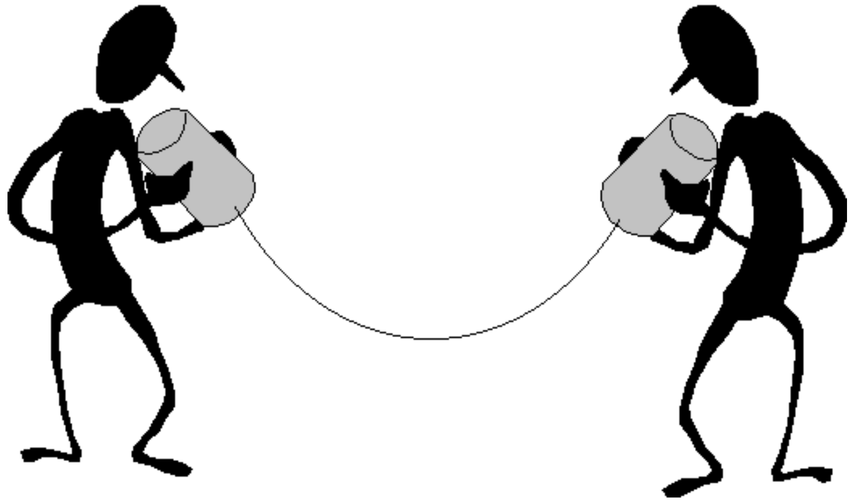
  [Hyafil & Rivest, *Information Processing Letters, 1976*]

- Instead, we'll use an information-theoretic heuristic to greedily choose splits



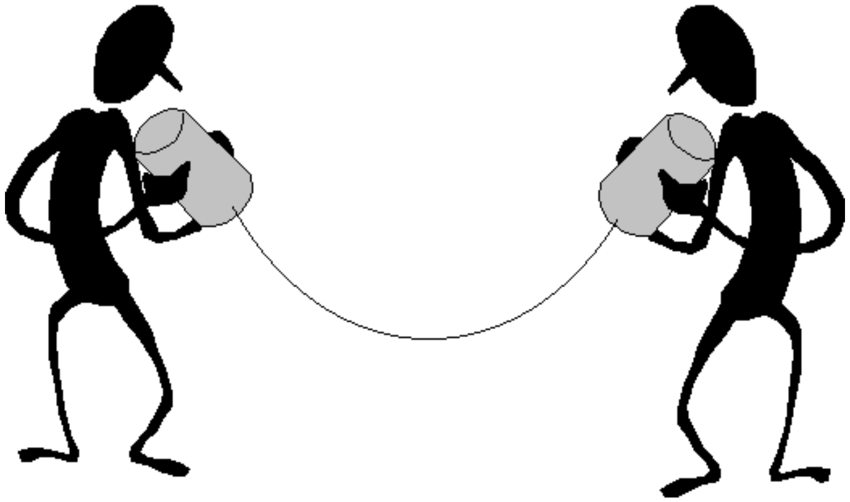
Information Theory: Super-Quick Intro

Information Theory: Super-Quick Intro



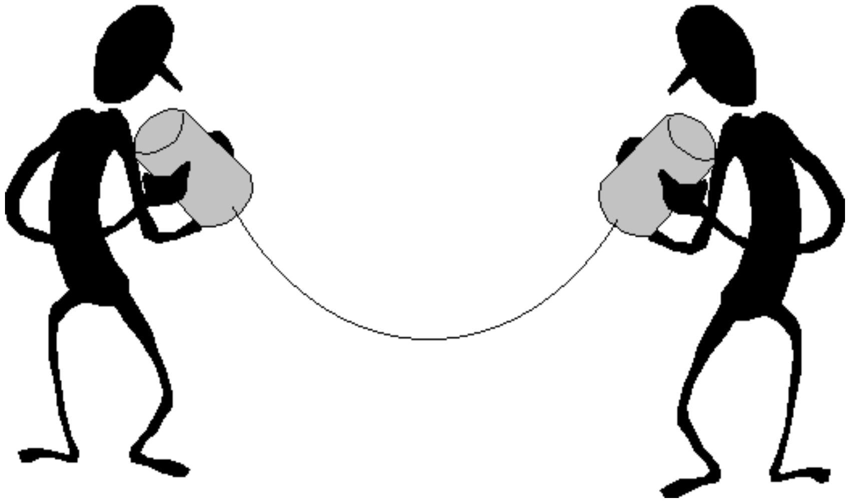
Information Theory: Super-Quick Intro

- **Goal**: communicate information to a receiver *in bits*
- Ex: as bikes go past, communicate the maker of each bike



Information Theory: Super-Quick Intro

- **Goal**: communicate information to a receiver *in bits*
- Ex: as bikes go past, communicate the maker of each bike



Information Theory: Encoding

Information Theory: Encoding

- Could send out the names of the manufacturers in binary coded ASCII
  - Suppose there are 4: **Trek**, **Specialized**, **Cervelo**, **Serrota**

Information Theory: Encoding

- Could send out the names of the manufacturers in binary coded ASCII
  - Suppose there are 4: **Trek**, **Specialized**, **Cervelo**, **Serrota**

Information Theory: Encoding

- Could send out the names of the manufacturers in binary coded ASCII
  - Suppose there are 4: **Trek**, **Specialized**, **Cervelo**, **Serrota**

Information Theory: Encoding

- Could send out the names of the manufacturers in binary coded ASCII
  - Suppose there are 4: Trek, Specialized, Cervelo, Serrota
- Inefficient… since there's just 4, we could **encode** them
  - # of bits: 2 per communication

Information Theory: Encoding

- Could send out the names of the manufacturers in binary coded ASCII
  - Suppose there are 4: **Trek**, **Specialized**, **Cervelo**, **Serrota**
- Inefficient… since there's just 4, we could **encode** them
  - # of bits: 2 per communication

type	code
Trek	11
Specialized	10
Cervelo	01
Serrota	00

Information Theory: Encoding

Information Theory: Encoding

- Now, some bikes are rarer than others…
  - **Cervelo** is a rarer specialty bike.
  - We could **save some bits**… make more popular messages fewer bits, rarer ones more bits
  - Note: this is **on average**

Information Theory: Encoding

- Now, some bikes are rarer than others…
  - **Cervelo** is a rarer specialty bike.
  - We could **save some bits**… make more popular messages fewer bits, rarer ones more bits
  - Note: this is **on average**

- Expected # bits: **1.75**

Information Theory: Encoding

- Now, some bikes are rarer than others…
  - **Cervelo** is a rarer specialty bike.
  - We could **save some bits**… make more popular messages fewer bits, rarer ones more bits
  - Note: this is **on average**

- Expected # bits: **1.75**

Type/probability	# bits	code
$P(\text{Trek}) = 0.5$	1	1
$P(\text{Specialized}) = 0.25$	2	01
$P(\text{Cervelo}) = 0.125$	3	001
$P(\text{Serrota}) = 0.125$	3	000

Information Theory: Encoding

- Now, some bikes are rarer than others…
  - **Cervelo** is a rarer specialty bike.
  - We could **save some bits**… make more popular messages fewer bits, rarer ones more bits
  - Note: this is **on average**

- Expected # bits: **1.75**

$$-\sum_{y \in \mathcal{Y}} P(y) \log_2 P(y)$$

Type/probability	# bits	code
$P(\text{Trek}) = 0.5$	1	1
$P(\text{Specialized}) = 0.25$	2	01
$P(\text{Cervelo}) = 0.125$	3	001
$P(\text{Serrota}) = 0.125$	3	000

Information Theory: Entropy

Information Theory: Entropy

- Measure of uncertainty for random variables/distributions

Information Theory: Entropy

- Measure of uncertainty for random variables/distributions

- **Expected number of bits** required to communicate the value of the variable

Information Theory: Entropy

- Measure of uncertainty for random variables/distributions

- **Expected number of bits** required to communicate the value of the variable

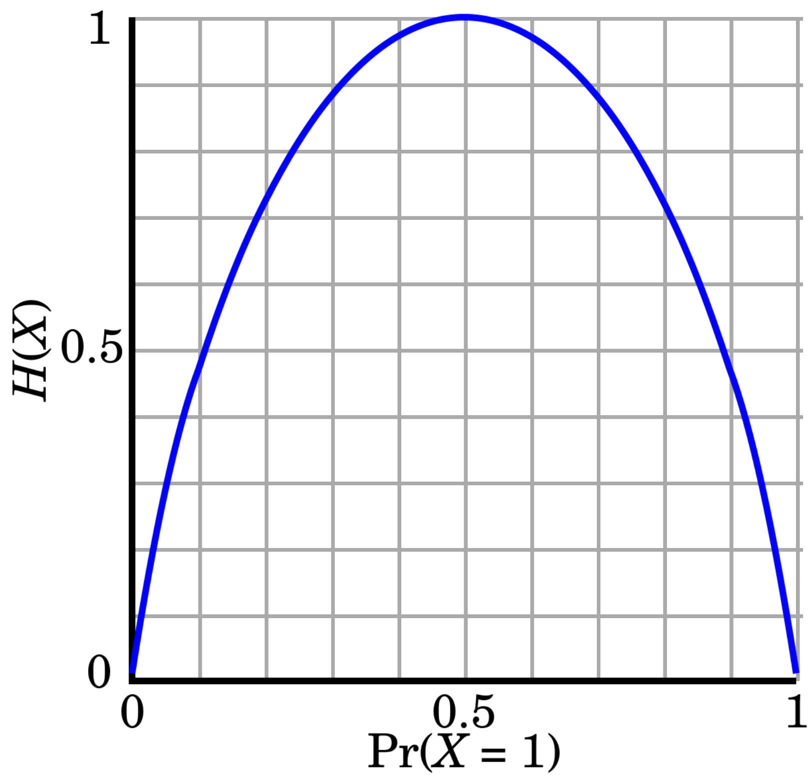
$$H(Y) = -\sum_{y \in \mathcal{Y}} P(y) \log_2 P(y)$$

Information Theory: Entropy

- Measure of uncertainty for random variables/distributions

- **Expected number of bits** required to communicate the value of the variable

$$H(Y) = -\sum_{y \in \mathcal{Y}} P(y) \log_2 P(y)$$



Information Theory: Conditional Entropy

$$H(Y|X) = \sum_{x \in \mathcal{X}} \Pr(X = x) H(Y|X = x)$$

Information Theory: Conditional Entropy

- Suppose we know *X*. **CE**: how much uncertainty left in *Y on average after X is known*?

$$H(Y|X) = \sum_{x \in \mathcal{X}} \Pr(X = x) H(Y|X = x)$$

Information Theory: Conditional Entropy

- Suppose we know *X*. **CE**: how much uncertainty left in *Y on average after X is known*?

$$H(Y|X) = \sum_{x \in \mathcal{X}} \Pr(X = x) H(Y|X = x)$$

- Here,

Information Theory: Conditional Entropy

- Suppose we know *X*. **CE**: how much uncertainty left in *Y on average after X is known*?

$$H(Y|X) = \sum_{x \in \mathcal{X}} \Pr(X = x) H(Y|X = x)$$

- Here,

$$H(Y|X=x) = -\sum_{y \in \mathcal{Y}} P(Y=y|X=x) \log_2 P(Y=y|X=x)$$

Information Theory: Conditional Entropy

- Suppose we know *X*. **CE**: how much uncertainty left in *Y on average after X is known*?

$$H(Y|X) = \sum_{x \in \mathcal{X}} \Pr(X = x) H(Y|X = x)$$

- Here,

$$H(Y|X=x) = -\sum_{y \in \mathcal{Y}} P(Y=y|X=x) \log_2 P(Y=y|X=x)$$

- What is it if Y=X?

Information Theory: Conditional Entropy

- Suppose we know *X*. **CE**: how much uncertainty left in *Y on average after X is known*?

$$H(Y|X) = \sum_{x \in \mathcal{X}} \Pr(X = x) H(Y|X = x)$$

- Here,

$$H(Y|X = x) = -\sum_{y \in \mathcal{Y}} P(Y = y|X = x) \log_2 P(Y = y|X = x)$$

- What is it if Y=X?
- What if Y is **independent** of X?

Information Theory: Conditional Entropy

Information Theory: Conditional Entropy

- Example. *Y* is still the bike maker, *X* is color.

Information Theory: Conditional Entropy

- Example. Y is still the bike maker, X is color.

Y=Type/X=Color	Black	White
Trek	0.25	0.25
Specialized	0.125	0.125
Cervelo	0.125	0
Serrota	0	0.125

Information Theory: Conditional Entropy

- Example. *Y* is still the bike maker, *X* is color.

Y=Type/X=Color	Black	White
Trek	0.25	0.25
Specialized	0.125	0.125
Cervelo	0.125	0
Serrota	0	0.125

Information Theory: Conditional Entropy

- Example. *Y* is still the bike maker, *X* is color.

Y=Type/X=Color	Black	White
Trek	0.25	0.25
Specialized	0.125	0.125
Cervelo	0.125	0
Serrota	0	0.125

$H(Y|X=black) = -0.5 \log(0.5) - 0.25 \log(0.25) - 0.25 \log(0.25) - 0 = 1.5$

$H(Y|X=white) = -0.5 \log(0.5) - 0.25 \log(0.25) - 0 - 0.25 \log(0.25) = 1.5$

$H(Y|X) = 0.5 * H(Y|X=black) + 0.5 * H(Y|X=white) = 1.5$

Information Theory: Mutual Information

Information Theory: Mutual Information

- Similar comparison between R.V.s:

Information Theory: Mutual Information

- Similar comparison between R.V.s:

$$I(Y;X) = H(Y) - H(Y|X)$$

Information Theory: Mutual Information

- Similar comparison between R.V.s:

$$I(Y;X) = H(Y) - H(Y|X)$$

Interpretation:

Information Theory: Mutual Information

- Similar comparison between R.V.s:

$$I(Y;X) = H(Y) - H(Y|X)$$

Interpretation:

- How much uncertainty of Y that X can reduce.

Information Theory: Mutual Information

- Similar comparison between R.V.s:

$$I(Y;X) = H(Y) - H(Y|X)$$

Interpretation:

- How much uncertainty of Y that X can reduce.
- Or, how much information about Y can you glean by knowing X?

Information Theory: Mutual Information

- Similar comparison between R.V.s:

$$I(Y;X) = H(Y) - H(Y|X)$$

Interpretation:

- How much uncertainty of Y that X can reduce.
- Or, how much information about Y can you glean by knowing X?

Y=Type/X=Color	Black	White
Trek	0.25	0.25
Specialized	0.125	0.125
Cervelo	0.125	0
Serrota	0	0.125

Information Theory: Mutual Information

- Similar comparison between R.V.s:

$$I(Y;X) = H(Y) - H(Y|X)$$

Interpretation:

- How much uncertainty of Y that X can reduce.
- Or, how much information about Y can you glean by knowing X?

Y=Type/X=Color	Black	White
Trek	0.25	0.25
Specialized	0.125	0.125
Cervelo	0.125	0
Serrota	0	0.125

I(Y:X) = H(Y) – H(Y|X) = 1.75 – 1.5 = 0.25

DT Learning: Back to Splits

DT Learning: Back to Splits

Want to choose split S that maximizes

DT Learning: Back to Splits

Want to choose split S that maximizes

$$\text{InfoGain}(D, S) = H_D(Y) - H_D(Y|S)$$

DT Learning: Back to Splits

Want to choose split S that maximizes

$$\text{InfoGain}(D, S) = H_D(Y) - H_D(Y|S)$$

ie, mutual information.

DT Learning: Back to Splits

Want to choose split S that maximizes

$$\text{InfoGain}(D, S) = H_D(Y) - H_D(Y|S)$$

ie, mutual information.

- Note: D denotes that this is the **empirical** entropy
  - We don't know the real distribution of *Y*, just have our dataset

DT Learning: Back to Splits

Want to choose split S that maximizes

$$\text{InfoGain}(D, S) = H_D(Y) - H_D(Y|S)$$

ie, mutual information.

- Note: D denotes that this is the **empirical** entropy
  - We don't know the real distribution of *Y*, just have our dataset

- Equivalent to maximally reducing the entropy of Y conditioned on a split S

DT Learning: InfoGain Example

Simple binary classification (**play tennis**?) with 4 features.

DT Learning: InfoGain Example

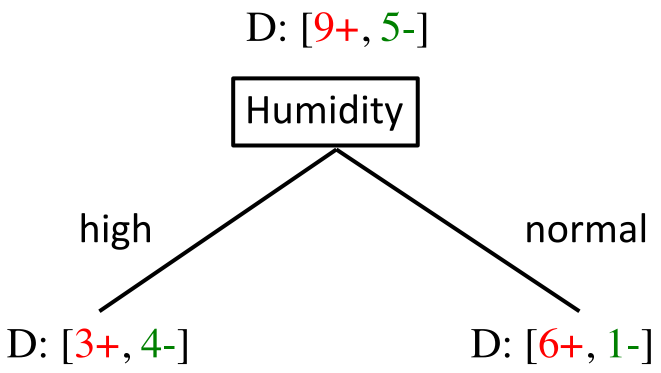
Simple binary classification (**play tennis**?) with 4 features.

PlayTennis: training examples

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

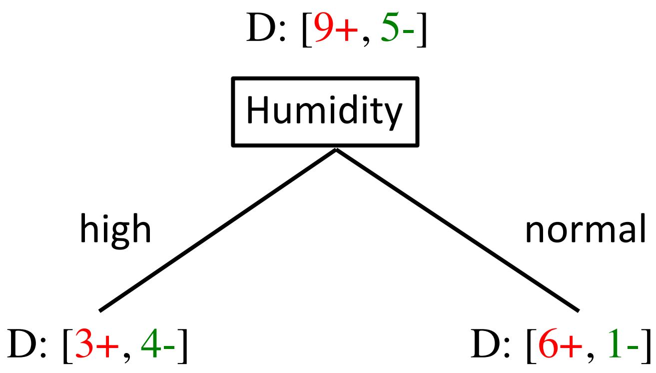
DT Learning: InfoGain For One Split

- What is the information gain of splitting on Humidity?



DT Learning: InfoGain For One Split

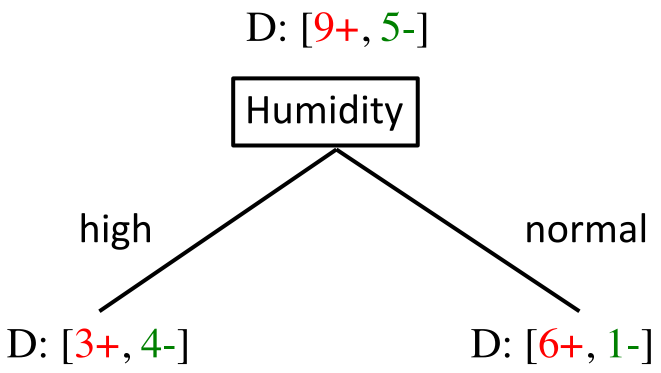
- What is the information gain of splitting on Humidity?



$$H_D(Y) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

DT Learning: InfoGain For One Split

- What is the information gain of splitting on Humidity?

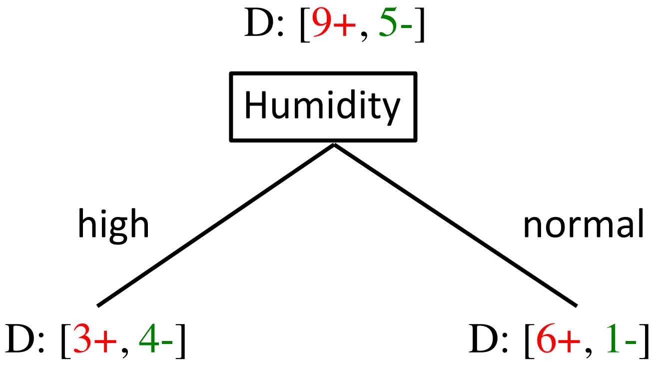


$$H_D(Y) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

$$H_D(Y \mid \text{high}) = -\frac{3}{7}\log_2\left(\frac{3}{7}\right) - \frac{4}{7}\log_2\left(\frac{4}{7}\right)$$
$$= 0.985$$

DT Learning: InfoGain For One Split

- What is the information gain of splitting on Humidity?



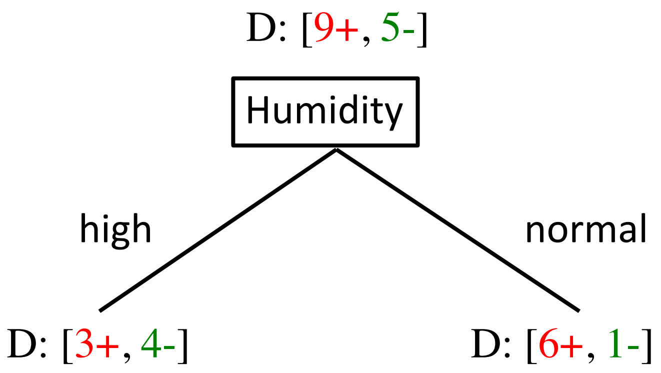
$$H_D(Y) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

$$H_D(Y \mid \text{high}) = -\frac{3}{7}\log_2\left(\frac{3}{7}\right) - \frac{4}{7}\log_2\left(\frac{4}{7}\right)$$
$$= 0.985$$

$$H_D(Y \mid \text{normal}) = -\frac{6}{7}\log_2\left(\frac{6}{7}\right) - \frac{1}{7}\log_2\left(\frac{1}{7}\right)$$
$$= 0.592$$

DT Learning: InfoGain For One Split

- What is the information gain of splitting on Humidity?



$$H_D(Y) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

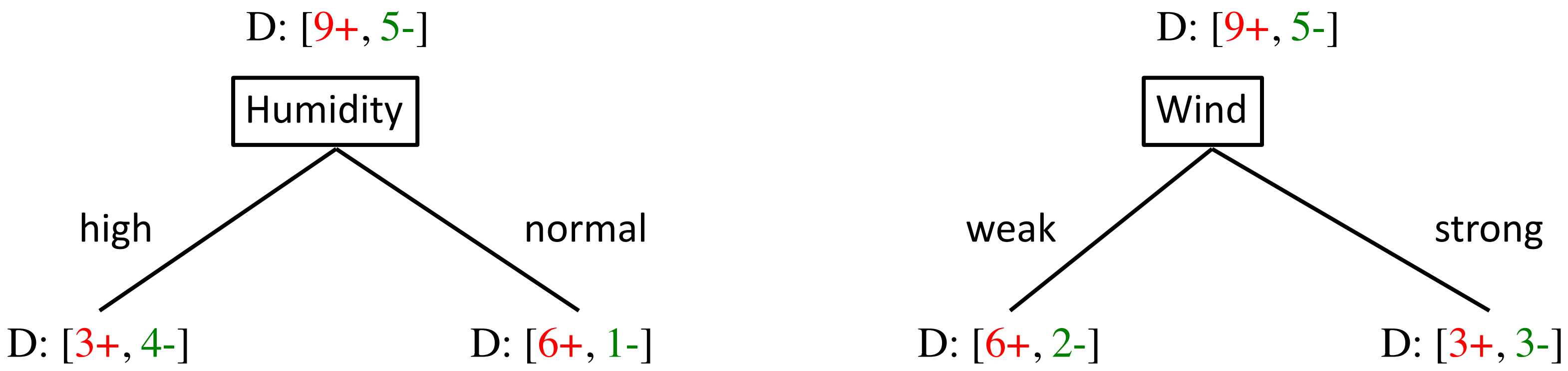
$$H_D(Y \mid \text{high}) = -\frac{3}{7}\log_2\left(\frac{3}{7}\right) - \frac{4}{7}\log_2\left(\frac{4}{7}\right) = 0.985$$

$$H_D(Y \mid \text{normal}) = -\frac{6}{7}\log_2\left(\frac{6}{7}\right) - \frac{1}{7}\log_2\left(\frac{1}{7}\right) = 0.592$$

$$\begin{aligned}\text{InfoGain}(D, \text{Humidity}) &= H_D(Y) - H_D(Y \mid \text{Humidity}) \\ &= 0.940 - \left[\frac{7}{14}(0.985) + \frac{7}{14}(0.592)\right] \\ &= 0.151\end{aligned}$$

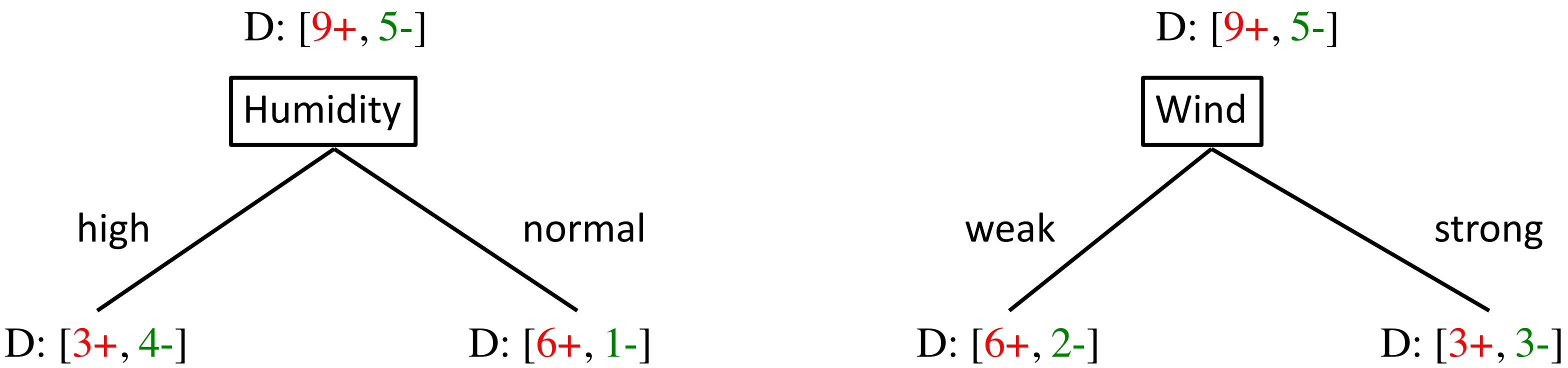
DT Learning: Comparing Split InfoGains

- Is it better to split on **Humidity** or **Wind**?



DT Learning: Comparing Split InfoGains

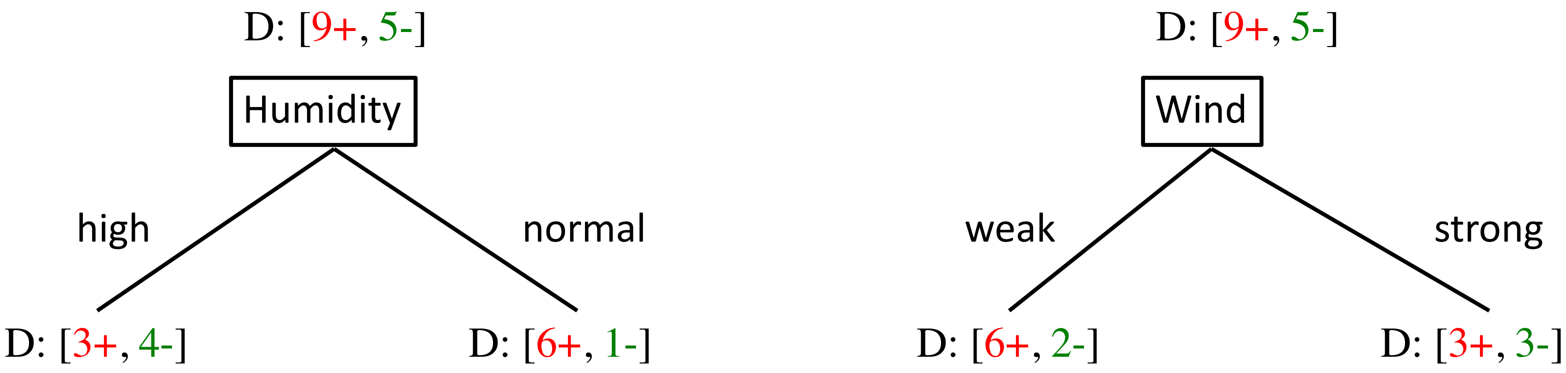
- Is it better to split on **Humidity** or **Wind**?



$$H_D(Y \mid \text{weak}) = 0.811$$

DT Learning: Comparing Split InfoGains

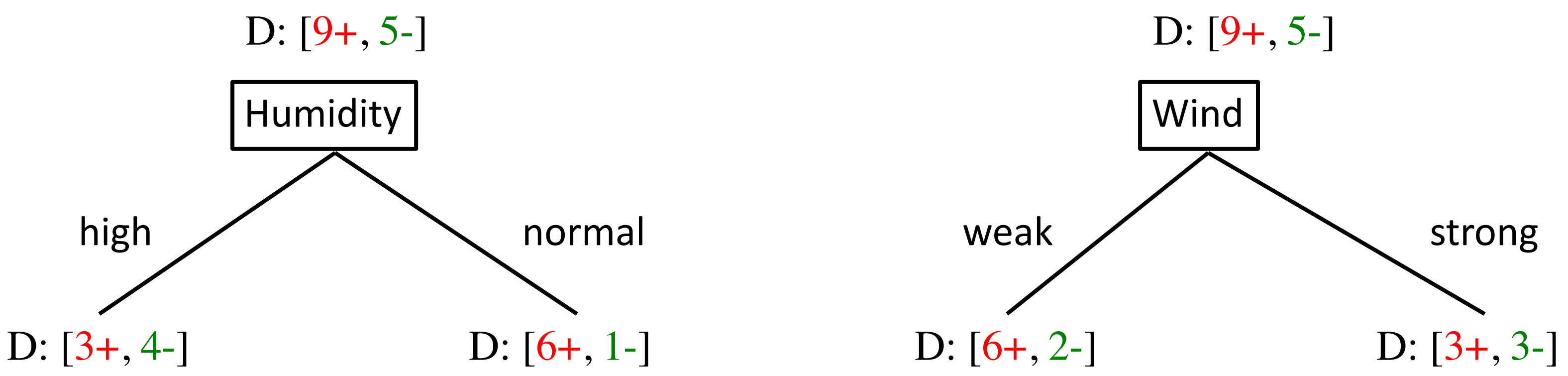
- Is it better to split on **Humidity** or **Wind**?



$H_D(Y \mid \text{weak}) = 0.811$ $\qquad$ $H_D(Y \mid \text{strong}) = 1.0$

DT Learning: Comparing Split InfoGains

- Is it better to split on **Humidity** or **Wind**?



$H_D(Y \mid \text{weak}) = 0.811$ $\quad$ $H_D(Y \mid \text{strong}) = 1.0$

✔

$$\begin{aligned}\text{InfoGain}(D, \text{Humidity}) &= 0.940 - \left[\frac{7}{14}(0.985) + \frac{7}{14}(0.592)\right] \\ &= 0.151\end{aligned}$$

$$\begin{aligned}\text{InfoGain}(D, \text{Wind}) &= 0.940 - \left[\frac{8}{14}(0.811) + \frac{6}{14}(1.0)\right] \\ &= 0.048\end{aligned}$$

DT Learning: InfoGain Limitations

- InfoGain is biased towards tests with many outcomes

DT Learning: InfoGain Limitations

- InfoGain is biased towards tests with many outcomes
  - Splitting on it results in many branches, each of which is “pure” (has instances of only one class)

DT Learning: InfoGain Limitations

- InfoGain is biased towards tests with many outcomes
  - Splitting on it results in many branches, each of which is "pure" (has instances of only one class)
  - In the extreme: A feature that uniquely identifies each instance

DT Learning: InfoGain Limitations

- InfoGain is biased towards tests with many outcomes
  - Splitting on it results in many branches, each of which is “pure” (has instances of only one class)
  - In the extreme: A feature that uniquely identifies each instance
  - **Maximal** information gain!

DT Learning: InfoGain Limitations

- InfoGain is biased towards tests with many outcomes
  - Splitting on it results in many branches, each of which is “pure” (has instances of only one class)
  - In the extreme: A feature that uniquely identifies each instance
  - **Maximal** information gain!

- Use **GainRatio**: normalize information gain by entropy

DT Learning: InfoGain Limitations

- InfoGain is biased towards tests with many outcomes
  - Splitting on it results in many branches, each of which is “pure” (has instances of only one class)
  - In the extreme: A feature that uniquely identifies each instance
  - **Maximal** information gain!

- Use **GainRatio**: normalize information gain by entropy

$$\text{GainRatio}(D, S) = \frac{\text{InfoGain}(D,S)}{H_D(S)} = \frac{H_D(Y) - H_D(Y|S)}{H_D(S)}$$

Homework: What is a good stopping criteria?

- **Learning Algorithm**:

Homework: What is a good stopping criteria?

- **Learning Algorithm**:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Homework: What is a good stopping criteria?

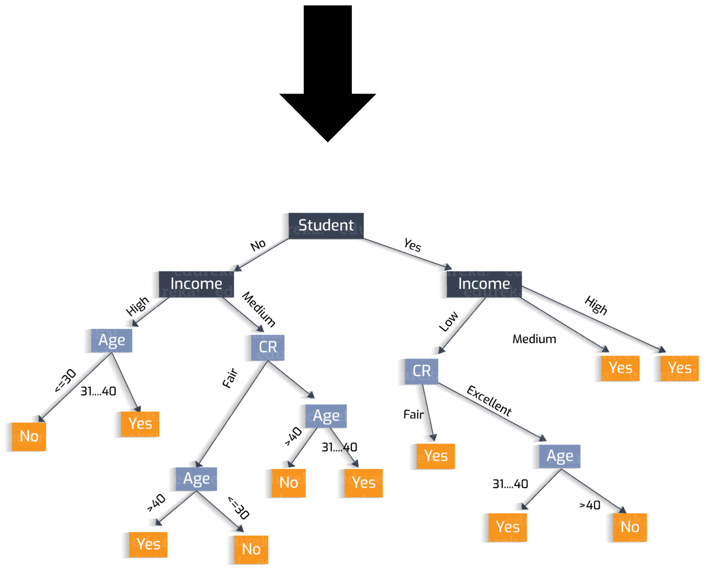
- **Learning Algorithm**:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Homework: What is a good stopping criteria?

- **Learning Algorithm**:

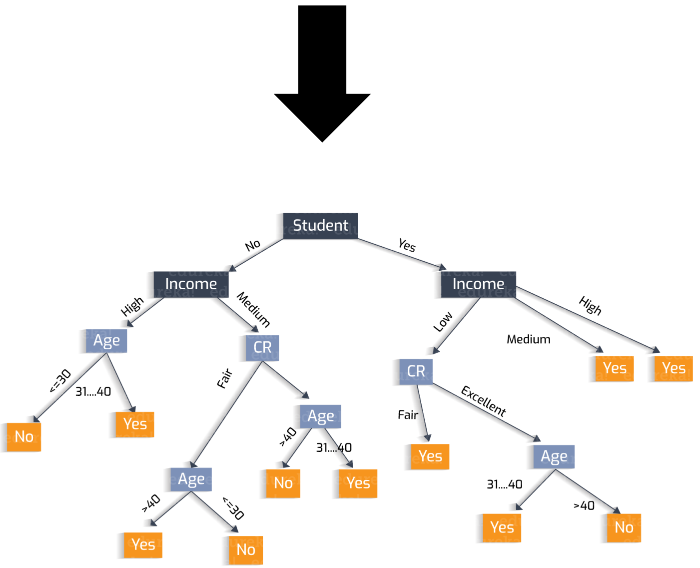
$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$



Homework: What is a good stopping criteria?

- **Learning Algorithm:**

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$



MakeSubtree(set of training instances D)

C = **DetermineCandidateSplits**(D)

if **stopping criteria** is met

make a leaf node N

determine class label for N

else

make an internal node N

S = **FindBestSplit**(D, C)

for each group k of S

D_k = subset of training data in group k

k^{th} child of N = MakeSubtree(D_k)

return subtree rooted at N

Inductive Bias

- Recall: ***Inductive bias***: assumptions a learner uses to predict y_i for a previously unseen instance $\boldsymbol{x}_i$
- Two components
  - *hypothesis space bias*: determines the models that can be represented
  - *preference bias*: specifies a preference ordering within the space of models

learner	**hypothesis space bias**	**preference bias**
Decision trees	trees with single-feature, axis-parallel splits	small trees identified by greedy search
k-NN	Decomposition of space determined by nearest neighbors	Instances in neighborhood belong to same class

Q3-1: Which of the following statements are True?

1. In a decision tree, once you split using one feature, you cannot split again using the same feature.
2. We should split along all features to create a decision tree.
3. We should keep splitting the tree until there is only one data point left at each leaf node.

Q3-1: Which of the following statements are True?

1. In a decision tree, once you split using one feature, you cannot split again using the same feature.
2. We should split along all features to create a decision tree.
3. We should keep splitting the tree until there is only one data point left at each leaf node.

They are all false!

Today’s Learning Outcomes

- **After today’s lecture:**
  - You will be able to explain how the k-nearest neighbor’s algorithm classifies unseen instances.
  - You will be able to explain the concept of an inductive bias.
  - You will be able to explain how a decision tree classifies instances.

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov