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

• 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 \( f : \mathcal{X} \rightarrow \mathcal{Y} \)
- 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
- Unknown target function
- Set of models (a.k.a. hypotheses):

Given

\[ \mathcal{X} \]

\[ f : \mathcal{X} \rightarrow \mathcal{Y} \]

\[ \mathcal{H} = \{ h | h : \mathcal{X} \rightarrow \mathcal{Y} \} \]
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

• 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): \( \mathcal{H} = \{ h \mid h : \mathcal{X} \rightarrow \mathcal{Y} \} \)

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 \( f : \mathcal{X} \rightarrow \mathcal{Y} \)

• 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 \( y^{(i)} \approx f(x^{(i)}) \)

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

  $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$
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 \( y^{(i)} \approx f(x^{(i)}) \)
\[
(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})
\]
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): \( \mathcal{H} = \{ h \mid h : \mathcal{X} \rightarrow \mathcal{Y} \} \)

Given

- Training set of instances for unknown target function, where \( y^{(i)} \approx f(x^{(i)}) \)
  \( (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)}) \)

\begin{align*}
\text{safe} & \quad \text{poisonous} & \text{safe}
\end{align*}
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

$X, Y, H$

• 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
      \[ \mathcal{Y} = \{-1, +1\} \]
    - Continuous
Supervised Learning: Objects

Three types of sets
- Input space, output space, hypothesis class

• Examples:
  - Input space: feature vectors
    \( \mathcal{X} \subseteq \mathbb{R}^d \)
  - Output space:
    - Discrete/Nominal
      \( \mathcal{Y} = \{-1, +1\} \)
    - Continuous
      \( \mathcal{Y} \subseteq \mathbb{R} \)
Supervised Learning: Objects

Three types of sets
- Input space, output space, hypothesis class

• Examples:
  - Input space: feature vectors \( \mathcal{X} \subseteq \mathbb{R}^d \)

  - Output space:
    - Discrete/Nominal
      \( \mathcal{Y} = \{-1, +1\} \)
    - Continuous
      \( \mathcal{Y} \subseteq \mathbb{R} \)

safe       poisonous

\[ 13.23^\circ \]
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:

\[ h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d \]
Hypothesis class

• Pick specific class of models. Ex: linear models:

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

\[ f^{(0)}(x) = x \]

\[ f^{(k)}(x) = \sigma(W_k^T f^{(k-1)}(x)) \]
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.

$$\hat{f} = \arg \min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}, y^{(i)}))$$
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**
Supervised Learning: Training & Generalization

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

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

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

- Recall: we want to generalize.

Hypothesis Class

Model prediction

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.

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

- 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
1-Nearest Neighbors: Algorithm

Training/learning: given

- safe
1-Nearest Neighbors: Algorithm

Training/learning: given

Prediction: for $x$, find nearest training point $x^{(i)}$
1-Nearest Neighbors: Algorithm

Training/learning: given

Prediction: for $x$, find nearest training point $x^{(j)}$
Return $y^{(j)}$
1-Nearest Neighbors: Algorithm

**Training/learning:** given

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

Return $y^{(i)}$
1-Nearest Neighbors: Algorithm

Training/learning: given

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

Return $y^{(i)}$
1-Nearest Neighbors: Algorithm

Training/learning: given

Prediction: for \( x \), find nearest training point \( x^{(i)} \)
Return \( y^{(i)} \) poisonous
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 “Voronoï 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

\[ \hat{y} = \arg \max_{y \in \mathcal{Y}} \sum_{i=1}^{k} 1(y = y^{(i)}) \]
**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

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

• 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: \( d([\text{‘a’, ‘b’, ‘c’}],[\text{‘d’, ‘b’, ‘e’}]) = 2 \)

\[
d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} 1\{x_a^{(i)} \neq x_a^{(j)}\}
\]
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: $d([\text{‘a’, ‘b’, ‘c’}], [\text{‘d’, ‘b’, ‘e’}]) = 2$

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

**Continuous features:**

- **Euclidean distance:**

  Ex: $d([0, 0], [4, 4]) = \sqrt{32}$
**k-Nearest Neighbors: Distances**

**Discrete features:** Hamming distance

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

\[
d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} 1\{x^{(i)}_a \neq x^{(j)}_a\}
\]

**Continuous features:**

- Euclidean distance:

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

  \[
d(x^{(i)}, x^{(j)}) = \left(\sum_{a=1}^{d} (x^{(i)}_a - x^{(j)}_a)^2\right)^{\frac{1}{2}}
\]
**k-Nearest Neighbors: Distances**

**Discrete features:** Hamming distance

Ex: \(d([\text{a}, \text{b}, \text{c}], [\text{d}, \text{b}, \text{e}]) = 2\)

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

**Continuous features:**

- **Euclidean distance:**
  
  Ex: \(d([0, 0], [4, 4]) = 32\)

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

- **L1 (Manhattan) dist.:**
k-Nearest Neighbors: Distances

Discrete features: Hamming distance

Ex: \( d([\text{`a', `b'}, `c'], [\text{`d', `b'}, `e']) = 2 \)

\[
d_H(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} 1\{x^{(i)}_a \neq x^{(j)}_a\}
\]

Continuous features:

• Euclidean distance:

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

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

• L1 (Manhattan) dist.:

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

\[
d(x^{(i)}, x^{(j)}) = \sum_{a=1}^{d} |x^{(i)}_a - x^{(j)}_a|
\]
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)

\[
\mu_a = \frac{1}{n} \sum_{i=1}^{n} x_a^{(i)} \\
\sigma_a = \left( \frac{1}{n} \sum_{i=1}^{n} (x_a^{(i)} - \mu_i)^2 \right)^{\frac{1}{2}}
\]
k-Nearest Neighbors: Standardization

Typical in data science applications. Recipe:
• Compute empirical mean/stddev for a feature (in train set)

\[ \mu_a = \frac{1}{n} \sum_{i=1}^{n} x_{a}^{(i)} \]
\[ \sigma_a = \left( \frac{1}{n} \sum_{i=1}^{n} (x_{a}^{(i)} - \mu_i)^2 \right)^{\frac{1}{2}} \]

• Standardize features:
**k-Nearest Neighbors**: Standardization

Typical in data science applications. Recipe:

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

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

- 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)

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

• 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)

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

• 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)

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

\[
\sigma_a = \left( \frac{1}{n} \sum_{i=1}^{n} (x_a^{(i)} - \mu_i)^2 \right)^{\frac{1}{2}}
\]

- 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:

\[ \hat{y} \leftarrow \text{arg max}_{v \in Y} \sum_{i=1}^{k} \frac{1}{d(x, x^{(i)})^2} \delta(v, y^{(i)}) \]

- 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 $x_i$. 
Inductive Bias

- **Inductive bias**: assumptions a learner uses to predict $y_i$ for a previously unseen instance $x_i$

- 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 \( x_i \)

- Two components (mostly)
  - *hypothesis space bias*: determines the models that can be represented
  - *preference bias*: specifies a preference ordering within the space of models

<table>
<thead>
<tr>
<th>learner</th>
<th>hypothesis space bias</th>
<th>preference bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )-NN</td>
<td>Decomposition of space determined by nearest neighbors</td>
<td>Instances in neighborhood belong to same class</td>
</tr>
</tbody>
</table>
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)?

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>+</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

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. \( w_i = d(p, x_i)^{\frac{1}{2}} \)
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. \( w_i = d(p, x_i)^{\frac{1}{2}} \)  
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

- thal
  - normal
  - fixed_defect
  - reversible_defect

- #_major_vessels > 0
  - present
  - false

- chest_pain_type
  - absent
  - 1
  - 2
  - 3
  - 4

- 1: absent
- 2: absent
- 3: absent
- 4: present
**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 $X_1 \cdots X_5$ 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?

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

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

  $$Y = X_2 \lor X_5$$
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?

\[ Y = X_2 X_5 \] (i.e., $Y = X_2 \land X_5$)

\[ Y = X_2 \lor X_5 \]

\[ Y = X_2 X_5 \lor X_3 \neg X_1 \]
Decision Trees: Textual Description

thal

#_major_vessels > 0

present

normal

fixed_defect

true

false

present

absent
Decision Trees: Textual Description

#_major_vessels > 0

- present
  - true
    - present
  - false
    - absent

thal
  - normal
  - fixed_defect
### Decision Trees: Textual Description

- **thal**
  - **normal**
    - **#_major_vessels > 0**
      - true: present
      - false: absent
  - **fixed_defect**: present

**Formal Representation**

- \( \text{thal} = \text{normal} \)
  - \([\text{#_major_vessels > 0}] = \text{true} \): present
  - \([\text{#_major_vessels > 0}] = \text{false} \): absent
- \( \text{thal} = \text{fixed_defect} \): present
Decision Trees: Mushrooms Example

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:

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

\[\text{MakeSubtree}(\text{set of training instances } D)\]

\[C = \text{DetermineCandidateSplits}(D)\]

if stopping criteria is met

make a leaf node \( N \)

determine class label for \( N \)

else

make an internal node \( N \)

\[S = \text{FindBestSplit}(D, C)\]

for each group \( k \) of \( S \)

\[D_k = \text{subset of training data in group } k\]

\[k^{th} \text{ child of } N = \text{MakeSubtree}(D_k)\]

return subtree rooted at \( N \)
Decision Trees: Learning

• Learning Algorithm:

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

\[
\text{MakeSubtree} (\text{set of training instances } D)
\]

\[
C = \text{DetermineCandidateSplits}(D)
\]

if stopping criteria is met

make a leaf node \( N \)

determine class label for \( N \)

else

make an internal node \( N \)

\[
S = \text{FindBestSplit}(D, C)
\]

for each group \( k \) of \( S \)

\[
D_k = \text{subset of training data in group } k
\]

\[
k^{th} \text{ child of } N = \text{MakeSubtree}(D_k)
\]

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
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

```
thal
  /   
normal fixed_defect reversible_defect
```
1. DT Learning: Candidate Splits

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

- Splits on nominal features have one branch per value
  
  ![Tree Diagram]

- Splits on numeric features use a threshold/interval
  
  ![Tree Diagram]
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

![Decision Tree Diagram]

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

![Diagram of weight distribution and decision tree split at weight ≤ 35]
// 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.

```
thal
  normal
  reversible_defect ∨ fixed_defect

color
  red ∨ blue
  green ∨ yellow
```
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:**

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

  
  \[
  \text{MakeSubtree(} \text{set of training instances } D) \]

  \[
  C = \text{DetermineCandidateSplits}(D)
  \]

  if stopping criteria is met

  make a leaf node \( N \)

  determine class label for \( N \)

  else

  make an internal node \( N \)

  \[
  S = \text{FindBestSplit}(D, C)
  \]

  for each group \( k \) of \( S \)

  \[
  D_k = \text{subset of training data in group } k
  \]

  \[
  \text{ } k^{th} \text{ child of } N = \text{MakeSubtree}(D_k)
  \]

  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 do 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?
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**

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**
  

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

![Diagram of a decision tree]

[Image of a decision tree diagram showing various decision nodes and outcomes. The tree includes decision criteria such as income, student status, and age, with outcomes for each decision path.]
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

<table>
<thead>
<tr>
<th>type</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trek</td>
<td>11</td>
</tr>
<tr>
<td>Specialized</td>
<td>10</td>
</tr>
<tr>
<td>Cervelo</td>
<td>01</td>
</tr>
<tr>
<td>Serrota</td>
<td>00</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Type/probability</th>
<th># bits</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P(\text{Trek}) = 0.5)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(P(\text{Specialized}) = 0.25)</td>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>(P(\text{Cervelo}) = 0.125)</td>
<td>3</td>
<td>001</td>
</tr>
<tr>
<td>(P(\text{Serrota}) = 0.125)</td>
<td>3</td>
<td>000</td>
</tr>
</tbody>
</table>
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**

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

<table>
<thead>
<tr>
<th>Type/probability</th>
<th># bits</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P(\text{Trek}) = 0.5)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(P(\text{Specialized}) = 0.25)</td>
<td>2</td>
<td>01</td>
</tr>
<tr>
<td>(P(\text{Cervelo}) = 0.125)</td>
<td>3</td>
<td>001</td>
</tr>
<tr>
<td>(P(\text{Serrota}) = 0.125)</td>
<td>3</td>
<td>000</td>
</tr>
</tbody>
</table>
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?

\[
H(Y \mid X) = \sum_{x \in \mathcal{X}} \Pr(X = x)H(Y \mid X = x)
\]
Information Theory: Conditional Entropy

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

\[
H(Y \mid X) = \sum_{x \in \mathcal{X}} \Pr(X = x)H(Y \mid X = x)
\]

• Here,
Information Theory: Conditional Entropy

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

\[ H(Y \mid X) = \sum_{x \in \mathcal{X}} \Pr(X = x) H(Y \mid X = x) \]

• Here,

\[ H(Y \mid X = x) = - \sum_{y \in \mathcal{Y}} P(Y = y \mid X = x) \log_2 P(Y = y \mid X = x) \]
Information Theory: Conditional Entropy

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

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

• Here,

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

• 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?

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

• Here,

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

• 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.

<table>
<thead>
<tr>
<th>Y=Type/X=Color</th>
<th>Black</th>
<th>White</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trek</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Specialized</td>
<td>0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>Cervelo</td>
<td>0.125</td>
<td>0</td>
</tr>
<tr>
<td>Serrota</td>
<td>0</td>
<td>0.125</td>
</tr>
</tbody>
</table>
Information Theory: Conditional Entropy

• Example. $Y$ is still the bike maker, $X$ is color.

<table>
<thead>
<tr>
<th>Y=Type / X=Color</th>
<th>Black</th>
<th>White</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trek</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Specialized</td>
<td>0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>Cervelo</td>
<td>0.125</td>
<td>0</td>
</tr>
<tr>
<td>Serrota</td>
<td>0</td>
<td>0.125</td>
</tr>
</tbody>
</table>
Information Theory: Conditional Entropy

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

<table>
<thead>
<tr>
<th>Y=Type/X=Color</th>
<th>Black</th>
<th>White</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trek</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Specialized</td>
<td>0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>Cervelo</td>
<td>0.125</td>
<td>0</td>
</tr>
<tr>
<td>Serrota</td>
<td>0</td>
<td>0.125</td>
</tr>
</tbody>
</table>

\[
H(Y|X=\text{black}) = -0.5 \log(0.5) - 0.25 \log(0.25) - 0.25 \log(0.25) - 0 = 1.5
\]
\[
H(Y|X=\text{white}) = -0.5 \log(0.5) - 0.25 \log(0.25) - 0 - 0.25 \log(0.25) = 1.5
\]
\[
H(Y|X) = 0.5 \times H(Y|X=\text{black}) + 0.5 \times H(Y|X=\text{white}) = 1.5
\]
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?

<table>
<thead>
<tr>
<th>Y=Type/X=Color</th>
<th>Black</th>
<th>White</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trek</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Specialized</td>
<td>0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>Cervelo</td>
<td>0.125</td>
<td>0</td>
</tr>
<tr>
<td>Serrota</td>
<td>0</td>
<td>0.125</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>Y=Type/X=Color</th>
<th>Black</th>
<th>White</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trek</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>Specialized</td>
<td>0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>Cervelo</td>
<td>0.125</td>
<td>0.0</td>
</tr>
<tr>
<td>Serrota</td>
<td>0</td>
<td>0.125</td>
</tr>
</tbody>
</table>

\[ I(Y:X) = H(Y) - H(Y|X) = 1.75 - 1.5 = 0.25 \]
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 \textbf{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.

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Wind</th>
<th>PlayTennis</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Overcast</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>
DT Learning: InfoGain For One Split

• What is the information gain of splitting on Humidity?

```
Humidity

high
D: [3+, 4-]

normal
D: [6+, 1-]
```

D: [9+, 5-]
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?

**Humidity**

<table>
<thead>
<tr>
<th>Humidity</th>
<th>D: [9+, 5-]</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>D: [3+, 4-]</td>
</tr>
<tr>
<td>normal</td>
<td>D: [6+, 1-]</td>
</tr>
</tbody>
</table>

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

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

\[
H_D (Y \mid \text{normal}) = - \frac{6}{7} \log_2 \left( \frac{6}{7} \right) - \frac{1}{7} \log_2 \left( \frac{1}{7} \right) = 0.592
\]
DT Learning: InfoGain For One Split

• What is the information gain of splitting on Humidity?

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

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

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

\[
\text{InfoGain}(D, \text{Humidity}) = H_D(Y) - H_D(Y \mid \text{Humidity})
\]

\[
= 0.940 - \left[ \frac{7}{14} (0.985) + \frac{7}{14} (0.592) \right]
\]

\[
= 0.151
\]
DT Learning: Comparing Split InfoGains

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

```
Humidity
(high)
D: [3+, 4-]
D: [6+, 1-]

Humidity
(normal)
D: [9+, 5-]

Wind
(weak)
D: [6+, 2-]
D: [3+, 3-]

Wind
(strong)
D: [9+, 5-]
```
DT Learning: Comparing Split InfoGains

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

```
Humidity
  high   normal
  D: [3+, 4-]  D: [6+, 1-]

Wind
  weak   strong
  D: [6+, 2-]  D: [3+, 3-]
```

\[ H_D(Y \mid \text{weak}) = 0.811 \]
DT Learning: Comparing Split InfoGains

• Is it better to split on Humidity or Wind?

- Humidity:
  - high
    - D: [3+, 4-]
  - normal
    - D: [6+, 1-]

- Wind:
  - weak
    - D: [6+, 2-]
  - strong
    - D: [3+, 3-]

\[ H_D(Y|\text{weak}) = 0.811 \quad H_D(Y|\text{strong}) = 1.0 \]
DT Learning: Comparing Split InfoGains

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

![Decision Tree Diagram]

\[
\begin{align*}
H_D(Y | \text{weak}) &= 0.811 \\
H_D(Y | \text{strong}) &= 1.0
\end{align*}
\]

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

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

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

\[
\text{MakeSubtree}(\text{set of training instances } D)
\]

\[
C = \text{DetermineCandidateSplits}(D)
\]

if stopping criteria is met

make a leaf node \( N \)

determine class label for \( N \)

else

make an internal node \( N \)

\[
S = \text{FindBestSplit}(D, C)
\]

for each group \( k \) of \( S \)

\[
D_k = \text{subset of training data in group } k
\]

\[
k^{th} \text{ child of } N = \text{MakeSubtree}(D_k)
\]

return subtree rooted at \( N \)
Inductive Bias

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

<table>
<thead>
<tr>
<th>learner</th>
<th>hypothesis space bias</th>
<th>preference bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision trees</td>
<td>trees with single-feature, axis-parallel splits</td>
<td>small trees identified by greedy search</td>
</tr>
<tr>
<td>$k$-NN</td>
<td>Decomposition of space determined by nearest neighbors</td>
<td>Instances in neighborhood belong to same class</td>
</tr>
</tbody>
</table>
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