Lecture 9: Nonparametric Methods (KNN, Decision Trees, Random Forests)

Tao LIN

SoE, Westlake University

November 11, 2025

Reading materials & Reference

Reading materials:

- Chapter 16, 18, Probabilistic Machine Learning: an introduction.
- Chapter 8, The Elements of Statistical Learning (Hastie, Tibshirani, Friedman).
- EPFL, CS-433 Machine Learning, Lecture on k-NN.

Table of Contents

Motivation: Why Nonparametric Methods?

- In previous lectures (like Linear/Logistic Regression), we focused on parametric models.

Motivation: Why Nonparametric Methods?

- In previous lectures (like Linear/Logistic Regression), we focused on parametric models.
- These models make **strong assumptions** about the data's distribution (e.g., data is linear, errors are Gaussian).

Motivation: Why Nonparametric Methods?

- In previous lectures (like Linear/Logistic Regression), we focused on parametric models.
- These models make **strong assumptions** about the data's distribution (e.g., data is linear, errors are Gaussian).
- Real-world data often doesn't follow these assumptions.

Motivation: Why Nonparametric Methods?

- In previous lectures (like Linear/Logistic Regression), we focused on parametric models.
- These models make **strong assumptions** about the data's distribution (e.g., data is linear, errors are Gaussian).
- Real-world data often doesn't follow these assumptions.
- We need methods that are more flexible and can learn complex, nonlinear relationships directly from the data.

Motivation: Why Nonparametric Methods?

- In previous lectures (like Linear/Logistic Regression), we focused on parametric models.
- These models make **strong assumptions** about the data's distribution (e.g., data is linear, errors are Gaussian).
- Real-world data often doesn't follow these assumptions.
- We need methods that are more flexible and can learn complex, nonlinear relationships directly from the data.

This leads us to nonparametric methods,
which make minimal assumptions about the data distribution.

Parametric vs. Nonparametric Models

Parametric Models

Nonparametric Models

Parametric vs. Nonparametric Models

Parametric Models

- Assume a fixed functional form (e.g., a line).

Nonparametric Models

- Do not assume a fixed functional form.

Parametric vs. Nonparametric Models

Parametric Models

- Assume a fixed functional form (e.g., a line).
- Model complexity is **fixed** and does **not** grow with the data size N.

Nonparametric Models

- Do not assume a fixed functional form.
- Model complexity **grows** with the data size N.

Parametric vs. Nonparametric Models

Parametric Models

- Assume a fixed functional form (e.g., a line).
- Model complexity is **fixed** and does **not** grow with the data size N.
- We only need to learn a fixed number of parameters $\mathbf{w}$.

Nonparametric Models

- Do not assume a fixed functional form.
- Model complexity **grows** with the data size N.
- The model "learns" directly from the data points.

Parametric vs. Nonparametric Models

Parametric Models

- Assume a fixed functional form (e.g., a line).
- Model complexity is **fixed** and does **not** grow with the data size N.
- We only need to learn a fixed number of parameters $\mathbf{w}$.
- Examples: Linear Regression, Logistic Regression.

Nonparametric Models

- Do not assume a fixed functional form.
- Model complexity **grows** with the data size N.
- The model "learns" directly from the data points.
- Examples: K-Nearest Neighbors (KNN), Decision Trees.

Parametric vs. Nonparametric Models

Parametric Models

- Assume a fixed functional form (e.g., a line).
- Model complexity is **fixed** and does **not** grow with the data size N.
- We only need to learn a fixed number of parameters $\mathbf{w}$.
- Examples: Linear Regression, Logistic Regression.
- **Risk:** High bias (if assumptions are wrong).

Nonparametric Models

- Do not assume a fixed functional form.
- Model complexity **grows** with the data size N.
- The model "learns" directly from the data points.
- Examples: K-Nearest Neighbors (KNN), Decision Trees.
- **Risk:** High variance (can overfit).

Table of Contents

K-Nearest Neighbors (KNN)

- **Core Idea:** “Similar inputs should have similar outputs.”

K-Nearest Neighbors (KNN)

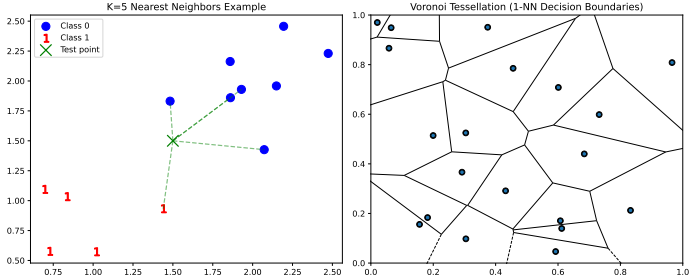
- **Core Idea:** “Similar inputs should have similar outputs.”
- To classify a new point $\mathbf{x}$, look at the K points in the training data that are **closest** to it.

K-Nearest Neighbors (KNN)

- **Core Idea:** “Similar inputs should have similar outputs.”
- To classify a new point $\mathbf{x}$, look at the K points in the training data that are **closest** to it.
- This is a memory-based or lazy learning algorithm.
  - **No “training” phase:** It just memorizes the entire training set $\mathcal{D}$.
  - **All computation happens at prediction time**.

K-Nearest Neighbors (KNN)

- **Core Idea:** “Similar inputs should have similar outputs.”
- To classify a new point $\mathbf{x}$, look at the K points in the training data that are **closest** to it.
- This is a memory-based or lazy learning algorithm.
  - **No “training” phase:** It just memorizes the entire training set $\mathcal{D}$.
  - **All computation happens at prediction time**.



KNN: The Algorithm

- **Input:** Training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, test point $\mathbf{x}$, number of neighbors K.

KNN: The Algorithm

- **Input:** Training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, test point $\mathbf{x}$, number of neighbors K.
- **Algorithm:**
  1. Compute the distance $d(\mathbf{x}, \mathbf{x}_i)$ from $\mathbf{x}$ to every point $\mathbf{x}_i$ in the training set.
  2. Find the set $\mathcal{N}_K(\mathbf{x})$ of the K points $\mathbf{x}_i$ with the smallest distances.
  3. Predict $\hat{y}$ based on the labels $\{y_i | \mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})\}$.

KNN: The Algorithm

- **Input:** Training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, test point $\mathbf{x}$, number of neighbors K.
- **Algorithm:**
  1. Compute the distance $d(\mathbf{x}, \mathbf{x}_i)$ from $\mathbf{x}$ to every point $\mathbf{x}_i$ in the training set.
  2. Find the set $\mathcal{N}_K(\mathbf{x})$ of the K points $\mathbf{x}_i$ with the smallest distances.
  3. Predict $\hat{y}$ based on the labels $\{y_i | \mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})\}$.
- **Prediction:**
  - Classification: Predict the **majority vote** of the neighbors $\hat{y} = \arg\max_c \sum_{\mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})} \mathbb{1}[y_i = c]$.
  - Regression: Predict the **average** of the neighbors' values $\hat{y} = \frac{1}{K} \sum_{\mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})} y_i$.

KNN: The Algorithm

- **Input:** Training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, test point $\mathbf{x}$, number of neighbors K.
- **Algorithm:**
  1. Compute the distance $d(\mathbf{x}, \mathbf{x}_i)$ from $\mathbf{x}$ to every point $\mathbf{x}_i$ in the training set.
  2. Find the set $\mathcal{N}_K(\mathbf{x})$ of the K points $\mathbf{x}_i$ with the smallest distances.
  3. Predict $\hat{y}$ based on the labels $\{y_i | \mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})\}$.
- **Prediction:**
  - Classification: Predict the **majority vote** of the neighbors $\hat{y} = \arg\max_c \sum_{\mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})} \mathbb{1}[y_i = c]$.
  - Regression: Predict the **average** of the neighbors' values $\hat{y} = \frac{1}{K} \sum_{\mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})} y_i$.
- **Variant:** We can use a *weighted* vote/average, giving more weight to closer neighbors.

KNN: The Algorithm

- **Input:** Training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, test point $\mathbf{x}$, number of neighbors K.
- **Algorithm:**
  1. Compute the distance $d(\mathbf{x}, \mathbf{x}_i)$ from $\mathbf{x}$ to every point $\mathbf{x}_i$ in the training set.
  2. Find the set $\mathcal{N}_K(\mathbf{x})$ of the K points $\mathbf{x}_i$ with the smallest distances.
  3. Predict $\hat{y}$ based on the labels $\{y_i | \mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})\}$.
- **Prediction:**
  - Classification: Predict the **majority vote** of the neighbors $\hat{y} = \arg\max_c \sum_{\mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})} \mathbb{1}[y_i = c]$.
  - Regression: Predict the **average** of the neighbors' values $\hat{y} = \frac{1}{K} \sum_{\mathbf{x}_i \in \mathcal{N}_K(\mathbf{x})} y_i$.
- **Variant:** We can use a *weighted* vote/average, giving more weight to closer neighbors.

Key choices:

- The number of neighbors K.
- The distance metric $d(\cdot, \cdot)$ (e.g., Euclidean, Manhattan). Features must be normalized!

KNN: Visual Example

- **K=5 Example**:
  - Test point (X) with 5 nearest neighbors
  - Class probability: $p(y = 1|x, \mathcal{D}) = \frac{3}{5}$
  - Prediction: Class 1 (majority vote)

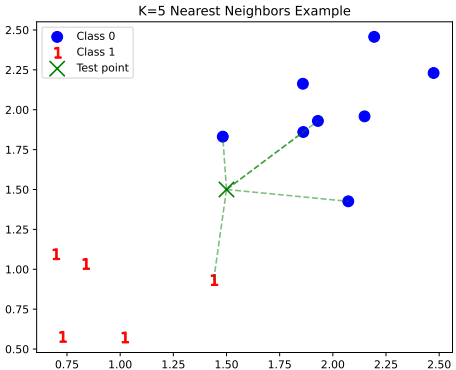
KNN: Visual Example

- **K=5 Example**:
  - Test point (X) with 5 nearest neighbors
  - Class probability: $p(y = 1|x, \mathcal{D}) = \frac{3}{5}$
  - Prediction: Class 1 (majority vote)

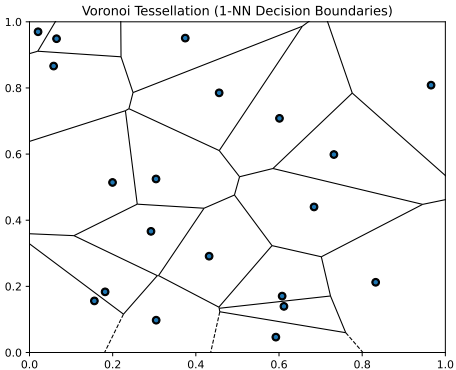
- **1-NN Decision Boundaries**:
  - Voronoi tessellation
  - Each region belongs to closest training point
  - Boundaries where nearest neighbor changes

KNN: Visual Example

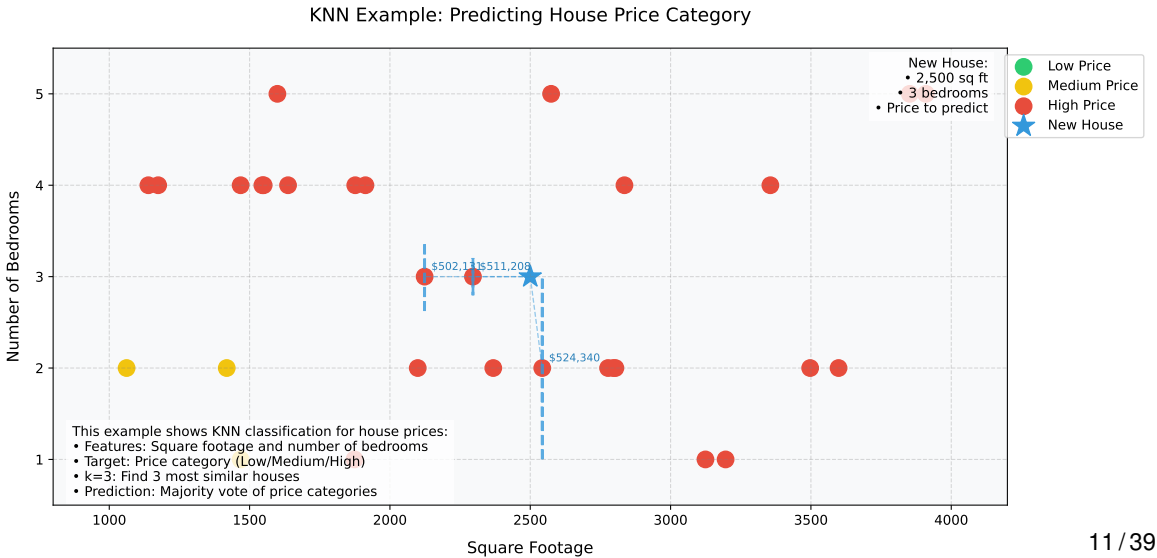
- **K=5 Example**:
  - Test point (X) with 5 nearest neighbors
  - Class probability: $p(y = 1|x, \mathcal{D}) = \frac{3}{5}$
  - Prediction: Class 1 (majority vote)



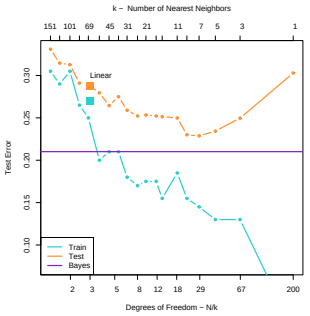
- **1-NN Decision Boundaries**:
  - Voronoi tessellation
  - Each region belongs to closest training point
  - Boundaries where nearest neighbor changes



KNN: Prediction house price categories



KNN: Choosing K (Bias-Variance Tradeoff)



Small K (e.g., $K = 1$)

- **Low Bias:** Very flexible, complex decision boundary.
- **High Variance:** Very sensitive to noise, prone to overfitting.
- Model is **complex**.

Large K (e.g., $K = N$)

- **High Bias:** Over-smoothed boundary (predicts the majority class).
- **Low Variance:** Very stable, not sensitive to noise.
- Model is **simple**.

The Curse of Dimensionality (CoD)

- KNN works well in low dimensions, but **fails** in high dimensions (d).

The Curse of Dimensionality (CoD)

- KNN works well in low dimensions, but **fails** in high dimensions (d).
- **Claim 1: A fixed-size training set covers a dwindling fraction of the space.**

The Curse of Dimensionality (CoD)

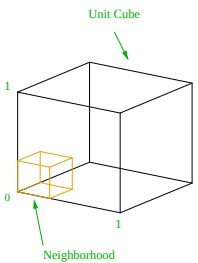
- KNN works well in low dimensions, but **fails** in high dimensions (d).
- **Claim 1: A fixed-size training set covers a dwindling fraction of the space.**
- To capture just $\alpha = 1\%$ of the data (which is uniformly distributed in a $[0, 1]^d$ hypercube), we need a smaller hypercube of side length $r = \alpha^{1/d}$.
  - $d = 1 : r = 0.01^{1/1} = 0.01$ (a tiny slice)
  - $d = 10 : r = 0.01^{1/10} \approx 0.63$ (63% of each axis)
  - $d = 100 : r = 0.01^{1/100} \approx 0.95$ (95% of each axis!)

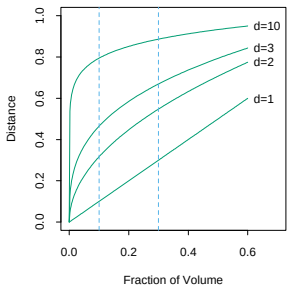
The Curse of Dimensionality (CoD)

- KNN works well in low dimensions, but **fails** in high dimensions (d).
- **Claim 1: A fixed-size training set covers a dwindling fraction of the space.**
- To capture just $\alpha = 1\%$ of the data (which is uniformly distributed in a $[0, 1]^d$ hypercube), we need a smaller hypercube of side length $r = \alpha^{1/d}$.
  - $d = 1 : r = 0.01^{1/1} = 0.01$ (a tiny slice)
  - $d = 10 : r = 0.01^{1/10} \approx 0.63$ (63% of each axis)
  - $d = 100 : r = 0.01^{1/100} \approx 0.95$ (95% of each axis!)
- **Implication:** In high dimensions, any “local” neighborhood that holds even 1% of the data is no longer “local” at all.

The Curse of Dimensionality (CoD)

- KNN works well in low dimensions, but **fails** in high dimensions (d).
- **Claim 1: A fixed-size training set covers a dwindling fraction of the space.**
- To capture just $\alpha = 1\%$ of the data (which is uniformly distributed in a $[0,1]^d$ hypercube), we need a smaller hypercube of side length $r = \alpha^{1/d}$.
  - $d = 1 : r = 0.01^{1/1} = 0.01$ (a tiny slice)
  - $d = 10 : r = 0.01^{1/10} \approx 0.63$ (63% of each axis)
  - $d = 100 : r = 0.01^{1/100} \approx 0.95$ (95% of each axis!)
- **Implication:** In high dimensions, any "local" neighborhood that holds even 1% of the data is no longer "local" at all.





The Curse of Dimensionality (CoD)

- **Claim 2: In high dimensions, all data points are far from each other.**

The Curse of Dimensionality (CoD)

- **Claim 2: In high dimensions, all data points are far from each other.**
- Consider $N = 500$ points uniformly distributed in $[0, 1]^d$. What is the side length r of the smallest hypercube centered at a test point that we expect to contain *at least one* neighbor (with 50% probability)?

The Curse of Dimensionality (CoD)

- **Claim 2: In high dimensions, all data points are far from each other.**
- Consider $N = 500$ points uniformly distributed in $[0, 1]^d$. What is the side length r of the smallest hypercube centered at a test point that we expect to contain *at least one* neighbor (with 50% probability)?
- The required side length is $r = (1 - (1/2)^{1/N})^{1/d}$.
  - For $N = 500, d = 10 \Longrightarrow r \approx 0.52$

The Curse of Dimensionality (CoD)

- **Claim 2: In high dimensions, all data points are far from each other.**
- Consider $N = 500$ points uniformly distributed in $[0, 1]^d$. What is the side length r of the smallest hypercube centered at a test point that we expect to contain *at least one* neighbor (with 50% probability)?
- The required side length is $r = (1 - (1/2)^{1/N})^{1/d}$.
  - For $N = 500, d = 10 \Longrightarrow r \approx 0.52$
- **Implication:** In 10 dimensions, we need our "local" cube to span *half the entire space* just to find *one* neighbor.

The Curse of Dimensionality (CoD)

- **Claim 2: In high dimensions, all data points are far from each other.**
- Consider $N = 500$ points uniformly distributed in $[0, 1]^d$. What is the side length r of the smallest hypercube centered at a test point that we expect to contain *at least one* neighbor (with 50% probability)?
- The required side length is $r = (1 - (1/2)^{1/N})^{1/d}$.
  - For $N = 500, d = 10 \Longrightarrow r \approx 0.52$
- **Implication:** In 10 dimensions, we need our "local" cube to span *half the entire space* just to find *one* neighbor.
- The concept of a "nearest" neighbor becomes meaningless as all points become equidistant and far away.

The Curse of Dimensionality (CoD)

- **Claim 2: In high dimensions, all data points are far from each other.**
- Consider $N = 500$ points uniformly distributed in $[0, 1]^d$. What is the side length r of the smallest hypercube centered at a test point that we expect to contain *at least one* neighbor (with 50% probability)?
- The required side length is $r = (1 - (1/2)^{1/N})^{1/d}$.
  - For $N = 500, d = 10 \Longrightarrow r \approx 0.52$
- **Implication:** In 10 dimensions, we need our "local" cube to span *half the entire space* just to find *one* neighbor.
- The concept of a "nearest" neighbor becomes meaningless as all points become equidistant and far away.

The core assumption of KNN (that "similar" points are "nearby")
breaks down in high dimensions.

Deeper Dive: Proof for Curse of Dimensionality (Claim 2)

Goal: Find the hypercube side length r such that the probability of finding *at least one* of N data points inside is 50%.

- **Volume of the Hypercube:**
  - Data is in a $[0, 1]^d$ space (Total Volume = 1).
  - The volume of our "local" hypercube is $V_r = r^d$.

Deeper Dive: Proof for Curse of Dimensionality (Claim 2)

Goal: Find the hypercube side length r such that the probability of finding *at least one* of N data points inside is 50%.

- **Volume of the Hypercube:**
  - Data is in a $[0, 1]^d$ space (Total Volume = 1).
  - The volume of our "local" hypercube is $V_r = r^d$.

- **Probability for a Single Point:**
  - Since points are uniformly distributed, the probability that a *single* point falls *outside* this cube is the remaining volume:

$$P(\text{point outside}) = 1 - V_r = 1 - r^d$$

Deeper Dive: Proof for Curse of Dimensionality (Claim 2)

Goal: Find the hypercube side length r such that the probability of finding *at least one* of N data points inside is 50%.

- **Volume of the Hypercube:**
  - Data is in a $[0, 1]^d$ space (Total Volume = 1).
  - The volume of our "local" hypercube is $V_r = r^d$.

- **Probability for a Single Point:**
  - Since points are uniformly distributed, the probability that a *single* point falls *outside* this cube is the remaining volume:

$$P(\text{point outside}) = 1 - V_r = 1 - r^d$$

- **Probability for All N Points:**
  - Since all N points are independent, the probability that *all N of them* fall *outside* the cube is:

$$P(\text{all N outside}) = (1 - r^d)^N$$

Deeper Dive: Proof for Curse of Dimensionality (Claim 2)

Goal: Find the hypercube side length r such that the probability of finding *at least one* of N data points inside is 50%.

- **Volume of the Hypercube:**
  - Data is in a $[0, 1]^d$ space (Total Volume = 1).
  - The volume of our "local" hypercube is $V_r = r^d$.

- **Probability for a Single Point:**
  - Since points are uniformly distributed, the probability that a *single* point falls *outside* this cube is the remaining volume:

$$P(\text{point outside}) = 1 - V_r = 1 - r^d$$

- **Probability for All N Points:**
  - Since all N points are independent, the probability that *all N of them* fall *outside* the cube is:

$$P(\text{all N outside}) = (1 - r^d)^N$$

- **Find the Complementary Probability:**
  - The probability we want (at least one *inside*) is the complement:

$$P(\text{at least one inside}) = 1 - P(\text{all N outside}) = 1 - (1 - r^d)^N$$

Deeper Dive: Proof for Curse of Dimensionality (Claim 2)

Goal: Find the hypercube side length r such that the probability of finding *at least one* of N data points inside is 50%.

- **Volume of the Hypercube:**
  - Data is in a $[0, 1]^d$ space (Total Volume = 1).
  - The volume of our "local" hypercube is $V_r = r^d$.

- **Probability for a Single Point:**
  - Since points are uniformly distributed, the probability that a *single* point falls *outside* this cube is the remaining volume:

$$P(\text{point outside}) = 1 - V_r = 1 - r^d$$

- **Probability for All N Points:**
  - Since all N points are independent, the probability that *all N of them* fall *outside* the cube is:

$$P(\text{all N outside}) = (1 - r^d)^N$$

- **Find the Complementary Probability:**
  - The probability we want (at least one *inside*) is the complement:

$$P(\text{at least one inside}) = 1 - P(\text{all N outside}) = 1 - (1 - r^d)^N$$

Deeper Dive: Generalization bound for 1-NN (Setup)

Let's formalize the problem for binary classification $\mathcal{Y} = \{0, 1\}$.

- **Setup:** $(X, Y) \sim \mathcal{D}$ over $\mathcal{X} \times \mathcal{Y} = [0, 1]^d \times \{0, 1\}$

Deeper Dive: Generalization bound for 1-NN (Setup)

Let's formalize the problem for binary classification $\mathcal{Y} = \{0, 1\}$.

- **Setup:** $(X, Y) \sim \mathcal{D}$ over $\mathcal{X} \times \mathcal{Y} = [0, 1]^d \times \{0, 1\}$
- **Goal:** Bound the classification error:

$$L(f) = \mathbb{P}_{(X,Y)\sim\mathcal{D}}(Y \neq f(X))$$

Deeper Dive: Generalization bound for 1-NN (Setup)

Let's formalize the problem for binary classification $\mathcal{Y} = \{0, 1\}$.

- **Setup:** $(X, Y) \sim \mathcal{D}$ over $\mathcal{X} \times \mathcal{Y} = [0, 1]^d \times \{0, 1\}$
- **Goal:** Bound the classification error:

$$L(f) = \mathbb{P}_{(X,Y)\sim\mathcal{D}}(Y \neq f(X))$$

- Let $\eta(x) = \mathbb{P}\{Y = 1 | X = x\}$ be the true, unknown conditional probability of class 1.

Deeper Dive: Generalization bound for 1-NN (Setup)

Let's formalize the problem for binary classification $\mathcal{Y} = \{0, 1\}$.

- **Setup:** $(X, Y) \sim \mathcal{D}$ over $\mathcal{X} \times \mathcal{Y} = [0, 1]^d \times \{0, 1\}$
- **Goal:** Bound the classification error:

$$L(f) = \mathbb{P}_{(X,Y)\sim\mathcal{D}}(Y \neq f(X))$$

- Let $\eta(x) = \mathbb{P}\{Y = 1 | X = x\}$ be the true, unknown conditional probability of class 1.
- **Baselines:**
  - The Bayes Classifier, $f_*(x)$, is the *optimal* possible classifier. It predicts the most likely class:

$$f_*(x) = \mathbf{1}_{\{\eta(x)>1/2\}}$$

Deeper Dive: Generalization bound for 1-NN (Setup)

Let's formalize the problem for binary classification $\mathcal{Y} = \{0, 1\}$.

- **Setup:** $(X, Y) \sim \mathcal{D}$ over $\mathcal{X} \times \mathcal{Y} = [0, 1]^d \times \{0, 1\}$
- **Goal:** Bound the classification error:

$$L(f) = \mathbb{P}_{(X,Y)\sim\mathcal{D}}(Y \neq f(X))$$

- Let $\eta(x) = \mathbb{P}\{Y = 1 | X = x\}$ be the true, unknown conditional probability of class 1.
- **Baselines:**
  - The Bayes Classifier, $f_*(x)$, is the *optimal* possible classifier. It predicts the most likely class:

$$f_*(x) = \mathbf{1}_{\{\eta(x)>1/2\}}$$

  - The error of this optimal classifier is the Bayes Risk, $\mathcal{L}(f_*)$:

$$\mathcal{L}(f_*) = \mathbb{E}_x[\min\{\eta(x), 1 - \eta(x)\}]$$

    This is the absolute minimum, irreducible error for the problem.

Proof of Bayes Classifier

Let $\eta(x) = \mathbb{P}\{Y = 1 | X = x\}$ be the true, unknown conditional probability of class 1.

$$\begin{aligned}
\eta(x) \geq 1/2 &\iff \mathbb{P}(Y = 1 | X = x) \geq 1/2 \\
&\iff \mathbb{P}(Y = 1 | X = x) \geq \mathbb{P}(Y = 0 | X = x) \\
&\iff 1 \in \underset{y \in \{0,1\}}{\operatorname{argmax}} \mathbb{P}(Y = y | X = x)
\end{aligned}$$

Thus $\mathbf{1}_{\eta(x) \geq 1/2} = \operatorname{argmax}_{y \in \{0,1\}} \mathbb{P}(Y = y | X = x) = f_*(x)$

Proof of Bayes Risk Formula

Let $\eta(x) = \mathbb{P}\{Y = 1|X = x\}$ be the true, unknown conditional probability of class 1.

$$
\begin{aligned}
\mathcal{L}(f_*) &= \mathbb{E}_{(X,Y)\sim\mathcal{D}}[\mathbf{1}_{f_*(X)\neq Y}] \\
&= \mathbb{E}_{X\sim\mathcal{D}_X}[\mathbb{E}_{Y\sim\mathcal{D}_{Y|X}}[\mathbf{1}_{f_*(X)\neq Y}|X]] \\
&= \mathbb{E}_{X\sim\mathcal{D}_X}[\mathbb{E}_{Y\sim\mathcal{D}_{Y|X}}[\mathbf{1}_{f_*(X)\neq Y}|X]\mathbf{1}_{\eta(X)\geq 1/2} + \mathbb{E}_{Y\sim\mathcal{D}_{Y|X}}[\mathbf{1}_{f_*(X)\neq Y}|X]\mathbf{1}_{\eta(X)<1/2}] \\
&= \mathbb{E}_{X\sim\mathcal{D}_X}[\mathbb{E}_{Y\sim\mathcal{D}_{Y|X}}[\mathbf{1}_{1\neq Y}|X]\mathbf{1}_{\eta(X)\geq 1/2} + \mathbb{E}_{Y\sim\mathcal{D}_{Y|X}}[\mathbf{1}_{0\neq Y}|X]\mathbf{1}_{\eta(X)<1/2}] \\
&= \mathbb{E}_{X\sim\mathcal{D}_X}[\mathbb{P}(Y = 0|X)\mathbf{1}_{\eta(X)\geq 1/2} + \mathbb{P}(Y = 1|X)\mathbf{1}_{\eta(X)<1/2}] \\
&= \mathbb{E}_{X\sim\mathcal{D}_X}[\min\{\eta(X), 1 - \eta(X)\}]
\end{aligned}
$$

Deeper Dive: Generalization bound for 1-NN (The Bound)

- **Key Assumption:** We need to assume that the problem is "learnable". We assume the probability function $\eta(x)$ is Lipschitz continuous:

$$|\eta(x) - \eta(x')| \leq c||x - x'||$$

(In words: if two points x and x' are close, their probabilities $\eta(x)$ and $\eta(x')$ must also be close.)

Deeper Dive: Generalization bound for 1-NN (The Bound)

- **Key Assumption:** We need to assume that the problem is "learnable". We assume the probability function $\eta(x)$ is Lipschitz continuous:

$$|\eta(x) - \eta(x')| \leq c||x - x'||$$

(In words: if two points x and x' are close, their probabilities $\eta(x)$ and $\eta(x')$ must also be close.)

- **The 1-NN Bound:** Let $\mathcal{L}(f_{S_{train}})$ be the risk of our 1-NN classifier trained on dataset S_{train}. It can be shown that:

$$\mathbb{E}_{S_{train}}[\mathcal{L}(f_{S_{train}})] \leq 2\mathcal{L}(f_*) + c \cdot \mathbb{E}_{S_{train},x}[||x - \text{nbh}_{S_{train},1}(x)||]$$

Deeper Dive: Generalization bound for 1-NN (The Bound)

- **Key Assumption:** We need to assume that the problem is "learnable". We assume the probability function $\eta(x)$ is Lipschitz continuous:

$$|\eta(x) - \eta(x')| \leq c||x - x'||$$

(In words: if two points x and x' are close, their probabilities $\eta(x)$ and $\eta(x')$ must also be close.)

- **The 1-NN Bound:** Let $\mathcal{L}(f_{S_{train}})$ be the risk of our 1-NN classifier trained on dataset S_{train}. It can be shown that:

$$\mathbb{E}_{S_{train}}[\mathcal{L}(f_{S_{train}})] \leq 2\mathcal{L}(f_*) + c \cdot \mathbb{E}_{S_{train},x}[||x - \text{nbh}_{S_{train},1}(x)||]$$

- The 1-NN error is bounded by:
  - (Twice the optimal Bayes risk) + (A "geometric term" that depends on the average distance to the nearest neighbor).

Deeper Dive: 1-NN Analysis (Interpretation)

What does the bound $\mathbb{E}[\mathcal{L}] \leq 2\mathcal{L}(f_*) +$ (geometric term) tell us?

- **Limit 1: As $N \to \infty$ (fixed d)**
  - The average distance to the nearest neighbor $\to 0$.
  - The geometric term vanishes.

$$\mathbb{E}_{S_{train}}[\mathcal{L}(f_{S_{train}})] \leq 2\mathcal{L}(f_*)$$

  - **Takeaway:** With infinite data, the 1-NN error is at most *twice the optimal (Bayes) error*. It's good, but not "consistent" (it doesn't converge *to* $\mathcal{L}(f_*)$).

Deeper Dive: 1-NN Analysis (Interpretation)

What does the bound $\mathbb{E}[\mathcal{L}] \leq 2\mathcal{L}(f_*) +$ (geometric term) tell us?

- **Limit 1: As $N \to \infty$ (fixed d)**
  - The average distance to the nearest neighbor $\to 0$.
  - The geometric term vanishes.

$$\mathbb{E}_{S_{train}}[\mathcal{L}(f_{S_{train}})] \leq 2\mathcal{L}(f_*)$$

  - **Takeaway:** With infinite data, the 1-NN error is at most *twice the optimal (Bayes) error*. It's good, but not "consistent" (it doesn't converge *to* $\mathcal{L}(f_*)$).

- **Limit 2: As $d \to \infty$ (fixed N)**
  - The geometric term $\approx 4c\sqrt{d}N^{-1/(d+1)}$.
  - To keep this error term constant, N must grow *exponentially* with d.
  - **Takeaway:** This is the mathematical proof of the Curse of Dimensionality. For a fixed number of samples N, the error explodes as dimension d increases.

Recap of kNN

- k-NN: a local averaging method for regression and classification
  - use a notion of distance to define *neighborhoods* (= k nearest neighbors)
  - the prediction is a function of these neighborhoods
    - e.g., majority selection for classification, weighted sum for regression
- Bias-variance: small/large k leads to low/high bias and high/low variance
- Curse of dimensionality: as $d \to \infty$, it is harder to define local neighborhoods
- For $N \to \infty$, 1-NN is competitive with Bayes classifier
- N needs to scale exponentially in d to achieve the same error

Table of Contents

Decision Trees

- A different approach that is more structured than KNN.

Decision Trees

- A different approach that is more structured than KNN.
- **Core Idea:** Recursively partition the feature space by asking a series of simple (axis-aligned) questions.

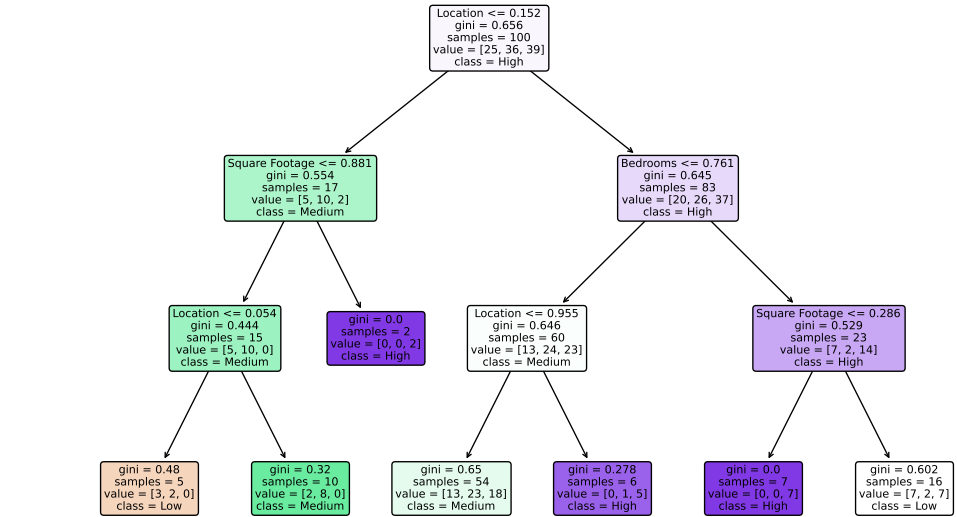
Decision Trees

- A different approach that is more structured than KNN.
- **Core Idea:** Recursively partition the feature space by asking a series of simple (axis-aligned) questions.
- This creates a tree structure:
  - **Internal Nodes:** A question about a feature (e.g., “Is ‘sq. footage‘ > 1500?”).
  - **Branches:** The answer to the question (e.g., “Yes” or “No”).
  - **Leaf Nodes:** A final prediction (e.g., “Expensive” or “Avg. Price”).

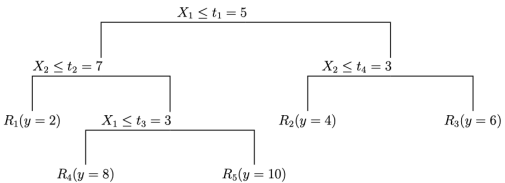
Decision Trees

- A different approach that is more structured than KNN.
- **Core Idea:** Recursively partition the feature space by asking a series of simple (axis-aligned) questions.
- This creates a tree structure:
  - **Internal Nodes:** A question about a feature (e.g., “Is ‘sq. footage‘ > 1500?”).
  - **Branches:** The answer to the question (e.g., “Yes” or “No”).
  - **Leaf Nodes:** A final prediction (e.g., “Expensive” or “Avg. Price”).
- This is a white-box model: the decision logic is highly interpretable.

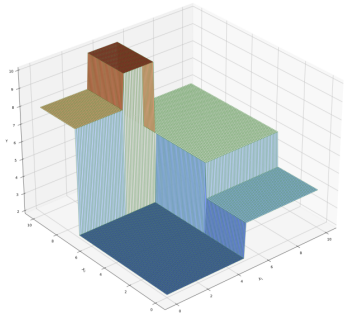
Decision Trees



Decision Trees: Regression Tree



(a)



(b)

Decision Trees: Complete Algorithm

- **Input**: Training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, feature set $\mathcal{F}$

Decision Trees: Complete Algorithm

- **Input**: Training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, feature set $\mathcal{F}$
- **Algorithm**:
  1. If stopping criteria met:
     - Create leaf node with majority class
     - Return
  2. For each feature $f \in \mathcal{F}$:
     - Find best split point t minimizing impurity
     - Calculate information gain: $\text{IG}(f, t) = I(D) - \sum_{s \in \{L,R\}} \frac{|D_s|}{|D|} I(D_s)$
  3. Choose (f^*, t^*) with maximum information gain
  4. Split $\mathcal{D}$ into $\mathcal{D}_L$ and $\mathcal{D}_R$
  5. Recursively grow left and right subtrees

Decision Trees: Finding the Best Split

- How do we grow the tree? We use a **greedy, recursive** algorithm.

Decision Trees: Finding the Best Split

- How do we grow the tree? We use a **greedy, recursive** algorithm.
- At any node, we have a dataset $\mathcal{D}$. We want to find the **best possible split** (a feature f and a threshold t).

Decision Trees: Finding the Best Split

- How do we grow the tree? We use a **greedy, recursive** algorithm.
- At any node, we have a dataset $\mathcal{D}$. We want to find the **best possible split** (a feature f and a threshold t).
- **Goal:** The split should make the resulting child nodes (left $\mathcal{D}_L$ and right $\mathcal{D}_R$) as pure as possible.
  - "Pure" means the nodes contain mostly samples from a *single* class.

Decision Trees: Finding the Best Split

- How do we grow the tree? We use a **greedy, recursive** algorithm.
- At any node, we have a dataset $\mathcal{D}$. We want to find the **best possible split** (a feature f and a threshold t).
- **Goal:** The split should make the resulting child nodes (left $\mathcal{D}_L$ and right $\mathcal{D}_R$) as pure as possible.
  - "Pure" means the nodes contain mostly samples from a *single* class.
- We measure purity (or impurity) and choose the split that **maximizes** the Information Gain:

$$\text{IG}(f, t) = \text{Impurity}(\mathcal{D}) - \left(\frac{|\mathcal{D}_L|}{|\mathcal{D}|} \text{Impurity}(\mathcal{D}_L) + \frac{|\mathcal{D}_R|}{|\mathcal{D}|} \text{Impurity}(\mathcal{D}_R) \right)$$

Decision Trees: Splitting Criteria (Impurity)

How do we measure Impurity$(\mathcal{D})$ for a node with class proportions p_k?

For Classification

For Regression

Decision Trees: Splitting Criteria (Impurity)

How do we measure Impurity$(\mathcal{D})$ for a node with class proportions p_k?

For Classification

- Gini Impurity
  - Measures prob. of misclassifying a random sample.
  - $G(p) = 1 - \sum_{k=1}^{K} p_k^2$
  - Max impurity = 0.5 (for 2 classes). Pure = 0.

For Regression

Decision Trees: Splitting Criteria (Impurity)

How do we measure Impurity$(\mathcal{D})$ for a node with class proportions p_k?

For Classification

- Gini Impurity
  - Measures prob. of misclassifying a random sample.
  - $G(p) = 1 - \sum_{k=1}^{K} p_k^2$
  - Max impurity = 0.5 (for 2 classes). Pure = 0.
- Entropy
  - Measures uncertainty (from information theory).
  - $H(p) = -\sum_{k=1}^{K} p_k \log_2 p_k$
  - Max impurity = 1.0 (for 2 classes). Pure = 0.

For Regression

Decision Trees: Splitting Criteria (Impurity)

How do we measure Impurity$(\mathcal{D})$ for a node with class proportions p_k?

For Classification

- Gini Impurity
  - Measures prob. of misclassifying a random sample.
  - $G(p) = 1 - \sum_{k=1}^{K} p_k^2$
  - Max impurity = 0.5 (for 2 classes). Pure = 0.
- Entropy
  - Measures uncertainty (from information theory).
  - $H(p) = -\sum_{k=1}^{K} p_k \log_2 p_k$
  - Max impurity = 1.0 (for 2 classes). Pure = 0.

For Regression

- Mean Squared Error (MSE)
  - Measures variance of the target variable y.
  - MSE $= \frac{1}{N} \sum_{i=1}^{N} (y_i - \bar{y})^2$
  - Goal is to minimize variance in the leaves.

Decision Trees: Pros and Cons

Advantages

Disadvantages

Decision Trees: Pros and Cons

Advantages

- **Highly interpretable**.

Disadvantages

- **High Variance:** Very prone to overfitting. A small change in data can lead to a completely different tree.

Decision Trees: Pros and Cons

Advantages

- **Highly interpretable**.
- Can handle numerical and categorical data.

Disadvantages

- **High Variance:** Very prone to overfitting. A small change in data can lead to a completely different tree.
- Greedy algorithm finds a local optimum, not guaranteed to be the *best* tree.

Decision Trees: Pros and Cons

Advantages

- **Highly interpretable**.
- Can handle numerical and categorical data.
- No feature scaling (e.g., normalization) is required.

Disadvantages

- **High Variance:** Very prone to overfitting. A small change in data can lead to a completely different tree.
- Greedy algorithm finds a local optimum, not guaranteed to be the *best* tree.
- Can struggle with non-axis-aligned boundaries.

Decision Trees: Pros and Cons

Advantages

- **Highly interpretable**.
- Can handle numerical and categorical data.
- No feature scaling (e.g., normalization) is required.
- Fast for prediction (just follow a path).

Disadvantages

- **High Variance:** Very prone to overfitting. A small change in data can lead to a completely different tree.
- Greedy algorithm finds a local optimum, not guaranteed to be the *best* tree.
- Can struggle with non-axis-aligned boundaries.

Decision Trees: Pros and Cons

Advantages

- **Highly interpretable**.
- Can handle numerical and categorical data.
- No feature scaling (e.g., normalization) is required.
- Fast for prediction (just follow a path).

Disadvantages

- **High Variance:** Very prone to overfitting. A small change in data can lead to a completely different tree.
- Greedy algorithm finds a local optimum, not guaranteed to be the *best* tree.
- Can struggle with non-axis-aligned boundaries.

The main problem of Decision Trees is their **high variance**.
How can we fix this?

Decision Trees: Pros and Cons

Advantages

- **Highly interpretable**.
- Can handle numerical and categorical data.
- No feature scaling (e.g., normalization) is required.
- Fast for prediction (just follow a path).

Disadvantages

- **High Variance:** Very prone to overfitting. A small change in data can lead to a completely different tree.
- Greedy algorithm finds a local optimum, not guaranteed to be the *best* tree.
- Can struggle with non-axis-aligned boundaries.

The main problem of Decision Trees is their **high variance**.
How can we fix this? Ensemble Learning!

Decision Trees: Real-world Applications

- **Medical Diagnosis**:
  - Features: Symptoms, test results, patient history
  - Example: Diagnosing heart disease
    - Chest pain type?
    - Blood pressure?
    - Age and gender?
    - Each split improves diagnostic accuracy

Decision Trees: Real-world Applications

- **Medical Diagnosis**:
  - Features: Symptoms, test results, patient history
  - Example: Diagnosing heart disease
    - Chest pain type?
    - Blood pressure?
    - Age and gender?
    - Each split improves diagnostic accuracy
- **Customer Churn Prediction**:
  - Features: Usage patterns, payment history, customer service interactions
  - Example: Telecom company predicting customer retention
    - Monthly bill amount?
    - Number of service calls?
    - Contract length?

Table of Contents

Ensemble Learning: Bagging

- **Main Idea:** Reduce variance by averaging many “noisy” (high-variance) models.

Ensemble Learning: Bagging

- **Main Idea:** Reduce variance by averaging many “noisy” (high-variance) models.
- Bagging = **B**ootstrap **Agg**regating.

Ensemble Learning: Bagging

- **Main Idea:** Reduce variance by averaging many “noisy” (high-variance) models.
- Bagging = **B**ootstrap **Agg**regating.
- **1. Bootstrap:**
  - Create T new datasets, $\mathcal{D}_1, \ldots, \mathcal{D}_T$.
  - Each $\mathcal{D}_t$ is created by sampling N times **with replacement** from the original data $\mathcal{D}$.

Ensemble Learning: Bagging

- **Main Idea:** Reduce variance by averaging many “noisy” (high-variance) models.
- Bagging = **B**ootstrap **Agg**regating.
- **1. Bootstrap:**
  - Create T new datasets, $\mathcal{D}_1, \ldots, \mathcal{D}_T$.
  - Each $\mathcal{D}_t$ is created by sampling N times **with replacement** from the original data $\mathcal{D}$.
- **2. Aggregate:**
  - Train one model h_t (e.g., a deep Decision Tree) on each bootstrap sample $\mathcal{D}_t$.
  - Final prediction is the average (regression) or majority vote (classification) of all T models.

Ensemble Learning: Bagging

- **Main Idea:** Reduce variance by averaging many “noisy” (high-variance) models.
- Bagging = **B**ootstrap **Agg**regating.
- **1. Bootstrap:**
  - Create T new datasets, $\mathcal{D}_1, \ldots, \mathcal{D}_T$.
  - Each $\mathcal{D}_t$ is created by sampling N times **with replacement** from the original data $\mathcal{D}$.
- **2. Aggregate:**
  - Train one model h_t (e.g., a deep Decision Tree) on each bootstrap sample $\mathcal{D}_t$.
  - Final prediction is the average (regression) or majority vote (classification) of all T models.
- This works because averaging *de-correlated* models reduces variance.

Random Forests

- **Problem with Bagging Trees:** Bagged trees can be highly correlated. If one feature is very strong, every tree will probably split on it first.

Random Forests

- **Problem with Bagging Trees:** Bagged trees can be highly correlated. If one feature is very strong, every tree will probably split on it first.
- Random Forests fix this by forcing more randomness.

Random Forests

- **Problem with Bagging Trees:** Bagged trees can be highly correlated. If one feature is very strong, every tree will probably split on it first.
- Random Forests fix this by forcing more randomness.

Random Forest = Bagging + Feature Randomness

Random Forests

- **Problem with Bagging Trees:** Bagged trees can be highly correlated. If one feature is very strong, every tree will probably split on it first.
- Random Forests fix this by forcing more randomness.

Random Forest = Bagging + Feature Randomness

Algorithm:

- 1. Create a bootstrap sample $\mathcal{D}_t$ (like Bagging).

Random Forests

- **Problem with Bagging Trees:** Bagged trees can be highly correlated. If one feature is very strong, every tree will probably split on it first.
- Random Forests fix this by forcing more randomness.

Random Forest = Bagging + Feature Randomness

Algorithm:

- 1. Create a bootstrap sample $\mathcal{D}_t$ (like Bagging).
- 2. Grow a decision tree h_t. At **each node** in the tree:
  - Randomly select a *subset* of m features (from the total p features).
  - Find the best split **only among those** m **features**.
  - (Typically $m \approx \sqrt{p}$ for classification).

Random Forests

- **Problem with Bagging Trees:** Bagged trees can be highly correlated. If one feature is very strong, every tree will probably split on it first.
- Random Forests fix this by forcing more randomness.

Random Forest = Bagging + Feature Randomness

Algorithm:

- 1. Create a bootstrap sample $\mathcal{D}_t$ (like Bagging).
- 2. Grow a decision tree h_t. At **each node** in the tree:
  - Randomly select a *subset* of m features (from the total p features).
  - Find the best split **only among those** m **features**.
  - (Typically $m \approx \sqrt{p}$ for classification).
- 3. Repeat T times and aggregate the predictions.

Random Forests

- **Problem with Bagging Trees:** Bagged trees can be highly correlated. If one feature is very strong, every tree will probably split on it first.
- Random Forests fix this by forcing more randomness.

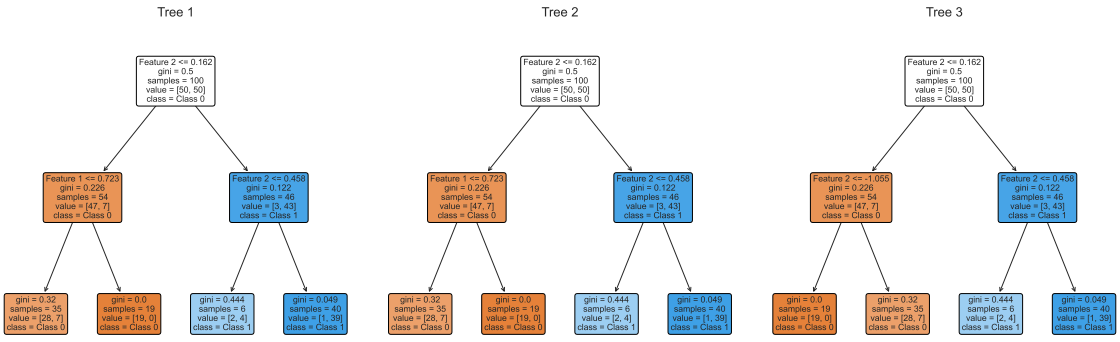
Random Forest = Bagging + Feature Randomness

Algorithm:

- 1. Create a bootstrap sample $\mathcal{D}_t$ (like Bagging).
- 2. Grow a decision tree h_t. At **each node** in the tree:
  - Randomly select a *subset* of m features (from the total p features).
  - Find the best split **only among those** m **features**.
  - (Typically $m \approx \sqrt{p}$ for classification).
- 3. Repeat T times and aggregate the predictions.

This **further de-correlates** the trees,
which significantly reduces variance and makes the model more robust.

Random Forests: Example



Random Forests: Out-of-Bag (OOB) Error

- A useful side-effect of bootstrapping:

Random Forests: Out-of-Bag (OOB) Error

- A useful side-effect of bootstrapping:
- When creating a bootstrap sample $\mathcal{D}_t$, each point $\mathbf{x}_i$ has a probability $(1 - 1/N)^N \approx e^{-1} \approx 0.368$ of *not* being picked.

Random Forests: Out-of-Bag (OOB) Error

- A useful side-effect of bootstrapping:
- When creating a bootstrap sample $\mathcal{D}_t$, each point $\mathbf{x}_i$ has a probability $(1 - 1/N)^N \approx e^{-1} \approx 0.368$ of *not* being picked.
- This means each tree h_t is trained on 63% of the data. The remaining 37% is its Out-of-Bag (OOB) sample.

Random Forests: Out-of-Bag (OOB) Error

- A useful side-effect of bootstrapping:
- When creating a bootstrap sample $\mathcal{D}_t$, each point $\mathbf{x}_i$ has a probability $(1 - 1/N)^N \approx e^{-1} \approx 0.368$ of *not* being picked.
- This means each tree h_t is trained on 63% of the data. The remaining 37% is its Out-of-Bag (OOB) sample.
- **To estimate performance:**
  - For each data point $\mathbf{x}_i$, make a prediction using *only* the trees that did **not** have $\mathbf{x}_i$ in their training set (its OOB trees).
  - Calculate the error (e.g., MSE, misclassification) on these OOB predictions.

Random Forests: Out-of-Bag (OOB) Error

- A useful side-effect of bootstrapping:
- When creating a bootstrap sample $\mathcal{D}_t$, each point $\mathbf{x}_i$ has a probability $(1 - 1/N)^N \approx e^{-1} \approx 0.368$ of *not* being picked.
- This means each tree h_t is trained on 63% of the data. The remaining 37% is its Out-of-Bag (OOB) sample.
- **To estimate performance:**
  - For each data point $\mathbf{x}_i$, make a prediction using *only* the trees that did **not** have $\mathbf{x}_i$ in their training set (its OOB trees).
  - Calculate the error (e.g., MSE, misclassification) on these OOB predictions.
- **Benefit:** This gives an unbiased estimate of the test error *for free*, without needing a separate validation set or cross-validation.

Random Forests: Feature Importance

Since RF is a “black box” how do we know which features are important?

1. Mean Decrease in Impurity (MDI)

2. Permutation Importance (MDA)

Random Forests: Feature Importance

Since RF is a “black box” how do we know which features are important?

1. Mean Decrease in Impurity (MDI)

- When training, measure the total impurity reduction (Gini/MSE) from all splits on a feature f_j.

2. Permutation Importance (MDA)

- A more robust method.

Random Forests: Feature Importance

Since RF is a “black box” how do we know which features are important?

1. Mean Decrease in Impurity (MDI)

- When training, measure the total impurity reduction (Gini/MSE) from all splits on a feature f_j.
- Average this reduction over all trees in the forest.

2. Permutation Importance (MDA)

- A more robust method.
  1. Calculate the OOB error for the forest.

Random Forests: Feature Importance

Since RF is a “black box” how do we know which features are important?

1. Mean Decrease in Impurity (MDI)

- When training, measure the total impurity reduction (Gini/MSE) from all splits on a feature f_j.
- Average this reduction over all trees in the forest.
- **Pros:** Fast (calculated during training).

2. Permutation Importance (MDA)

- A more robust method.
  1. Calculate the OOB error for the forest.
  2. For a feature f_j: **shuffle** (permute) its values in the OOB data.

Random Forests: Feature Importance

Since RF is a “black box” how do we know which features are important?

1. Mean Decrease in Impurity (MDI)

- When training, measure the total impurity reduction (Gini/MSE) from all splits on a feature f_j.
- Average this reduction over all trees in the forest.
- **Pros:** Fast (calculated during training).
- **Cons:** Biased towards high-cardinality features.

2. Permutation Importance (MDA)

- A more robust method.
  1. Calculate the OOB error for the forest.
  2. For a feature f_j: **shuffle** (permute) its values in the OOB data.
  3. Recalculate OOB error with the shuffled data.

Random Forests: Feature Importance

Since RF is a “black box” how do we know which features are important?

1. Mean Decrease in Impurity (MDI)

- When training, measure the total impurity reduction (Gini/MSE) from all splits on a feature f_j.
- Average this reduction over all trees in the forest.
- **Pros:** Fast (calculated during training).
- **Cons:** Biased towards high-cardinality features.

2. Permutation Importance (MDA)

- A more robust method.
  1. Calculate the OOB error for the forest.
  2. For a feature f_j: **shuffle** (permute) its values in the OOB data.
  3. Recalculate OOB error with the shuffled data.
- The *drop* in performance is the “importance” of f_j.

Random Forests: Feature Importance

Since RF is a “black box” how do we know which features are important?

1. Mean Decrease in Impurity (MDI)

- When training, measure the total impurity reduction (Gini/MSE) from all splits on a feature f_j.
- Average this reduction over all trees in the forest.
- **Pros:** Fast (calculated during training).
- **Cons:** Biased towards high-cardinality features.

2. Permutation Importance (MDA)

- A more robust method.
  1. Calculate the OOB error for the forest.
  2. For a feature f_j: **shuffle** (permute) its values in the OOB data.
  3. Recalculate OOB error with the shuffled data.
- The *drop* in performance is the “importance” of f_j.
- **Pros:** More reliable, less biased.

Table of Contents

Summary of Progression

The story of this lecture:

- **KNN:** Simple & intuitive, but slow at prediction and fails in high dimensions (CoD).

Summary of Progression

The story of this lecture:

- **KNN:** Simple & intuitive, but slow at prediction and fails in high dimensions (CoD).
- **Decision Tree:** Solves speed/scaling issues and is interpretable, but has very high variance (overfits).

Summary of Progression

The story of this lecture:

- **KNN:** Simple & intuitive, but slow at prediction and fails in high dimensions (CoD).
- **Decision Tree:** Solves speed/scaling issues and is interpretable, but has very high variance (overfits).
- **Random Forest:** Solves overfitting by *ensembling* trees, giving high performance but losing interpretability.

Comparison of Nonparametric Methods

Property	KNN	Decision Tree	Random Forest
	Medium	**High**	Low
	High (low K)	**Very High**	Low
	Fast (Lazy)	Medium	Slow
	Slow	Fast	Medium
	Yes	No	No
	Poor	Medium	Good

Comparison of Nonparametric Methods

Property	**KNN**	**Decision Tree**	**Random Forest**
Interpretability	Medium	**High**	Low
	High (low K)	**Very High**	Low
	Fast (Lazy)	Medium	Slow
	Slow	Fast	Medium
	Yes	No	No
	Poor	Medium	Good

Comparison of Nonparametric Methods

Property	**KNN**	**Decision Tree**	**Random Forest**
Interpretability	Medium	**High**	Low
Overfitting	High (low K)	**Very High**	Low
	Fast (Lazy)	Medium	Slow
	Slow	Fast	Medium
	Yes	No	No
	Poor	Medium	Good

Comparison of Nonparametric Methods

Property	**KNN**	**Decision Tree**	**Random Forest**
Interpretability	Medium	**High**	Low
Overfitting	High (low K)	**Very High**	Low
Training Speed	**Fast** (Lazy)	Medium	Slow
	Slow	Fast	Medium
	Yes	No	No
	Poor	Medium	Good

Comparison of Nonparametric Methods

Property	**KNN**	**Decision Tree**	**Random Forest**
Interpretability	Medium	**High**	Low
Overfitting	High (low K)	**Very High**	Low
Training Speed	**Fast** (Lazy)	Medium	Slow
Prediction Speed	**Slow**	Fast	Medium
	Yes	No	No
	Poor	Medium	Good

Comparison of Nonparametric Methods

Property	**KNN**	**Decision Tree**	**Random Forest**
Interpretability	Medium	**High**	Low
Overfitting	High (low K)	**Very High**	Low
Training Speed	**Fast** (Lazy)	Medium	Slow
Prediction Speed	**Slow**	Fast	Medium
Needs Scaling?	**Yes**	No	No
	Poor	Medium	Good

Comparison of Nonparametric Methods

Property	**KNN**	**Decision Tree**	**Random Forest**
Interpretability	Medium	**High**	Low
Overfitting	High (low K)	**Very High**	Low
Training Speed	**Fast** (Lazy)	Medium	Slow
Prediction Speed	**Slow**	Fast	Medium
Needs Scaling?	**Yes**	No	No
High-D Perf.	Poor	Medium	Good

Real-world Applications

- **KNN**:
  - Recommendation systems
  - Image recognition (simple cases)
  - Anomaly detection

Real-world Applications

- **KNN**:
  - Recommendation systems
  - Image recognition (simple cases)
  - Anomaly detection
- **Decision Trees**:
  - Medical diagnosis
  - Credit scoring
  - Customer churn prediction

Real-world Applications

- **KNN**:
  - Recommendation systems
  - Image recognition (simple cases)
  - Anomaly detection
- **Decision Trees**:
  - Medical diagnosis
  - Credit scoring
  - Customer churn prediction
- **Random Forests**:
  - Bioinformatics
  - Financial forecasting
  - Computer vision