Classification Methods

CMSC 173 - Machine Learning

Noel Jeffrey Pinton

October 13, 2025

Department of Computer Science
University of the Philippines - Cebu

Outline

Introduction & Motivation

What is Classification?

Definition

Classification is a supervised learning task where we predict a discrete class label for new observations based on training examples.

Key Characteristics

- Supervised learning
- Discrete output (classes/categories)
- Learn from labeled training data
- Make predictions on new data

Formulation

Given training data:

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$$

where:

- $\mathbf{x}_i \in \mathbb{R}^d$ = feature vector
- $y_i \in \{1, 2, \ldots, C\}$ = class label

Goal: Learn function $f : \mathbb{R}^d \to \{1, \ldots, C\}$

Types

Binary: 2 classes (spam/not spam)
Multi-class: $C > 2$ classes (digits 0-9)

Classification vs Regression

Classification

- **Output:** Discrete categories
- **Examples:**
  - Email: spam/not spam
  - Medical: disease diagnosis
  - Image: object recognition
  - Finance: loan approval
- **Metrics:** Accuracy, precision, recall
- **Goal:** Predict class membership

Regression

- **Output:** Continuous values
- **Examples:**
  - House price prediction
  - Temperature forecasting
  - Stock price estimation
  - Age prediction
- **Metrics:** MSE, MAE, R^2
- **Goal:** Predict numerical value

Key Difference

Classification predicts **categories**, regression predicts **quantities**.

Real-World Applications

Medical & Healthcare

- **Disease diagnosis**: Cancer detection, diabetes screening
- **Medical imaging**: X-ray, MRI classification
- **Drug discovery**: Molecular classification

Finance & Business

- **Credit scoring**: Loan default prediction
- **Fraud detection**: Transaction classification
- **Customer churn**: Retention analysis

Technology & Media

- **Image recognition**: Object detection, face recognition
- **Text classification**: Sentiment analysis, spam filtering
- **Recommendation**: Content categorization

Science & Engineering

- **Biology**: Species identification, gene classification
- **Quality control**: Defect detection
- **Remote sensing**: Land cover classification

Common Theme

All involve learning patterns from labeled examples to classify new instances!

Classification Methods Overview

This lecture covers three fundamental classification methods:

Naive Bayes

Probabilistic

- Based on Bayes theorem
- Assumes feature independence
- Fast, simple
- Works with small data

Best for: Text classification, real-time prediction

K-Nearest Neighbors

Instance-based

- Distance-based
- Non-parametric
- No training phase
- Intuitive

Best for: Pattern recognition, recommendation systems

Decision Trees

Rule-based

- Hierarchical decisions
- Interpretable
- Handles mixed types
- Foundation for ensembles

Best for: Medical diagnosis, credit scoring

Learning Objectives

Understand theory, implementation, and practical application of each method.

Naive Bayes Classification

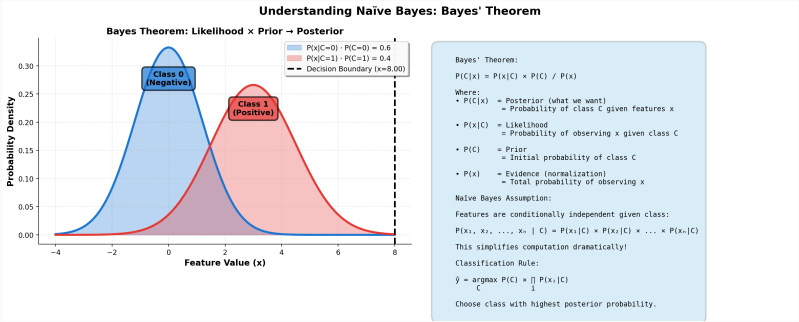
Bayes' Theorem: Foundation

Bayes' Theorem

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k) \cdot P(C_k)}{P(\mathbf{x})}$$

where:

- $P(C_k|\mathbf{x})$ = **Posterior**: Probability of class C_k given features $\mathbf{x}$
- $P(\mathbf{x}|C_k)$ = **Likelihood**: Probability of features given class
- $P(C_k)$ = **Prior**: Probability of class (before seeing data)
- $P(\mathbf{x})$ = **Evidence**: Probability of features (normalization constant)



Classification Rule

The "Naive" Assumption

Feature Independence

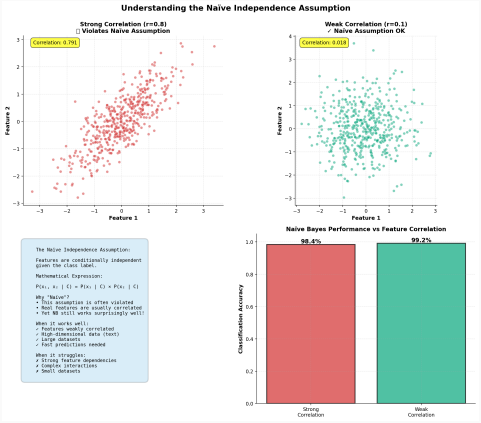
Assumption: Features are conditionally independent given the class:

$$P(\mathbf{x}|C_k) = P(x_1, x_2, \ldots, x_d|C_k)$$
$$= P(x_1|C_k) \cdot P(x_2|C_k) \cdots P(x_d|C_k)$$
$$= \prod_{i=1}^{d} P(x_i|C_k)$$

Why "Naive"?

This assumption is often violated in practice, but:

- Makes computation tractable
- Reduces parameters exponentially
- Works surprisingly well empirically



Practical Impact

Enables classification with minimal training data and fast prediction!

Naive Bayes: Classification Formula

Combining Bayes theorem with independence assumption:

Full Formula

$$P(C_k|\mathbf{x}) \propto P(C_k) \prod_{i=1}^{d} P(x_i|C_k)$$

Classification decision:

$$\hat{y} = \arg\max_k \left[P(C_k) \prod_{i=1}^{d} P(x_i|C_k) \right]$$

In Practice (Log Probabilities)

To avoid numerical underflow, use log probabilities:

$$\hat{y} = \arg\max_k \left[\log P(C_k) + \sum_{i=1}^{d} \log P(x_i|C_k) \right]$$

Key Insight

We only need to estimate $P(C_k)$ and $P(x_i|C_k)$ from training data!

Types of Naive Bayes Classifiers

1. Gaussian Naive Bayes

For **continuous** features: Assume features follow Gaussian distribution

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_{k,i}^2}} \exp\left(-\frac{(x_i - \mu_{k,i})^2}{2\sigma_{k,i}^2}\right)$$

Parameters: Mean $\mu_{k,i}$ and variance $\sigma_{k,i}^2$ for each feature i and class k

Use cases: Iris classification, continuous sensor data

2. Multinomial Naive Bayes

For **discrete counts** (e.g., word frequencies): Assume multinomial distribution

$$P(x_i|C_k) = \frac{N_{k,i} + \alpha}{\sum_j N_{k,j} + \alpha d}$$

where $N_{k,i}$ = count of feature i in class k, α = smoothing parameter

Use cases: Text classification, document categorization

3. Bernoulli Naive Bayes

For **binary** features: Assume Bernoulli distribution

$$P(x_i|C_k) = P(i|C_k)^{x_i} \cdot (1 - P(i|C_k))^{(1-x_i)}$$

Gaussian Naive Bayes: Details

Training (Parameter Estimation)

For each class C_k and feature i:

1. Prior probability:

$$P(C_k) = \frac{n_k}{n}$$

where n_k = number of samples in class k

2. Mean:

$$\mu_{k,i} = \frac{1}{n_k} \sum_{\mathbf{x}_j \in C_k} x_{j,i}$$

3. Variance:

$$\sigma^2_{k,i} = \frac{1}{n_k} \sum_{\mathbf{x}_j \in C_k} (x_{j,i} - \mu_{k,i})^2$$

Prediction

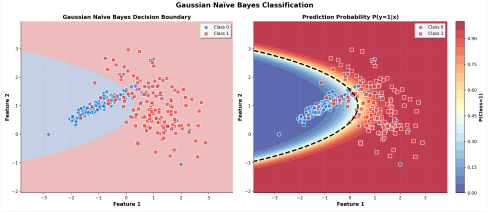
For new sample $\mathbf{x}$:

1. Compute log-likelihood for each class:

$$\log P(C_k|\mathbf{x}) = \log P(C_k) + \sum_{i=1}^{d} \log P(x_i|C_k)$$

2. Choose class with highest value:

$$\hat{y} = \arg\max_k \log P(C_k|\mathbf{x})$$



Naive Bayes Example: Binary Classification

Dataset: Weather conditions for playing tennis

Training Data

Outlook	Temp	Humidity	Play
Sunny	Hot	High	No
Sunny	Hot	High	No
Overcast	Hot	High	Yes
Rainy	Mild	High	Yes
Rainy	Cool	Normal	Yes
Rainy	Cool	Normal	No
Overcast	Cool	Normal	Yes
Sunny	Mild	High	No
Sunny	Cool	Normal	Yes

Question

Predict: Outlook=Sunny, Temp=Cool, Humidity=High

Step 1: Compute Priors

$P(\text{Yes}) = \frac{5}{9} \approx 0.56 \quad P(\text{No}) = \frac{4}{9} \approx 0.44$

Step 2: Compute Likelihoods

For Class = Yes:

- $P(\text{Sunny}|\text{Yes}) = \frac{2}{5} = 0.40$
- $P(\text{Cool}|\text{Yes}) = \frac{3}{5} = 0.60$
- $P(\text{High}|\text{Yes}) = \frac{1}{5} = 0.20$

For Class = No:

- $P(\text{Sunny}|\text{No}) = \frac{2}{4} = 0.50$
- $P(\text{Cool}|\text{No}) = \frac{1}{4} = 0.25$
- $P(\text{High}|\text{No}) = \frac{3}{4} = 0.75$

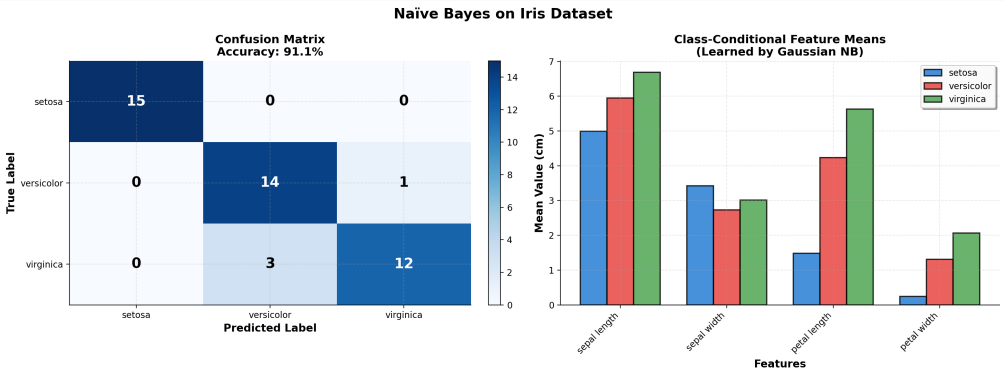
Step 3: Calculate Posteriors

$P(\text{Yes}|\mathbf{x}) \propto 0.56 \times 0.40 \times 0.60 \times 0.20 = \mathbf{0.027}$

$P(\text{No}|\mathbf{x}) \propto 0.44 \times 0.50 \times 0.25 \times 0.75 = \mathbf{0.041}$

Prediction: No (higher posterior)

Naive Bayes: Iris Dataset Example



Gaussian NB Performance

- **Training Accuracy:** 96.0%
- **Test Accuracy:** 93.3%
- **Training Time:** $<$1ms
- **Prediction Time:** $<$1ms

Key Observations

- Linear decision boundaries
- Fast training and prediction
- Some overlap between classes
- Good generalization despite naive assumption

Laplace Smoothing

Problem: Zero Probabilities

If a feature value never appears with a class in training: $P(x_i|C_k) = 0$

This makes the entire product zero: $P(C_k|\mathbf{x}) = 0$

Solution: Laplace (Additive) Smoothing

Add pseudo-count α (typically $\alpha = 1$):

For discrete features:

$$P(x_i = v|C_k) = \frac{N_{k,v} + \alpha}{N_k + \alpha \cdot V}$$

where:

- $N_{k,v}$ = count of value v in class k
- N_k = total count in class k
- V = number of unique values for feature i

Effect

- Prevents zero probabilities
- Provides small probability to unseen events
- $\alpha = 1$ called "Laplace smoothing", $\alpha < 1$ called "Lidstone smoothing"

Naive Bayes Algorithm (Pseudocode)

Algorithm 1 Gaussian Naive Bayes

Require: Training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$

Ensure: Class prediction for new sample $\mathbf{x}$

Training Phase:
for each class $k = 1, \ldots, C$ **do**
  Compute prior: $P(C_k) = \frac{n_k}{n}$
  for each feature $i = 1, \ldots, d$ **do**
    Compute mean: $\mu_{k,i} = \frac{1}{n_k} \sum_{\mathbf{x}_j \in C_k} x_{j,i}$
    Compute variance: $\sigma^2_{k,i} = \frac{1}{n_k} \sum_{\mathbf{x}_j \in C_k} (x_{j,i} - \mu_{k,i})^2$
  end for
end for

Prediction Phase:
for each class $k = 1, \ldots, C$ **do**
  $\text{score}_k = \log P(C_k)$
  for each feature $i = 1, \ldots, d$ **do**
    $\text{score}_k \mathrel{+}= \log P(x_i | C_k)$ (using Gaussian PDF)
  end for
end for
return $\arg\max_k \text{score}_k$

Complexity

Naive Bayes: Advantages & Disadvantages

Advantages

- **Fast:** Training and prediction are very efficient
- **Simple:** Easy to understand and implement
- **Small data:** Works well with limited training samples
- **Multi-class:** Naturally handles multiple classes
- **Scalable:** Scales linearly with features and samples
- **Probabilistic:** Provides probability estimates
- **Online learning:** Can update incrementally

Disadvantages

- **Independence assumption:** Rarely true in practice
- **Zero frequency:** Needs smoothing for unseen values
- **Poor estimates:** Probability estimates not always accurate
- **Feature correlations:** Cannot capture dependencies
- **Continuous data:** Distribution assumption may not hold
- **Imbalanced data:** Prior bias with skewed classes

When to Use Naive Bayes

Good for: Text classification, spam filtering, real-time prediction, high-dimensional data
Avoid when: Feature independence badly violated, need probability calibration

K-Nearest Neighbors (KNN)

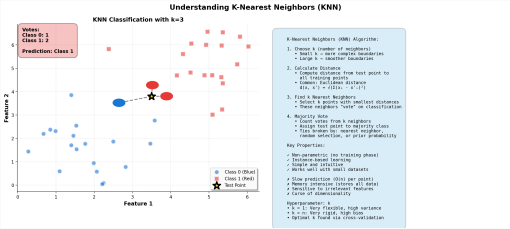
K-Nearest Neighbors: The Idea

Core Concept

"You are the average of your k closest neighbors"

Classification rule:

1. Find k nearest training samples
2. Vote based on their labels
3. Assign most common class



Key Characteristics

- **Non-parametric:** No model to train
- **Instance-based:** Stores all training data
- **Lazy learning:** Computation at prediction time
- **Intuitive:** Easy to understand and visualize

Intuition

Similar inputs should have similar outputs!

KNN: Distance Metrics

Common Distance Metrics

For feature vectors $\mathbf{x} = (x_1, \ldots, x_d)$ and $\mathbf{y} = (y_1, \ldots, y_d)$:

1. Euclidean Distance (L2)

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

Most common choice, assumes all features equally important

2. Manhattan Distance (L1)

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|$$

Less sensitive to outliers, good for high dimensions

3. Minkowski Distance (general)

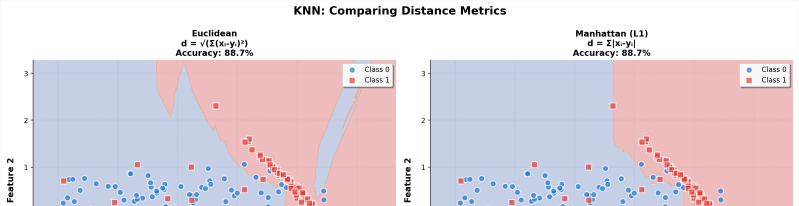
$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}$$

Generalization: $p = 1$ (Manhattan), $p = 2$ (Euclidean)

4. Chebyshev Distance (L∞)

$$d(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|$$

Maximum difference across any dimension



KNN Algorithm

Algorithm 2 K-Nearest Neighbors Classification

Require: Training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, parameter k, test sample $\mathbf{x}_{\text{test}}$
Ensure: Predicted class $\hat{y}$

Training Phase:
Store all training samples $\mathcal{D}$ (no explicit training!)

Prediction Phase:
Initialize distance array $D = []$
for each training sample $(\mathbf{x}_i, y_i)$ in $\mathcal{D}$ **do**
  Compute distance: $d_i = d(\mathbf{x}_{\text{test}}, \mathbf{x}_i)$
  Append (d_i, y_i) to D
end for
Sort D by distance (ascending)
Select k nearest neighbors: $N_k = \{(d_1, y_1), \ldots, (d_k, y_k)\}$
Classification (Majority Vote):
Count occurrences of each class in N_k
return class with highest count

Complexity

Training: $O(1)$ **Prediction:** $O(nd)$ per query

KNN Example: Manual Calculation

Binary classification with $k = 3$, Euclidean distance

Training Data

ID	x_1	x_2	Class
A	1	2	Red
B	2	3	Red
C	3	1	Red
D	5	4	Blue
E	5	6	Blue
F	6	5	Blue

Test Point

$\mathbf{x}_{\text{test}} = (4, 3)$

Predict class with $k = 3$

Step 1: Compute Distances

$$d(A) = \sqrt{(4-1)^2 + (3-2)^2} = \sqrt{10} \approx \mathbf{3.16}$$

$$d(B) = \sqrt{(4-2)^2 + (3-3)^2} = \sqrt{4} = \mathbf{2.00}$$

$$d(C) = \sqrt{(4-3)^2 + (3-1)^2} = \sqrt{5} \approx \mathbf{2.24}$$

$$d(D) = \sqrt{(4-5)^2 + (3-4)^2} = \sqrt{2} \approx \mathbf{1.41}$$

$$d(E) = \sqrt{(4-5)^2 + (3-6)^2} = \sqrt{10} \approx 3.16$$

$$d(F) = \sqrt{(4-6)^2 + (3-5)^2} = \sqrt{8} \approx 2.83$$

Step 2: Find 3-Nearest Neighbors

Sorted: D(1.41), B(2.00), C(2.24), ...

3 nearest: D (Blue), B (Red), C (Red)

Step 3: Majority Vote

Blue: 1 vote **Red:** 2 votes

Prediction: Red (majority class)

KNN: Choosing K

Effect of K

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

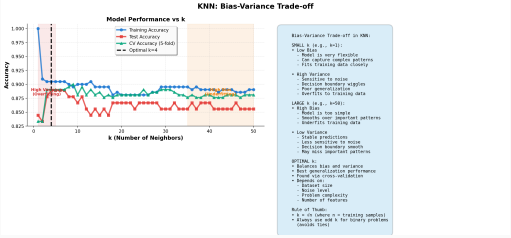
- **Flexible** decision boundary
- **Low bias**, high variance
- Sensitive to noise
- Prone to overfitting

Large K (e.g., $k = n$):

- **Smooth** decision boundary
- **High bias**, low variance
- More robust to noise
- Prone to underfitting

Rule of Thumb

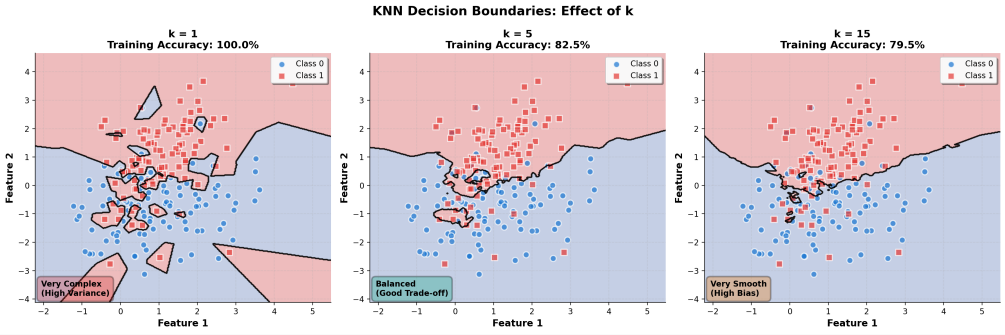
$k = \sqrt{n}$ or use cross-validation



Best Practice

- Use **odd** k for binary classification (avoid ties)
- Try multiple values: $k \in \{1, 3, 5, 7, 9, \ldots\}$
- Use cross-validation to select optimal k
- Consider $k \leq 20$ for most problems

KNN: Decision Boundaries



$k = 1$

- Highly irregular
- Captures all details
- Overfits training data

$k = 5$

- Balanced complexity
- Smooth but flexible
- Good generalization

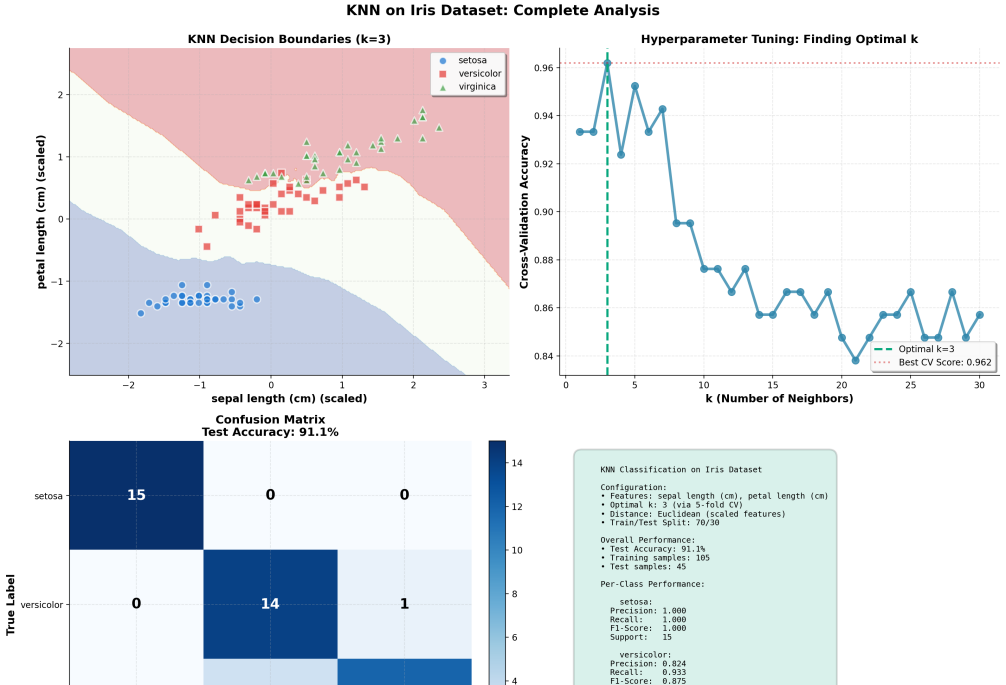
$k = 15$

- Very smooth
- Less flexible
- May underfit

Key Insight

KNN creates **non-linear** decision boundaries that adapt to local structure!

KNN: Iris Dataset Example

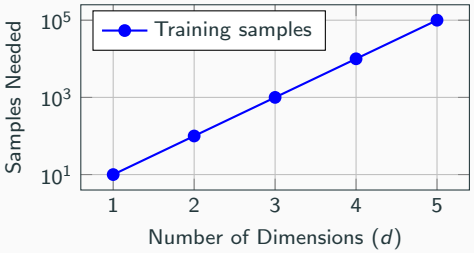


Curse of Dimensionality

Problem: Distance Concentration

In high dimensions, distances between points become similar!

All points appear equidistant $\Rightarrow$ "nearest" neighbors not actually close



Why This Happens

- Volume of hypersphere grows exponentially with d
- Most data lies near surface of hypersphere
- Ratio of nearest to farthest distance $\to 1$
- Need exponentially more data as d increases

Rule of Thumb

KNN works best with $d < 20$ features

Mitigation Strategies

- **Feature selection:** Remove irrelevant features

KNN: Advantages & Disadvantages

Advantages

- **Simple:** Easy to understand and implement
- **No training:** No explicit model fitting
- **Non-parametric:** No assumptions about data distribution
- **Flexible:** Non-linear decision boundaries
- **Multi-class:** Naturally handles multiple classes
- **Adaptive:** Decision boundary adapts locally
- **Incremental:** Easy to add new training data

Disadvantages

- **Slow prediction:** $O(nd)$ per query
- **Memory intensive:** Stores all training data
- **Curse of dimensionality:** Poor in high dimensions
- **Scaling sensitive:** Must normalize features
- **Imbalanced data:** Majority class dominates
- **Choosing k:** Requires cross-validation
- **No interpretability:** Cannot explain decisions

When to Use KNN

Good for: Small to medium datasets ($n < 10,000$), low dimensions ($d < 20$), non-linear patterns
Avoid when: Large datasets, high dimensions, real-time requirements, need interpretability

Decision Trees

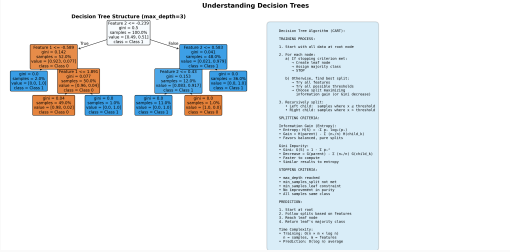
Decision Trees: The Idea

Core Concept

Learn a **tree of if-then-else rules** to classify data

Structure:

- **Root node:** Start of tree (top)
- **Internal nodes:** Decision/test on feature
- **Branches:** Outcome of test
- **Leaf nodes:** Class predictions (bottom)



Classification Process

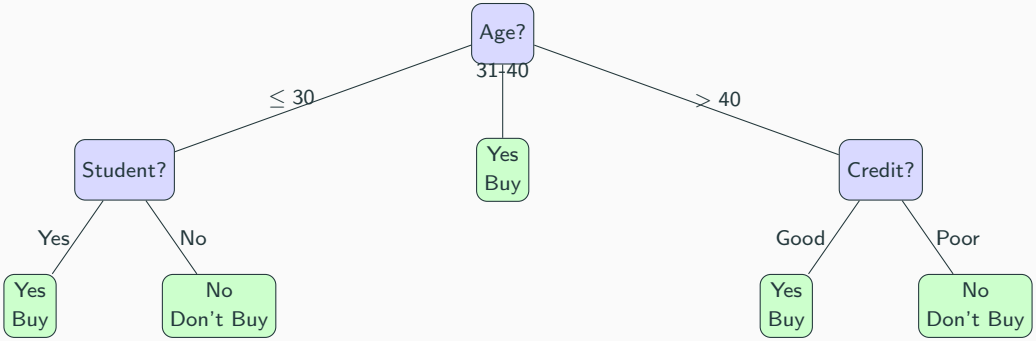
1. Start at root
2. Test feature at current node
3. Follow branch based on result
4. Repeat until leaf
5. Return class at leaf

Key Advantage

Highly **interpretable** - can explain every decision!

Decision Tree: Example

Classification: Will a person buy a computer?



Example Classification

Query: Age=35, Student=No, Credit=Good

Path: Age $>$ 40? No $\rightarrow$ Age 31-40? Yes $\rightarrow$ **Buy**

Building Decision Trees: CART Algorithm

CART = Classification And Regression Trees

Greedy recursive algorithm:

1. Start with all data at root
2. Find **best split** that maximizes information gain
3. Partition data into two subsets
4. Recursively build left and right subtrees
5. Stop when stopping criterion met (pure node, max depth, min samples)

Key Question

How do we determine the "**best split**"?

$\Rightarrow$ Use splitting criteria: Gini impurity or entropy

Splitting Decision

For each feature j and threshold t:

- Split data: $\mathcal{D}_{\text{left}} = \{\mathbf{x} | x_j \leq t\}$, $\mathcal{D}_{\text{right}} = \{\mathbf{x} | x_j > t\}$
- Compute impurity of split
- Choose (j, t) with lowest impurity

Splitting Criteria

1. Gini Impurity (CART default)

Measures probability of incorrect classification:

$$\text{Gini}(D) = 1 - \sum_{k=1}^{C} p_k^2$$

where p_k = proportion of class k in dataset D

Range: [0, 1], where 0 = pure (all same class), higher = more mixed

Interpretation: Probability of misclassifying if label randomly assigned

2. Entropy (ID3, C4.5 algorithms)

Measures disorder/uncertainty:

$$\text{Entropy}(D) = -\sum_{k=1}^{C} p_k \log_2(p_k)$$

Range: [0, $\log_2 C$], where 0 = pure, higher = more uncertain

Interpretation: Average number of bits needed to encode class

Information Gain

Definition

Information Gain = Reduction in impurity after split

$$\text{IG}(D, j, t) = \text{Impurity}(D) - \left(\frac{|D_L|}{|D|} \text{Impurity}(D_L) + \frac{|D_R|}{|D|} \text{Impurity}(D_R) \right)$$

where:

- D = parent dataset
- D_L = left child (samples with $x_j \leq t$)
- D_R = right child (samples with $x_j > t$)
- $|D|$ = number of samples

Splitting Algorithm

For each feature j and each possible threshold t:

1. Compute information gain $\text{IG}(D, j, t)$
2. Keep track of best (j^*, t^*) with highest gain

Split on (j^*, t^*) to maximize information gain

Goal

Maximize purity of child nodes (minimize impurity)

Decision Tree Example: Manual Construction

Dataset: Play tennis? (same as Naive Bayes example)

Training Data (9 samples)

Outlook	Temp	Humidity	Play
Sunny	Hot	High	No
Sunny	Hot	High	No
Overcast	Hot	High	Yes
Rainy	Mild	High	Yes
Rainy	Cool	Normal	Yes
Rainy	Cool	Normal	No
Overcast	Cool	Normal	Yes
Sunny	Mild	High	No
Sunny	Cool	Normal	Yes

Class Distribution

Yes: 5, No: 4

Step 1: Compute Root Entropy

$$\text{Entropy(root)} = -\frac{5}{9}\log_2\frac{5}{9} - \frac{4}{9}\log_2\frac{4}{9}$$
$$\approx 0.994 \text{ bits}$$

Step 2: Try Splitting on Outlook

Sunny (3 samples): No=2, Yes=1
Entropy $= -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} \approx 0.918$

Overcast (2 samples): Yes=2
Entropy $= 0$ (pure!)

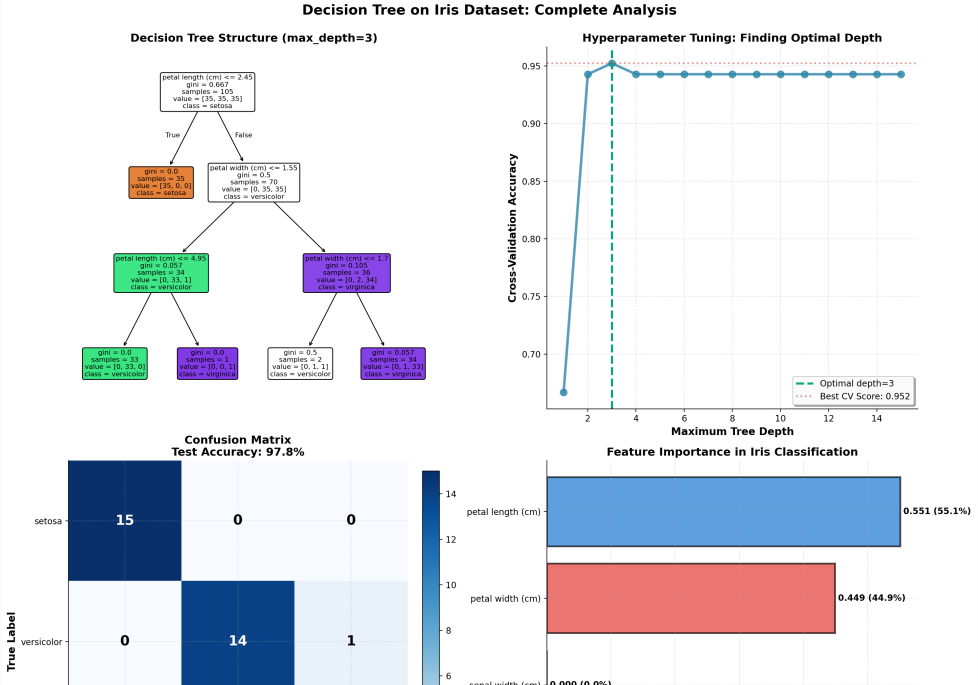
Rainy (4 samples): Yes=2, No=1, Yes=1
Entropy $= -\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1.0$

Step 3: Information Gain

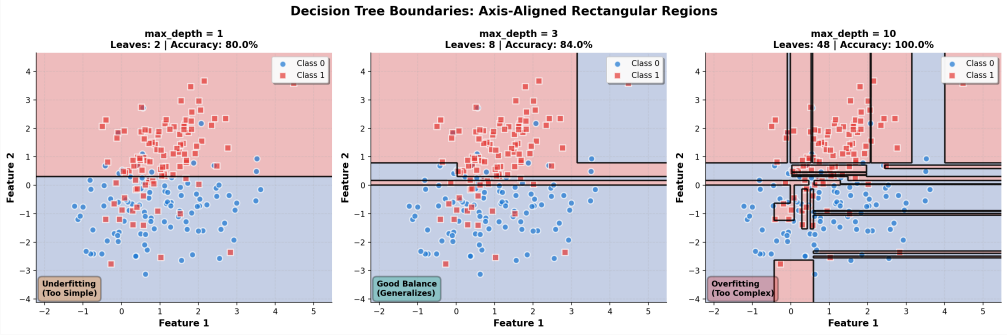
$$\text{IG(Outlook)} = 0.994 - \left(\frac{3}{9}(0.918) + \frac{2}{9}(0) + \frac{4}{9}(1.0)\right)$$
$$= 0.994 - 0.750 = \mathbf{0.244}$$

Result: Outlook has good information gain!

Decision Tree: Iris Dataset Example



Decision Boundaries: Different Depths



Depth = 2

- Simple boundaries
- High bias
- Underfits

Depth = 5

- Balanced
- Good generalization
- Optimal complexity

Depth = 20

- Complex boundaries
- High variance
- Overfits

Key Insight

Decision trees create **axis-aligned rectangular** decision regions!

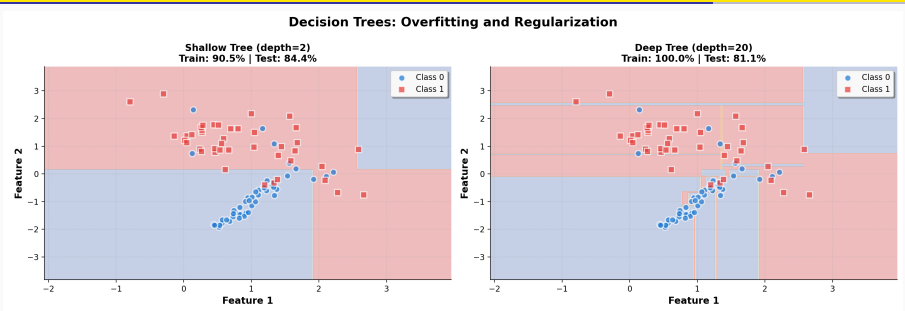
Decision Tree Algorithm (Pseudocode)

Algorithm 3 CART Decision Tree (Recursive)

Require: Dataset D, features F, max_depth, min_samples
Ensure: Decision tree T

function BuildTree(D, depth)
if stopping criterion met **then**
  return LeafNode(majority class in D)
end if
best_gain $\leftarrow 0$
$(j^*, t^*) \leftarrow$ None
for each feature $j \in F$ **do**
  for each threshold t (midpoints of sorted values) **do**
    Split: $D_L \leftarrow \{\mathbf{x} \in D : x_j \leq t\}$, $D_R \leftarrow \{\mathbf{x} \in D : x_j > t\}$
    gain $\leftarrow$ InformationGain(D, D_L, D_R)
    if gain $>$ best_gain **then**
      best_gain $\leftarrow$ gain
      $(j^*, t^*) \leftarrow (j, t)$
    end if
  end for
end for
$D_L, D_R \leftarrow$ Split D on feature j^* at threshold t^*
$T_L \leftarrow$ BuildTree(D_L, depth + 1)
$T_R \leftarrow$ BuildTree(D_R, depth + 1)
return DecisionNode(j^*, t^*, T_L, T_R)

Overfitting and Pruning



Preventing Overfitting in Decision Trees:

PROBLEM:
- Deep trees memorize training data
- Perfect training accuracy (100%)
- Poor test accuracy
- High variance, low bias

REGULARIZATION STRATEGIES:

1. PRE-PRUNING (Early Stopping)
   Set constraints BEFORE growing tree:

   max_depth
   - Limit tree depth
   - Most common constraint
   - Try: 3-10 for start

   min_samples_split
   - Minimum samples to split node
   - Default: 2
   - Try: 10-50 for large datasets

   min_samples_leaf
   - Minimum samples in leaf
   - Prevents tiny leaves
   - Try: 5-20

   max_leaf_nodes
   - Limit total leaves
   - Controls complexity directly

   min_impurity_decrease
   - Minimum gain to split
   - Prunes weak splits

2. POST-PRUNING
   Grow full tree, then prune:

   ccp_alpha (cost-complexity)
   - Penalizes tree complexity
   - Higher α → smaller tree
   - Find via cross-validation

3. ENSEMBLE METHODS

Feature Importance

Computing Importance

For each feature j:

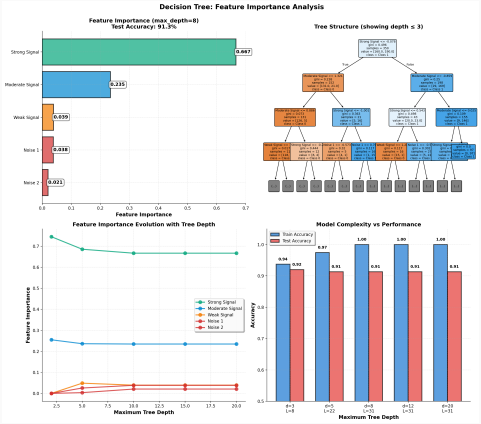
$$\text{Importance}(j) = \sum_{\text{nodes using } j} \frac{n_{\text{node}}}{n_{\text{total}}} \cdot \Delta\text{Impurity}$$

where:

- Sum over all nodes that split on feature j
- Weight by fraction of samples at node
- ΔImpurity = reduction in impurity

Interpretation

- Higher importance = more useful for prediction
- Normalized to sum to 1.0
- Features not used have 0 importance
- Can identify redundant features



Use Cases

Feature selection, domain insights, model interpretation, debugging

Ensemble Methods Preview

Limitation: Single Tree Instability

- Small changes in data $\rightarrow$ very different tree
- High variance
- Sensitive to training set

Solution: Ensemble Methods

Combine multiple trees for better performance:

1. Random Forest

- Train many trees on bootstrap samples
- Random feature subset at each split
- Average predictions (voting for classification)

2. Gradient Boosting (XGBoost, LightGBM)

- Train trees sequentially
- Each tree corrects errors of previous
- Weighted combination of predictions

3. AdaBoost

- Weighted training samples
- Focus on hard-to-classify examples

Decision Trees: Advantages & Disadvantages

Advantages

- **Interpretable:** Easy to visualize and explain
- **No scaling:** Works with raw features
- **Mixed types:** Handles numerical and categorical
- **Non-linear:** Captures complex patterns
- **Feature selection:** Automatic importance ranking
- **Missing values:** Can handle with surrogates
- **Fast prediction:** $O(\log n)$ time

Disadvantages

- **Overfitting:** Prone to high variance
- **Instability:** Small data changes = big tree changes
- **Axis-aligned:** Cannot capture diagonal boundaries well
- **Biased:** Favors features with many values
- **Imbalanced:** Struggles with skewed classes
- **Local optima:** Greedy algorithm
- **Extrapolation:** Poor outside training range

When to Use Decision Trees

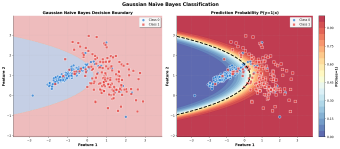
Good for: Interpretability needed, mixed feature types, feature interactions, baseline model
Avoid when: Need best accuracy (use ensembles), many irrelevant features

Method Comparison & Selection

Comparison: Decision Boundaries

Visual comparison on synthetic 2D dataset:

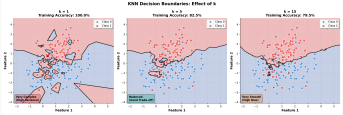
Naive Bayes



Properties

- Linear/quadratic
- Smooth probabilistic
- Assumes Gaussian

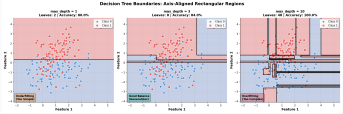
K-Nearest Neighbors



Properties

- Non-linear
- Locally adaptive
- No assumptions

Decision Tree



Properties

- Axis-aligned
- Rectangular regions
- Rule-based

Key Insight

Different methods create fundamentally different decision boundaries!

Detailed Comparison Table

Criterion	Naive Bayes	K-Nearest Neighbors	Decision Trees
Type	Probabilistic	Instance-based	Rule-based
Training Time	Very fast ($O(nd)$)	None ($O(1)$)	Medium ($O(nd \log n)$)
Prediction Time	Very fast ($O(Cd)$)	Slow ($O(nd)$)	Fast ($O(\log n)$)
Memory	Low (parameters only)	High (stores all data)	Medium (tree structure)
Interpretability	Medium (probabilistic)	Low (no model)	High (rules)
Assumptions	Feature independence	Locality principle	None
Overfitting Risk	Low	Medium-High	High (needs pruning)
Handling Noise	Robust	Sensitive	Medium
Feature Scaling	Not needed	Critical	Not needed
Missing Values	Can handle	Requires imputation	Can handle
Categorical Features	Natural	Needs encoding	Natural
High Dimensions	Good	Poor (curse)	Medium
Imbalanced Data	Prior adjustment	Class weighting	Sample weighting

Recommendation

Try all three methods and use cross-validation to choose the best for your data!

When to Use Each Method

Use Naive Bayes When:

- **Text classification** (spam filtering, document categorization)
- **Real-time prediction** required (fast training and prediction)
- **High-dimensional data** (works well even with many features)
- **Small training set** (few parameters to estimate)
- **Probabilistic output** needed (uncertainty estimates)
- **Baseline model** (quick first approach)

Use K-Nearest Neighbors When:

- **Small to medium dataset** ($n < 10,000$)
- **Low dimensions** ($d < 20$)
- **Non-linear patterns** present
- **No training time** budget
- **Anomaly detection** (outliers far from neighbors)
- **Recommendation systems** (similarity-based)

Use Decision Trees When:

- **Interpretability crucial** (medical, legal, financial decisions)
- **Mixed feature types** (numerical and categorical)
- **Feature interactions** important

Performance Comparison: Iris Dataset

Accuracy Comparison

Method	Train	Test
Naive Bayes	96.0%	93.3%
KNN ($k = 5$)	96.7%	96.7%
Decision Tree (depth=3)	98.3%	93.3%

Timing Comparison

Method	Train	Predict
Naive Bayes	$<$1 ms	$<$1 ms
KNN	0 ms	2-5 ms
Decision Tree	1-2 ms	$<$1 ms

Key Insights

- All methods perform well on Iris
- KNN best test accuracy
- Decision tree shows slight overfit
- Naive Bayes fastest overall
- Timing differences negligible for small data

Important Note

Performance varies greatly by dataset!

Always use cross-validation to compare methods on **your specific data**.

Best Practices & Guidelines

General Best Practices

Data Preprocessing

- **Handle missing values:** Impute or remove (method-dependent)
- **Scale features:** Essential for KNN, not for Naive Bayes/Trees
- **Encode categorical:** One-hot encoding (KNN), label encoding (Trees)
- **Remove outliers:** Especially important for KNN
- **Feature engineering:** Create domain-specific features
- **Balance classes:** Use SMOTE, class weights, or resampling

Model Selection & Tuning

- **Split data properly:** Train/validation/test or cross-validation
- **Tune hyperparameters:** Grid search or random search
- **Compare multiple methods:** Don't commit to one prematurely
- **Use appropriate metrics:** Accuracy, precision, recall, F1, AUC
- **Validate generalization:** Check on held-out test set

Model Interpretation

- **Analyze errors:** Confusion matrix, error analysis
- **Feature importance:** Which features matter most?
- **Decision boundaries:** Visualize for 2D data
- **Cross-validate:** Report mean and std of metrics

Common Pitfalls & Solutions

Pitfall 1: Not Scaling Features (KNN)

Problem: Features with large ranges dominate distance calculations

Solution: Always use StandardScaler or MinMaxScaler for KNN

Pitfall 2: Using Test Data for Hyperparameter Tuning

Problem: Test accuracy is optimistically biased

Solution: Use separate validation set or cross-validation

Pitfall 3: Overfitting Deep Decision Trees

Problem: Tree memorizes training data, poor generalization

Solution: Use pruning (max_depth, min_samples_split, min_samples_leaf)

Pitfall 4: Ignoring Class Imbalance

Problem: Majority class dominates, poor minority class performance

Solution: Use class weights, SMOTE, or stratified sampling

Pitfall 5: Naive Bayes with Correlated Features

Problem: Independence assumption violated, overconfident predictions

Solution: Remove redundant features or use different method

Hyperparameter Tuning Guide

Naive Bayes

- **Type:** Gaussian, Multinomial, or Bernoulli (match to data type)
- **Smoothing α:** Try [0.1, 0.5, 1.0, 2.0, 5.0] (for discrete features)
- **Priors:** Uniform or class-balanced (adjust for imbalance)

K-Nearest Neighbors

- k: Try odd values [1, 3, 5, 7, 9, 15, 21] (cross-validate!)
- **Distance metric:** Euclidean, Manhattan, Minkowski
- **Weights:** 'uniform' or 'distance' (weight by inverse distance)
- **Algorithm:** 'brute', 'kd_tree', 'ball_tree' (for efficiency)

Decision Trees

- **max_depth:** Try [3, 5, 7, 10, 15, None] (most important!)
- **min_samples_split:** Try [2, 10, 20, 50] (prevent overfitting)
- **min_samples_leaf:** Try [1, 5, 10, 20] (smoother boundaries)
- **criterion:** 'gini' or 'entropy' (usually similar performance)
- **max_features:** 'sqrt', 'log2', or None (for Random Forest)

Tip

Use GridSearchCV or RandomizedSearchCV from scikit-learn for systematic tuning!

Feature Engineering Tips

For Naive Bayes

- **Text:** Use TF-IDF or count vectors (Multinomial NB)
- **Discretize:** Bin continuous features if needed
- **Remove correlated:** Drop highly redundant features
- **Log transform:** For skewed distributions (Gaussian NB)

For K-Nearest Neighbors

- **Dimensionality reduction:** Use PCA to reduce features
- **Feature selection:** Remove irrelevant/noisy features
- **Polynomial features:** Create interaction terms
- **Distance metric:** Choose appropriate for domain (e.g., cosine for text)

For Decision Trees

- **Keep raw features:** Trees handle non-linearity automatically
- **Interaction terms:** Not needed (tree finds them)
- **Categorical encoding:** Use label encoding (not one-hot)
- **Create domain features:** Trees can use them directly

Universal Tip

Domain knowledge is more valuable than complex feature engineering!

Summary & Conclusion

Key Takeaways

Core Concepts

- **Classification:** Supervised learning for discrete labels
- **Three fundamental approaches:** Probabilistic, instance-based, rule-based
- **Different assumptions:** Match method to data characteristics
- **Trade-offs:** Speed vs accuracy vs interpretability

Method Summaries

- **Naive Bayes:** Fast probabilistic, assumes independence, great for text
- **K-Nearest Neighbors:** Simple instance-based, non-parametric, slow prediction
- **Decision Trees:** Interpretable rules, non-linear, prone to overfit

Best Practices

- **Preprocess appropriately:** Scaling for KNN, encoding for trees
- **Use cross-validation:** For model selection and hyperparameter tuning
- **Try multiple methods:** No single best classifier for all problems
- **Interpret results:** Understand why model makes predictions

What We Covered

1. **Introduction:** Classification definition, applications, comparison with regression
2. **Naive Bayes:** Bayes theorem, independence assumption, Gaussian/Multinomial/Bernoulli variants, worked example
3. **K-Nearest Neighbors:** Distance metrics, algorithm, choosing k, curse of dimensionality, worked example
4. **Decision Trees:** CART algorithm, splitting criteria (Gini/Entropy), pruning, feature importance, worked example
5. **Comparison:** Decision boundaries, performance metrics, when to use each method
6. **Best Practices:** Data preprocessing, hyperparameter tuning, common pitfalls, feature engineering

Next Steps

- **Practice:** Implement all three methods from scratch
- **Experiment:** Try on different datasets (UCI ML Repository)
- **Read ahead:** Ensemble methods (Random Forest, Boosting)
- **Workshop:** Hands-on exercises in Jupyter notebook

Further Reading

Textbooks

- **Hastie et al.**: "The Elements of Statistical Learning" (Ch. 9: Additive Models, Trees)
- **Bishop**: "Pattern Recognition and Machine Learning" (Ch. 4: Linear Models for Classification)
- **Murphy**: "Machine Learning: A Probabilistic Perspective" (Ch. 3, 16: Generative and Discriminative Models)
- **Mitchell**: "Machine Learning" (Ch. 3: Decision Tree Learning)

Key Papers

- Breiman et al. (1984): "Classification and Regression Trees (CART)"
- Quinlan (1986): "Induction of Decision Trees (ID3)"
- Cover & Hart (1967): "Nearest Neighbor Pattern Classification"
- Rish (2001): "An Empirical Study of the Naive Bayes Classifier"

Implementations

- **scikit-learn**: GaussianNB, MultinomialNB, KNeighborsClassifier, DecisionTreeClassifier
- **Documentation**: https://scikit-learn.org/stable/supervised_learning.html
- **Tutorials**: scikit-learn user guide, Kaggle Learn

Questions?

Thank you for your attention!

Noel Jeffrey Pinton
Department of Computer Science
University of the Philippines - Cebu