IN402

Machine Learning

Chapter

6

K-Nearest Neighbors: Learning Through Similarity

Author: Abbas El-Hajj Youssef

University: Lebanese University

Department: Department of Computer Science

These notes extend course materials taught by Course Instructor with additional content from textbooks and supplementary resources.

Disclaimer: *This is not an official course document.*

Contents

1 The Paradox of Memory-Based Learning

Imagine you're exploring an art museum for the first time. You pause before an unfamiliar painting—swirling brushstrokes, shimmering light dancing on water lilies. "What style is this?" you wonder.

Your companion, a seasoned art enthusiast, doesn't launch into a lecture about color theory or artistic movements. Instead, she gestures to three nearby canvases: "Look at these—notice the soft edges, the play of light, the atmospheric quality. They all share something, don't they?" In that moment, you understand: this is Impressionism. Not because someone defined it for you, but because you recognized the family resemblance.

This is **learning through similarity**—the principle that underlies one of machine learning's most elegant yet powerful algorithms: **K-Nearest Neighbors (K-NN)**.

The Universal Pattern

Throughout your life, you've used this strategy countless times:

- **New restaurant?** You check reviews from customers with similar tastes.
- **Medical diagnosis?** Doctors compare symptoms to similar past cases.
- **Recommendation systems?** Streaming apps suggest movies based on what similar users watch.
- **Language learning?** You recognize new words by their similarity to ones you know.

The underlying principle is deceptively simple: *things that are similar tend to behave similarly*. This isn't just intuition—it's the foundation of K-NN.

1.1 The Fork in the Road: Two Philosophies of Learning

Before we dive into K-NN, we need to understand where it sits in the landscape of machine learning approaches. Consider two fundamentally different ways to learn from data:

1.1.1 The Parametric Approach: Compression Through Abstraction

You've already encountered this philosophy in linear regression and logistic regression. These methods:

- Learn a fixed set of parameters (weights $\boldsymbol{w}$, bias b) that capture patterns
- Compress all training data into these learned parameters
- Discard the original data after training—predictions use only the parameters
- Make strong assumptions about the relationship between inputs and outputs

Think of this as writing a rulebook. Once you've written the rules, you can throw away all the examples you used to derive them.

1.1.2 The Non-Parametric Approach: Learning by Example

K-NN embodies a radically different philosophy:

- Stores *every* training example in memory
- Makes no assumptions about the functional form of relationships

- Predicts by consulting similar past experiences
- Adapts naturally to local patterns in the data

This is like keeping a comprehensive photo album. When you encounter something new, you flip through your album to find similar examples and learn from them.

The Tradeoff: Rules vs. Examples

Aspect	Parametric (e.g., Linear Regression)	Non-Parametric (K-NN)
Training	Learn parameters, discard data	Store all data
Assumptions	Strong (e.g., linearity)	Minimal
Flexibility	Fixed complexity	Adapts to data
Memory	$O(n)$ parameters	$O(mn)$ full dataset
Prediction	Fast: $O(n)$	Slow: $O(mn)$
Interpretability	Explicit formula	Example-based

Neither approach is universally superior—each excels in different scenarios. K-NN shines when:

1. Decision boundaries are highly non-linear
2. You have sufficient memory to store training data
3. You can afford slower predictions
4. Local patterns matter more than global structure

What You'll Master

By the end of this chapter, you'll have a complete understanding of:

1. **The Core Algorithm:** How K-NN makes predictions through local voting
2. **Geometric Intuition:** Visualizing decision boundaries through Voronoi tessellations
3. **Design Choices:** Distance metrics, choosing K, and feature scaling
4. **The Bias-Variance Lens:** How K controls model flexibility
5. **Practical Considerations:** When to use K-NN, computational costs, and limitations
6. **Real Applications:** From the Two Moons dataset to recommender systems

Most importantly, you'll develop an intuition for *when* K-NN is the right tool—and when it isn't.

1.2 A Motivating Example: The Iris Classification Challenge

Let's make this concrete. Imagine you're a botanist in 1936, holding an iris flower. Your task: determine its species based on measurements of its petals and sepals.
You have a notebook containing measurements from 150 previously identified irises of three species: *setosa*, *versicolor*, and *virginica*. Your new flower has:

- Petal length: 4.5 cm
- Petal width: 1.4 cm

The Parametric Approach: You could fit a logistic regression model, learning decision boundaries that separate the species. This gives you an equation: "If petal length > 2.5 and petal width < 1.7, predict *versicolor*."
The K-NN Approach: You flip through your notebook, find the three most similar flowers (same petal measurements), and see that all three were *versicolor*. Done. Your prediction: *versicolor*.
No equations. No parameters to learn. Just similarity and voting.

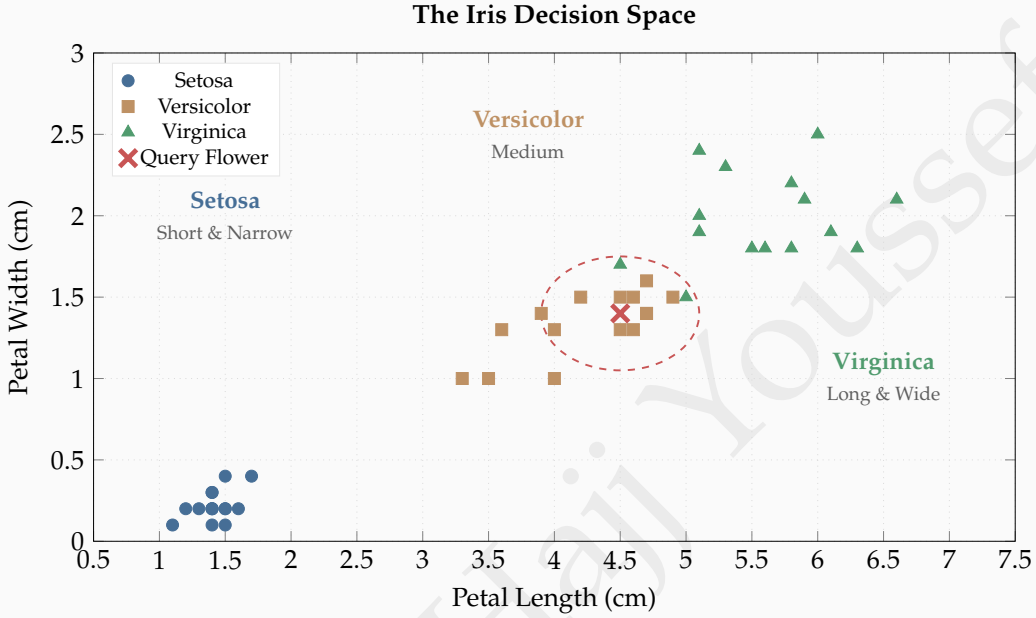


Figure 1: *The Iris dataset in feature space. Each point represents a flower, positioned by its petal measurements. The query flower (red ×) is surrounded by three nearest neighbors (dashed circle), all versicolor. Notice how species form natural clusters—K-NN exploits this local structure.*

This chapter will unpack how this simple idea—comparing to similar examples—becomes a rigorous, powerful machine learning algorithm.

2 The Geometry of Similarity: Voronoi Tessellations

Before we formalize the algorithm, let's develop geometric intuition. Understanding how K-NN partitions space makes everything else click into place.

2.1 The Simplest Case: When $K = 1$

Imagine you're a real estate agent. You have six houses you've sold, each at a different location in a city. When a client asks about the likely price of a house at a new location, you consult your records and quote the price of the *single closest* house you've sold before.

This is 1-Nearest Neighbor ($K = 1$). It creates a remarkable partition of space: every location in the city "belongs" to whichever house is nearest.

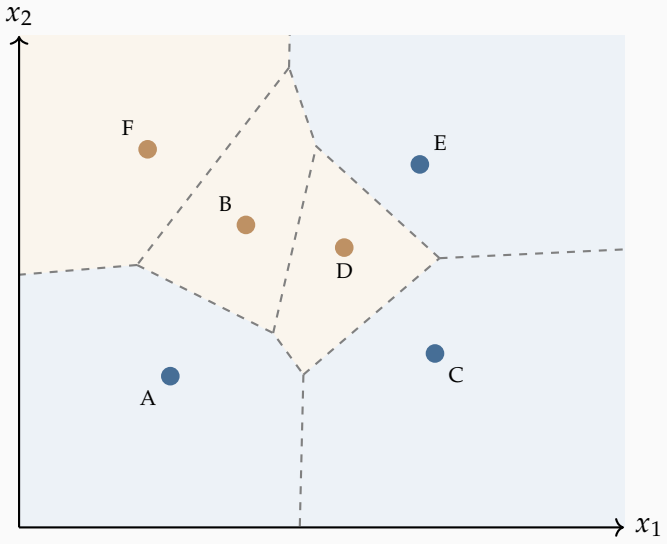


Figure 2: ***Voronoi tessellation for*** $K = 1$**.** *Each colored region (cell) contains all query points for which that training example is the nearest neighbor. The dashed boundaries lie exactly equidistant between neighboring points—these are perpendicular bisectors. Any query falling in cell A will be classified as Blue (Class 0). Notice the irregular, adaptive boundaries—this is* $K = 1$*'s remarkable flexibility and its greatest weakness.*

Why These Boundaries Form

The decision boundary appears where the predicted class changes. With $K = 1$, this happens exactly where two training points of different classes are equidistant from a query point.

Mathematically, the boundary between points $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$ is:

$$\{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{x}^{(i)}\| = \|\boldsymbol{x} - \boldsymbol{x}^{(j)}\|\}$$

This defines a **perpendicular bisector**—the locus of points equidistant from two reference points. This is pure geometry, dating back to Euclid.

The resulting partition is called a **Voronoi diagram**, named after mathematician Georgy Voronoi (1908). These structures appear throughout nature: cell membranes, giraffe skin patterns, and even the territorial divisions of competing ant colonies all follow Voronoi geometry.

2.2 The Effect of Increasing K: Smoothing Through Democracy

When $K = 1$, every training point is a dictator over its territory. But what happens when we consult multiple neighbors and take a vote?

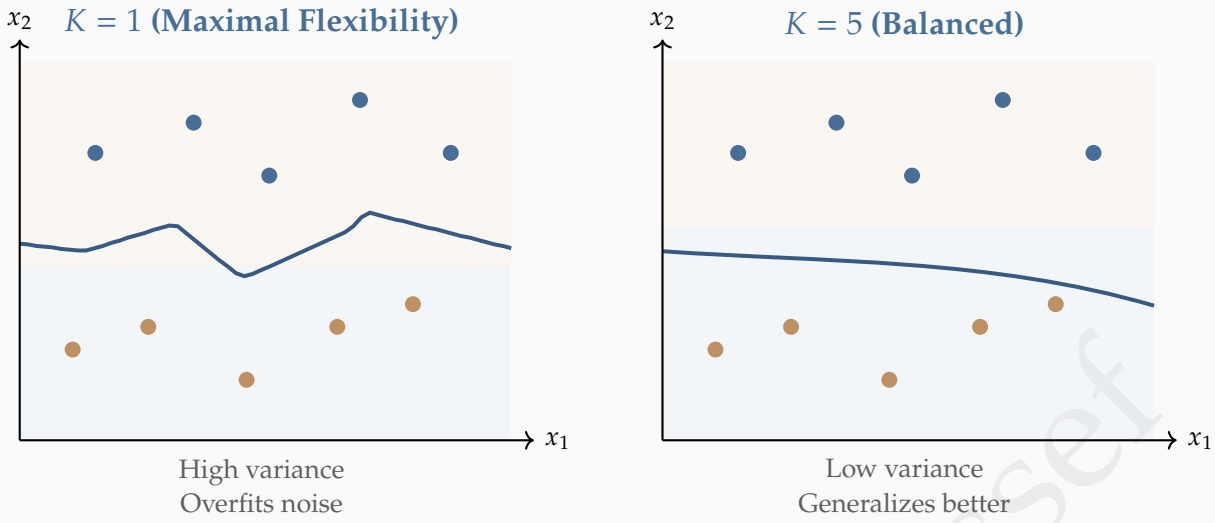


Figure 3: ***The smoothing effect of larger*** K**.** *Both plots use identical training data (10 points). Left:* $K = 1$ *produces an erratic boundary that chases every point's influence. Right:* $K = 5$ *averages over neighbors, yielding a smooth, stable boundary. This is the bias-variance tradeoff in geometric form.*

Democracy vs. Autocracy in Classification

Think of each value of K as a different voting system:

- $K = 1$**:** Autocracy—the single nearest neighbor dictates the prediction. Fast, but one outlier can completely mislead you.
- $K = 5$**:** Small committee—enough voices to smooth out individual quirks, but still responsive to local structure.
- $K = m$ **(all training points):** Universal suffrage—everyone votes. But now distant, irrelevant examples dilute the signal. You'll just predict the majority class everywhere.

The art of K-NN lies in finding the democratic sweet spot: enough neighbors to resist noise, but not so many that you lose local sensitivity.

3 The Algorithm: From Intuition to Implementation

Having visualized how K-NN partitions space, let's formalize the procedure. The algorithm is remarkably simple—just three steps—yet its power emerges from systematic application of one core principle: *similarity predicts behavior*.

3.1 Problem Setup and Notation

We have a labeled training dataset:

$$\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$$

where:

- $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ is an n-dimensional feature vector
- $y^{(i)}$ is the corresponding label (discrete for classification, continuous for regression)

- m is the number of training examples

Goal: Given a new query point $\boldsymbol{x}_q$, predict its label $\hat{y}_q$.

3.2 The Three-Step Algorithm

K-Nearest Neighbors Algorithm

Input: Training set $\mathcal{D}$, query $\boldsymbol{x}_q$, hyperparameter K, distance metric $d(\cdot, \cdot)$

Step 1: Compute Distances
Calculate the distance from $\boldsymbol{x}_q$ to every training point:

$$d_i = d(\boldsymbol{x}_q, \boldsymbol{x}^{(i)}) \quad \text{for } i = 1, 2, \ldots, m$$

Step 2: Identify K Nearest Neighbors
Sort the distances and select the indices of the K smallest:

$$\mathcal{N}_K(\boldsymbol{x}_q) = \{\text{indices of the } K \text{ training points with smallest } d_i\}$$

Step 3: Aggregate Predictions
For Classification: Majority vote among the K neighbors

$$\hat{y}_q = \arg\max_c \sum_{i \in \mathcal{N}_K(\boldsymbol{x}_q)} \mathbb{1}[y^{(i)} = c]$$

where $\mathbb{1}[\cdot]$ is the indicator function (1 if true, 0 otherwise).
For Regression: Average the target values

$$\hat{y}_q = \frac{1}{K} \sum_{i \in \mathcal{N}_K(\boldsymbol{x}_q)} y^{(i)}$$

Output: Predicted label $\hat{y}_q$

3.3 A Concrete Worked Example

Let's trace through the algorithm step-by-step with actual numbers.

K-NN Classification with $K = 3$

Training Data: Six points in $\mathbb{R}^2$

ID	x_1	x_2	Class
A	1	2	Red
B	2	3	Red
C	3	3	Blue
D	6	5	Blue
E	7	7	Blue
F	8	6	Red

Query: Classify $\boldsymbol{x}_q = (5, 4)$ using $K = 3$ and Euclidean distance.

Step 1: Compute Distances

Using the Euclidean distance formula

$$d(\boldsymbol{x}_q, \boldsymbol{x}^{(i)}) = \sqrt{(x_{q,1} - x_1^{(i)})^2 + (x_{q,2} - x_2^{(i)})^2},$$

we get:

$$\begin{aligned}
d(q, A) &= \sqrt{(5-1)^2 + (4-2)^2} = \sqrt{20} = 4.47, \\
d(q, B) &= \sqrt{(5-2)^2 + (4-3)^2} = \sqrt{10} = 3.16, \\
d(q, C) &= \sqrt{(5-3)^2 + (4-3)^2} = \sqrt{5} = 2.24, \\
d(q, D) &= \sqrt{(5-6)^2 + (4-5)^2} = \sqrt{2} = 1.41, \\
d(q, E) &= \sqrt{(5-7)^2 + (4-7)^2} = \sqrt{13} = 3.61, \\
d(q, F) &= \sqrt{(5-8)^2 + (4-6)^2} = \sqrt{13} = 3.61.
\end{aligned}$$

Step 2: Sort and Select $K = 3$ Nearest

Rank	Neighbor	Distance	Class
1	D	1.41	Blue
2	C	2.24	Blue
3	B	3.16	Red
4	E	3.61	Blue
5	F	3.61	Red
6	A	4.47	Red

The 3-nearest neighbors are:

$$\mathcal{N}_3(\boldsymbol{x}_q) = \{D, C, B\}.$$

Step 3: Majority Vote

Count votes:

- **Blue:** 2 votes (from D and C)
- **Red:** 1 vote (from B)

Prediction: $\hat{y}_q$ = **Blue** (majority wins)

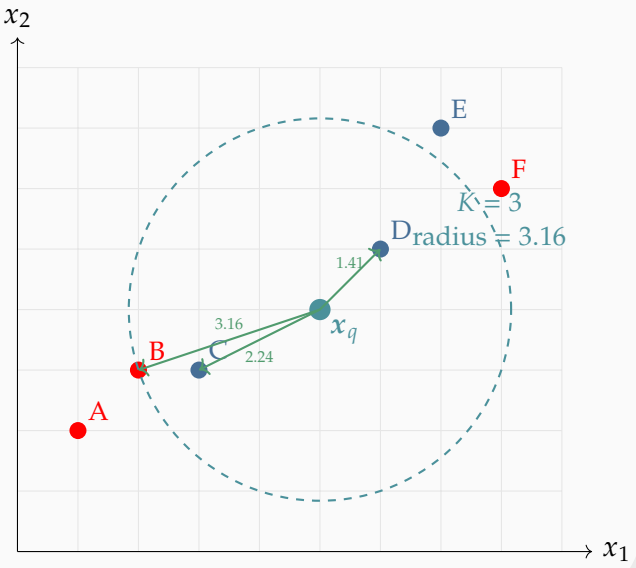


Figure 4: *Geometric view of the classification. The query point lies within distance 3.16 of three training points. Two neighbors are Blue, one is Red—Blue wins the vote.*

What Makes This Work?

Observe that the query point $(5, 4)$ lies closer to the Blue cluster than the Red cluster. K-NN does not explicitly learn boundaries or parameters. It simply consults nearby labeled points. By relying on local geometry, it naturally adapts to the structure of the data without building a model.

3.4 K-NN for Regression: Predicting Continuous Values

The algorithm adapts seamlessly to regression by replacing voting with averaging.

House Price Prediction

Task: Predict the price of a house with 1850 sq ft based on similar houses.
Training Data:

House	Area (sq ft)	Price ($1000s)
H_1	1500	200
H_2	1700	240
H_3	1800	260
H_4	1900	280
H_5	2200	350

Query: House with 1850 sq ft, using $K = 3$

Step 1: Compute Distances (absolute difference)

$$d(1850, 1500) = 350$$
$$d(1850, 1700) = 150$$

$$d(1850, 1800) = 50$$
$$d(1850, 1900) = 50$$
$$d(1850, 2200) = 350$$

Step 2: Select 3-Nearest Nearest neighbors: H_3 (50), H_4 (50), H_2 (150)

Step 3: Average Prices

$$\hat{y}_q = \frac{260 + 280 + 240}{3} = \frac{780}{3} = 260 \text{ thousand dollars}$$

Prediction: \$260,000

Interpretation: The model predicts a price that's the average of similar-sized houses. If we had used $K = 1$, we'd predict \$260k or \$280k (tie between H_3 and H_4). With $K = 5$ (all houses), we'd get \$266k—less sensitive to local patterns.

4 Distance Metrics: Defining Similarity

K-NN's notion of "nearest" depends entirely on how we measure distance. This choice profoundly shapes predictions—it's not just a technical detail, it's a philosophical statement about what makes examples similar.

4.1 The Standard Metrics

Common Distance Functions

For points $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n$:

Euclidean Distance (L_2 norm):

$$d_2(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{\sum_{j=1}^{n} (x_j - x'_j)^2} = \|\boldsymbol{x} - \boldsymbol{x}'\|_2$$

The straight-line "as the crow flies" distance. Most common in practice.

Manhattan Distance (L_1 norm):

$$d_1(\boldsymbol{x}, \boldsymbol{x}') = \sum_{j=1}^{n} |x_j - x'_j| = \|\boldsymbol{x} - \boldsymbol{x}'\|_1$$

Also called "taxicab" or "city block" distance—the path a taxi takes on a grid. More robust to outliers.

Minkowski Distance (L_p norm):

$$d_p(\boldsymbol{x}, \boldsymbol{x}') = \left(\sum_{j=1}^{n} |x_j - x'_j|^p \right)^{1/p}$$

Generalizes both: $p = 1$ gives Manhattan, $p = 2$ gives Euclidean, $p \to \infty$ gives Chebyshev (maximum coordinate difference).

4.2 Geometric Intuition: The Shape of Distance

Different metrics create different notions of "equidistant." This has profound implications for which points K-NN considers neighbors.

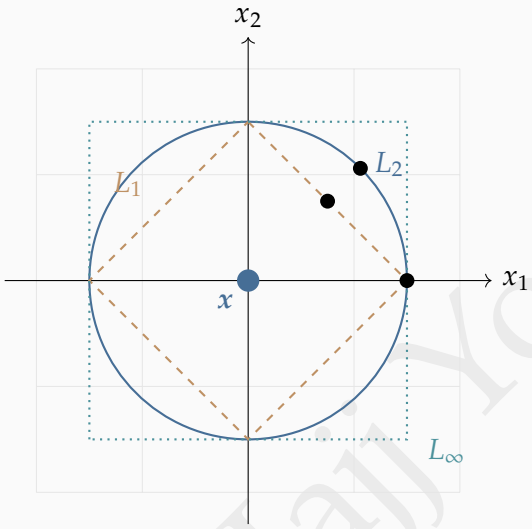


Figure 5: ***The geometry of distance metrics.*** *All points on each curve are equidistant from the origin under their respective metric. Euclidean (L_2) forms circles, Manhattan (L_1) forms diamonds, Chebyshev (L_∞) forms squares. The point $(1.5, 0)$ is equally far under all three metrics, but $(1, 1)$ is closer under L_∞ than L_2. Your choice of metric determines which training points qualify as "neighbors."*

When to Use Each Metric

Euclidean (L_2):

- Default choice for most applications
- Best when all features are on comparable scales
- Assumes straight-line distance is meaningful
- Example: Physical measurements (height, weight)

Manhattan (L_1):

- More robust to outliers (no squaring amplifies large differences)
- Natural for grid-based problems
- Better when movement is constrained to axes
- Example: City navigation, warehouse logistics

Specialized Metrics:

- **Cosine similarity:** For high-dimensional sparse data (text, user preferences)
- **Hamming distance:** For categorical/binary features

- **Mahalanobis distance:** Accounts for feature correlations

4.3 The Critical Importance of Feature Scaling

Here's a shocking fact: K-NN will fail catastrophically if you don't scale your features. This isn't a minor technicality—it's the difference between a useful model and garbage.

Why K-NN is Scale-Sensitive

Consider predicting house prices with two features:

- **Square footage:** ranges from 800 to 3500 (span: 2700)
- **Number of bedrooms:** ranges from 1 to 5 (span: 4)

When computing Euclidean distance:

$$d = \sqrt{(\Delta\text{sqft})^2 + (\Delta\text{bedrooms})^2}$$

A 100 sq ft difference contributes $100^2 = 10{,}000$ to the squared distance. A 1 bedroom difference contributes only $1^2 = 1$. The distance is completely dominated by square footage—bedrooms become irrelevant, even if they're equally important for pricing!
The problem: K-NN measures geometric distance in raw feature space. Features with larger numerical ranges automatically dominate the distance calculation, regardless of their predictive importance.

Feature Scaling: Before and After

Scenario: Classify a house using 2 features and $K = 1$

House	Sq. Feet	Bedrooms	Class
A	1500	3	Affordable
B	1600	2	Expensive
Query	1550	3	?

Without Standardization:

$$d(Q, A) = \sqrt{(1550 - 1500)^2 + (3 - 3)^2} = \sqrt{2500 + 0} = 50.0$$
$$d(Q, B) = \sqrt{(1550 - 1600)^2 + (3 - 2)^2} = \sqrt{2500 + 1} = 50.01$$

The nearest neighbor is A (by 0.01!). But notice: the 1-bedroom difference contributed only 0.02% to the distance difference. Bedrooms are effectively ignored.

With Standardization: $z_j = \frac{x_j - \mu_j}{\sigma_j}$
Using training statistics: $\mu_{\text{sqft}} = 1550$, $\sigma_{\text{sqft}} = 50$ $\mu_{\text{bed}} = 2.5$, $\sigma_{\text{bed}} = 0.5$

Standardized features:

$$A' : \left[\frac{1500-1550}{50}, \frac{3-2.5}{0.5}\right] = [-1.0, 1.0]$$
$$B' : \left[\frac{1600-1550}{50}, \frac{2-2.5}{0.5}\right] = [1.0, -1.0]$$
$$Q' : \left[\frac{1550-1550}{50}, \frac{3-2.5}{0.5}\right] = [0.0, 1.0]$$

Distances in standardized space:

$$d(Q', A') = \sqrt{(0-(-1))^2 + (1-1)^2} = \sqrt{1+0} = 1.0$$
$$d(Q', B') = \sqrt{(0-1)^2 + (1-(-1))^2} = \sqrt{1+4} = 2.24$$

Result: Now A is clearly closer (1.0 vs. 2.24). The shared bedroom count matters! Prediction: **Affordable**.
Critical Rule: Always compute μ_j, σ_j on training data only, then apply to test data. This prevents information leakage.

Standardization Procedure

Step 1: Compute statistics on training set only

$$\mu_j = \frac{1}{m_{\text{train}}} \sum_{i=1}^{m_{\text{train}}} x_j^{(i)}, \quad \sigma_j = \sqrt{\frac{1}{m_{\text{train}}} \sum_{i=1}^{m_{\text{train}}} (x_j^{(i)} - \mu_j)^2}$$

Step 2: Transform both training and test sets using these statistics

$$z_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

Step 3: Apply K-NN to transformed features $\boldsymbol{z}^{(i)}$ instead of raw $\boldsymbol{x}^{(i)}$
Why this ordering? If we computed statistics on test data, we'd be "peeking" at information that should be unseen. This violates the fundamental principle of honest evaluation.

5 Choosing K: The Bias-Variance Tradeoff Revisited

The hyperparameter K is K-NN's complexity knob. Turn it one way, you get a flexible model that tracks every wiggle in the data. Turn it the other way, you get a smooth model that might miss important patterns. This is the bias-variance tradeoff in its purest form.

5.1 The Spectrum of Complexity

How K Controls Model Behavior

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

- **Low bias:** Flexible enough to capture complex patterns
- **High variance:** Extremely sensitive to individual training points
- **Risk:** Overfitting—memorizes noise and outliers
- **Boundary:** Jagged, intricate, wraps around each point
- **Training accuracy:** Often 100% (memorization)

Large K (e.g., $K = m/2$):

- **High bias:** May be too smooth to capture real patterns
- **Low variance:** Stable, robust to individual points
- **Risk:** Underfitting—averages away genuine structure
- **Boundary:** Very smooth, potentially too simple
- **Extreme case:** $K = m$ always predicts majority class

Optimal K:

- Balances local responsiveness with stability
- Typically found through cross-validation
- Often an odd number (avoids ties in binary classification)
- Dataset-dependent: no universal "best" value

5.2 Visualizing the Bias-Variance Tradeoff

Recall from Chapter 3 that total error decomposes as:

$$\text{Expected Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

For K-NN, K directly controls this tradeoff:

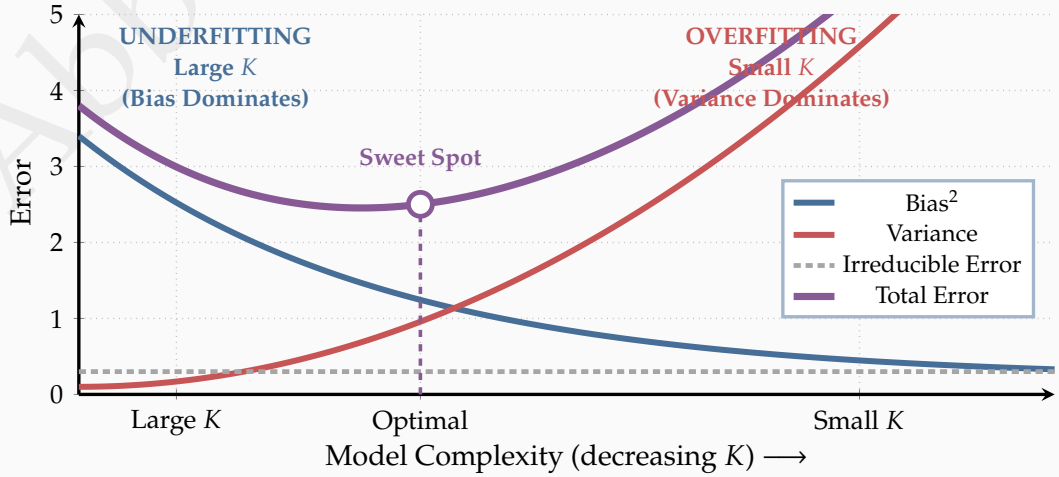


Figure 6: ***The K complexity curve.*** *Smaller K makes K-NN more flexible with higher variance; larger K increases bias. The optimal K balances this trade-off.*

Understanding Through Extremes

$K = 1$ **(Minimum bias, maximum variance):**

- Each training point is a dictator over its Voronoi cell
- Training accuracy: 100% (every point correctly classifies itself)
- Test performance: Often poor—one mislabeled point or outlier pollutes its entire region

$K = m$ **(Maximum bias, minimum variance):**

- Every query consults all training points
- Prediction: Always the majority class (ignores query location entirely!)
- Training accuracy: Whatever the class imbalance dictates
- Test performance: Terrible—model learned nothing about local structure

The art lies between these extremes.

5.3 Finding the Optimal K: Cross-Validation

We can't use training accuracy to choose K—it would always favor $K = 1$. We need an honest estimate of generalization performance.

k-Fold Cross-Validation for K Selection

Goal: Find the value of K that minimizes validation error
Procedure:

1. Define candidate values: $K \in \{1, 3, 5, 7, 9, 11, 15, 21, \ldots\}$
2. Split training data into k folds (typically $k = 5$ or $k = 10$)
3. For each candidate K:
   - For each fold $i = 1, \ldots, k$:
     - Use fold i as validation set
     - Train on remaining $k - 1$ folds
     - Compute validation accuracy/error
   - Average validation scores: $\text{CV}_K = \frac{1}{k} \sum_{i=1}^{k} \text{Score}_i$
4. Select $K^* = \arg\max_K \text{CV}_K$ (or $\arg\min$ for error metrics)
5. Retrain on full training set with K^*
6. Evaluate final model on held-out test set

Critical distinction: The k in "k-fold" is the number of validation folds. The K in "K-NN" is the number of neighbors. Don't confuse them!

6 Walking Through Cross-Validation: A Complete Example

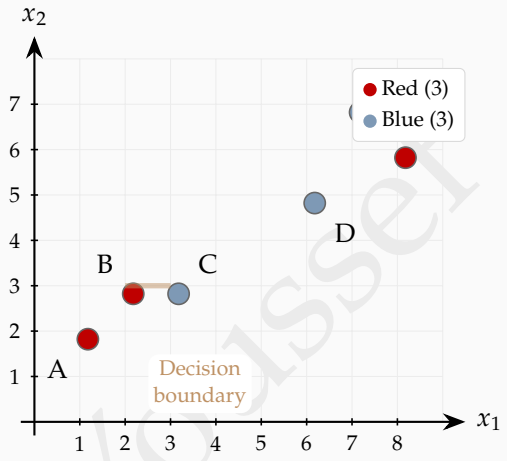
Before examining large datasets, let's trace cross-validation step-by-step using our familiar six-point dataset. This microscopic view reveals exactly how cross-validation provides honest performance estimates.

6.1 The Setup: Choosing K with Limited Data

Training Dataset

ID	x_1	x_2	Class
A	1	2	Red
B	2	3	Red
C	3	3	Blue
D	6	5	Blue
E	7	7	Blue
F	8	6	Red

Hyperparameter Candidates:
$K \in \{1, 3, 5\}$



Class overlap at boundary; isolated outlier F

Why Leave-One-Out Cross-Validation?

The Challenge: With only 6 examples, traditional train/validation splits leave too few points for reliable evaluation.

The Solution: Leave-One-Out CV (LOOCV)—each point becomes the test set exactly once while training on the remaining 5 points.

Advantages	Disadvantages
• Maximizes training data usage	• Computationally expensive
• Nearly unbiased estimate	• High variance in estimates
• Deterministic results	• Impractical for large datasets
• Ideal for small datasets	• Models highly correlated

Best for: Small datasets ($m < 100$) where every example counts for reliable validation.

Cross-Validation Procedure

For each fold $i = 1, \ldots, 6$**:**

1. **Split:** Hold out test point $\boldsymbol{x}^{(i)}$ with true label $y^{(i)}$; train on remaining 5 points
2. **For each candidate** $K \in \{1, 3, 5\}$**:**
   - Compute distances from $\boldsymbol{x}^{(i)}$ to all training points
   - Select K nearest neighbors

- Vote on class label $\hat{y}_K^{(i)}$ via majority rule
- Record correctness: $\mathbb{1}[\hat{y}_K^{(i)} = y^{(i)}]$

3. **Aggregate:** After all folds, compute accuracy for each K:

$$\text{CV-Accuracy}_K = \frac{1}{6}\sum_{i=1}^{6} \mathbb{1}[\hat{y}_K^{(i)} = y^{(i)}]$$

4. **Select:** Choose optimal hyperparameter:

$$K^* = \arg\max_{K\in\{1,3,5\}} \text{CV-Accuracy}_K$$

6.2 Fold-by-Fold Analysis

We now examine each fold in detail, tracking predictions across all K values.

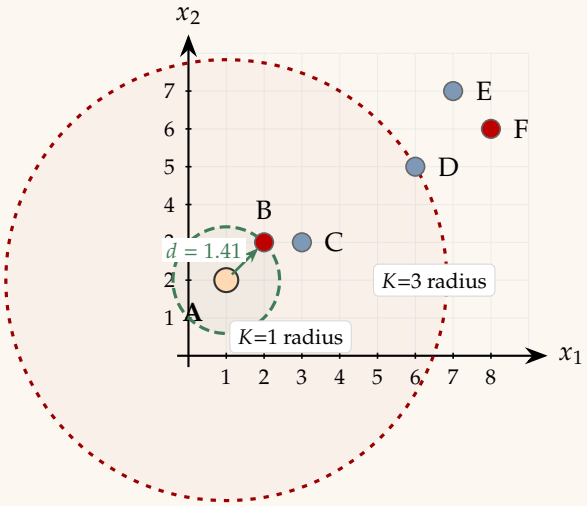
Fold 1: Prediction for Point A

Fold 1

Test Sample: $A = (1, 2)$ **True Class:** **Red**

Nearest Neighbors

Point	Distance	Class
B	**1.41**	Red
C	2.24	Blue
D	5.83	Blue
E	7.81	Blue
F	8.06	Red



Predictions by K:

- $K = 1 \rightarrow$ Nearest: B $\rightarrow$ Predict **Red** ✓
- $K = 3 \rightarrow \{B, C, D\} \rightarrow$ Vote: 1R vs 2B $\rightarrow$ Predict **Blue** ×
- $K = 5 \rightarrow$ All neighbors $\rightarrow$ Vote: 2R vs 3B $\rightarrow$ Predict **Blue** ×

Interpretation: Only $K = 1$ correctly predicts A. Larger K values are overwhelmed by the distant Blue cluster, demonstrating over-smoothing.

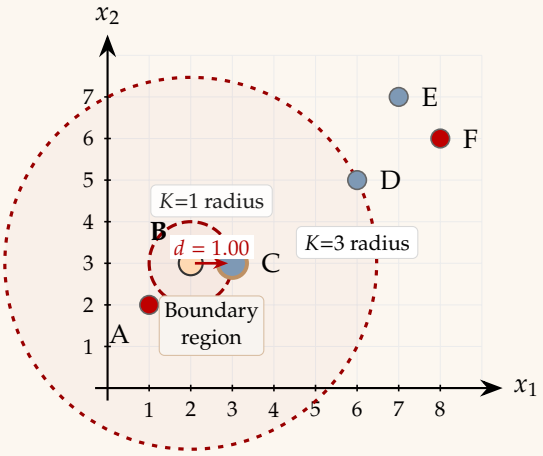
Fold 2: Prediction for Point B

Fold 2

Test Sample: $B = (2, 3)$ **True Class: Red**

Nearest Neighbors

Point	Distance	Class
C	**1.00**	Blue
A	1.41	Red
D	4.47	Blue
E	6.40	Blue
F	6.71	Red



Predictions by K:

- $K = 1 \rightarrow$ Nearest: C $\rightarrow$ Predict **Blue** ×
- $K = 3 \rightarrow \{C, A, D\} \rightarrow$ Vote: 1R vs 2B $\rightarrow$ Predict **Blue** ×
- $K = 5 \rightarrow$ All neighbors $\rightarrow$ Vote: 2R vs 3B $\rightarrow$ Predict **Blue** ×

Interpretation: Complete failure across all K values. Point B sits at the class boundary; its nearest neighbor C is Blue, causing consistent misclassification regardless of K.

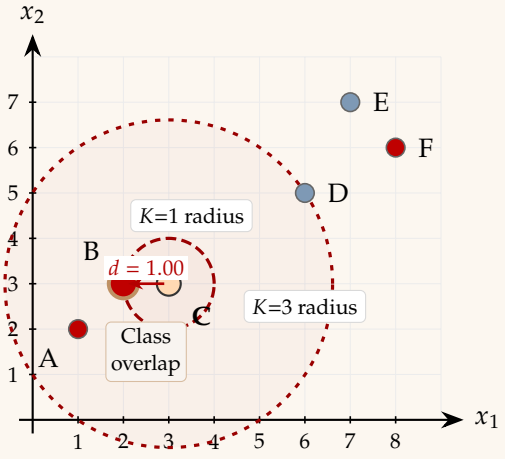
Fold 3: Prediction for Point C

Fold 3

Test Sample: $C = (3, 3)$ **True Class: Blue**

Nearest Neighbors

Point	Distance	Class
B	**1.00**	Red
A	2.24	Red
D	3.61	Blue
E	5.66	Blue
F	5.83	Red



Predictions by K:

- $K = 1 \rightarrow$ Nearest: B $\rightarrow$ Predict **Red** ×
- $K = 3 \rightarrow \{B, A, D\} \rightarrow$ Vote: 2R vs 1B $\rightarrow$ Predict **Red** ×
- $K = 5 \rightarrow$ All neighbors $\rightarrow$ Vote: 3R vs 2B $\rightarrow$ Predict **Red** ×

Interpretation: Complete failure. Point C (Blue) is surrounded by Red neighbors, revealing genuine class overlap at the decision boundary.

The Boundary Pattern Emerges

Points B and C are only 1.0 units apart yet belong to different classes—they sit precisely at the decision boundary where Red and Blue territories meet. This spatial proximity with label disagreement represents **irreducible error**.

Key insight: Cross-validation reveals not just *which* K is best, but *whether the problem is learnable at all*. Poor performance across all K values signals fundamental challenges: overlapping classes, insufficient features, or inherent noise in the labeling process.

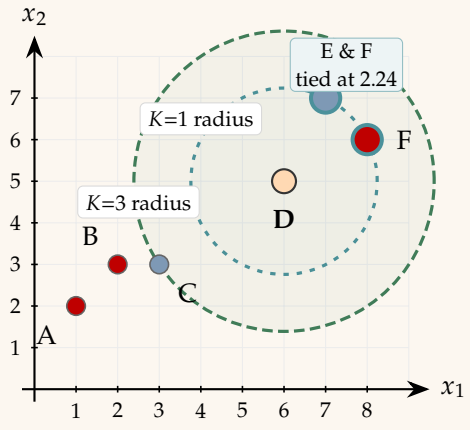
Fold 4: Prediction for Point D

Fold 4

Test Sample: $D = (6, 5)$ **True Class:** Blue

Nearest Neighbors

Point	Distance	Class
E	**2.24**	Blue
F	**2.24**	Red
C	3.61	Blue
B	4.47	Red
A	5.83	Red



Predictions by K:

- $K = 1 \rightarrow$ Tie: E and F (both 2.24) $\rightarrow$ Tie-break: F $\rightarrow$ Predict **Red** ×
- $K = 3 \rightarrow \{E, F, C\} \rightarrow$ Vote: 2B vs 1R $\rightarrow$ Predict **Blue** ✓
- $K = 5 \rightarrow$ All neighbors $\rightarrow$ Vote: 2B vs 3R $\rightarrow$ Predict **Red** ×

Interpretation: Only $K = 3$ correctly predicts D. For $K = 1$, E and F are equidistant, and arbitrary tie-breaking selects the wrong class. For $K = 5$, distant Red points dominate. This demonstrates how K>1 can resolve ambiguous cases through neighbor pooling.

Handling Ties in K-NN

Common strategies when multiple neighbors are equidistant:

1. **Random selection** among tied neighbors
2. **Arbitrary ordering** (e.g., by index or alphabetical)
3. **Increase** K until the tie breaks naturally
4. **Weight by class priors** from the training set
5. **Use odd** K to reduce voting ties (doesn't help distance ties)

Practical tip: Add tiny random noise ($\epsilon \sim 10^{-10}$) to features during distance computation, or use weighted voting schemes where closer neighbors have higher influence.

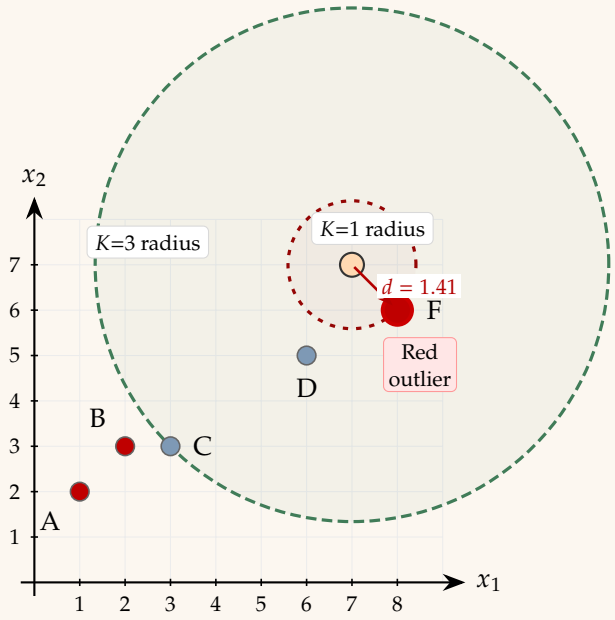
Fold 5: Prediction for Point E

Fold 5

Test Sample: $E = (7, 7)$ **True Class:** Blue

Nearest Neighbors

Point	Distance	Class
F	**1.41**	Red
D	2.24	Blue
C	5.66	Blue
B	6.40	Red
A	7.81	Red



Predictions by K:

- $K = 1 \rightarrow$ Nearest: F $\rightarrow$ Predict **Red** ×
- $K = 3 \rightarrow \{F, D, C\} \rightarrow$ Vote: 1R vs 2B $\rightarrow$ Predict **Blue** ✓
- $K = 5 \rightarrow$ All neighbors $\rightarrow$ Vote: 3R vs 2B $\rightarrow$ Predict **Red** ×

Interpretation: Only $K = 3$ correctly predicts E. The nearest neighbor F is a Red outlier, misleading $K = 1$. For $K = 5$, the majority vote is dominated by distant Red points. This demonstrates the outlier resistance of moderate K values.

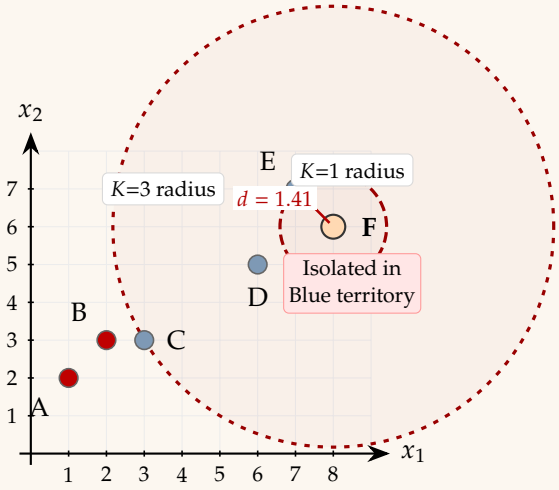
Fold 6: Prediction for Point F

Fold 6

Test Sample: $F = (8, 6)$ **True Class: Red**

Nearest Neighbors

Point	Distance	Class
E	**1.41**	Blue
D	2.24	Blue
C	5.83	Blue
B	6.71	Red
A	8.06	Red



Predictions by K:

- $K = 1 \rightarrow$ Nearest: E $\rightarrow$ Predict **Blue** $\times$
- $K = 3 \rightarrow \{E, D, C\} \rightarrow$ Vote: 3B vs 0R $\rightarrow$ Predict **Blue** $\times$
- $K = 5 \rightarrow$ All neighbors $\rightarrow$ Vote: 3B vs 2R $\rightarrow$ Predict **Blue** $\times$

Interpretation: Complete failure across all K values. Point F is a Red outlier isolated in Blue territory. All three nearest neighbors are Blue—no local voting method can overcome this spatial isolation.

6.3 Final Verdict: Aggregating All Folds

After examining all six folds, we aggregate the results to select the optimal K:

Fold	Point	True Class	$K = 1$	$K = 3$	$K = 5$
1	A (1,2)	Red	✓	×	×
2	B (2,3)	Red	×	×	×
3	C (3,3)	Blue	×	×	×
4	D (6,5)	Blue	×	✓	×
5	E (7,7)	Blue	×	✓	×
6	F (8,6)	Red	×	×	×
Correct Predictions			1/6	2/6	0/6
CV Accuracy			**16.7%**	**33.3%**	**0.0%**

Cross-Validation Verdict

The Winner: $K = 3$ (33.3% accuracy). It strikes the best balance, avoiding the brittle overfitting of $K = 1$ and the excessive smoothing of $K = 5$.

Two Critical Lessons:

1. **Detection of Overfitting:** $K = 1$ has 100% *training* accuracy (perfect memory) but only 16.7% *CV* accuracy. Cross-validation successfully exposed this generalization failure.
2. **Diagnosing Data Quality:** When performance is poor across *all* hyperparameters ($K = 1, 3, 5$), the issue is usually the data itself. Here, class overlap (B vs C) and outliers (F) limit the maximum possible accuracy, regardless of the model.

From Toy Example to Real World

While this 6-point example illustrates the *mechanics* of LOOCV, real-world applications differ in scale:

- **Strategy:** With larger datasets ($m > 100$), prefer **5-fold or 10-fold CV** rather than Leave-One-Out to reduce computation and variance.
- **Search Range:** Test K values roughly up to $\sqrt{m}$.
- **Best Practice:** Always use **stratified splits** to ensure every fold has a representative ratio of classes.

Bottom line: The calculations scale directly, but statistical reliability improves drastically with more data.

7 A Complete Case Study: The Two Moons Dataset

To see K-NN's power and limitations in action, let's work through a comprehensive example using the Two Moons dataset—a deliberately challenging problem that showcases K-NN's ability to learn complex, non-linear decision boundaries.

7.1 The Challenge: Non-Linear Separability

The Two Moons dataset consists of 200 points arranged in two interleaving crescent shapes. Each crescent represents a different class, creating a problem where no straight line can separate the classes—linear classifiers would fail miserably.

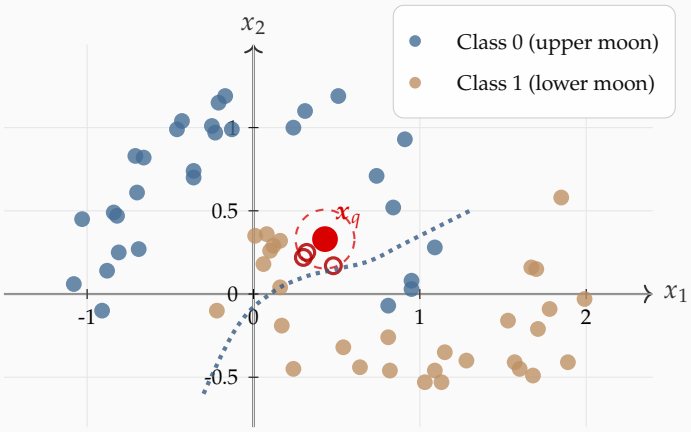


Figure 7: ***The Two Moons dataset.*** *Two interleaving crescents form a non-linearly separable classification problem. The query point $\boldsymbol{x}_q = [0.43, 0.33]^T$ (red) lies near the decision boundary. Its three nearest neighbors (highlighted with thick circles) are all Class 1, leading to confident prediction. The dotted curve shows where K-NN's prediction transitions between classes—notice how it adapts to the crescent geometry without explicit programming.*

7.2 Why This Dataset Matters

The Linear Classifier's Dilemma

Consider what a linear classifier (e.g., logistic regression) would do:

- A line can only create two half-planes
- Any line separating some blue from orange also mixes many points incorrectly
- Best achievable accuracy: approximately 50% (no better than random!)

But K-NN has no such limitation. It creates boundaries that:

- Curve and bend to follow the crescent shapes
- Adapt locally to point configurations
- Require no feature engineering or transformation
- Emerge naturally from geometric proximity

This showcases K-NN's fundamental strength: **learning arbitrarily complex boundaries without parametric assumptions**.

7.3 Detailed Classification Example

Let's classify the boundary point $\boldsymbol{x}_q = [0.43, 0.33]^T$ using $K = 3$.

Step-by-Step Classification

Query: $\boldsymbol{x}_q = [0.43, 0.33]^T$

Step 1: Compute Distances

Nearest neighbors (from 200 points):

Point	Coords	Dist	Class
$\boldsymbol{x}^{(1)}$	(0.32, 0.25)	0.136	1
$\boldsymbol{x}^{(2)}$	(0.48, 0.17)	0.168	1
$\boldsymbol{x}^{(3)}$	(0.30, 0.22)	0.170	1
$\boldsymbol{x}^{(4)}$	(0.16, 0.32)	0.271	1
$\boldsymbol{x}^{(5)}$	(0.24, 1.00)	0.676	0

Example distance:

$$d_1 = \sqrt{(0.11)^2 + (0.08)^2} = \sqrt{0.0185} = 0.136$$

Step 2: Choose $K = 3$

$\mathcal{N}_3(\boldsymbol{x}_q) = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}\}$ All are Class 1.

Step 3: Majority Vote

Class 0: 0 votes Class 1: 3 votes

Prediction: $\hat{y}_q$ = Class 1 (unanimous)

Confidence note: A 3–0 vote indicates high certainty; a 2–1 split would suggest proximity to the boundary.

7.4 The Dramatic Effect of K: Boundary Evolution

Now let's see how different values of K create vastly different decision boundaries on the same data.

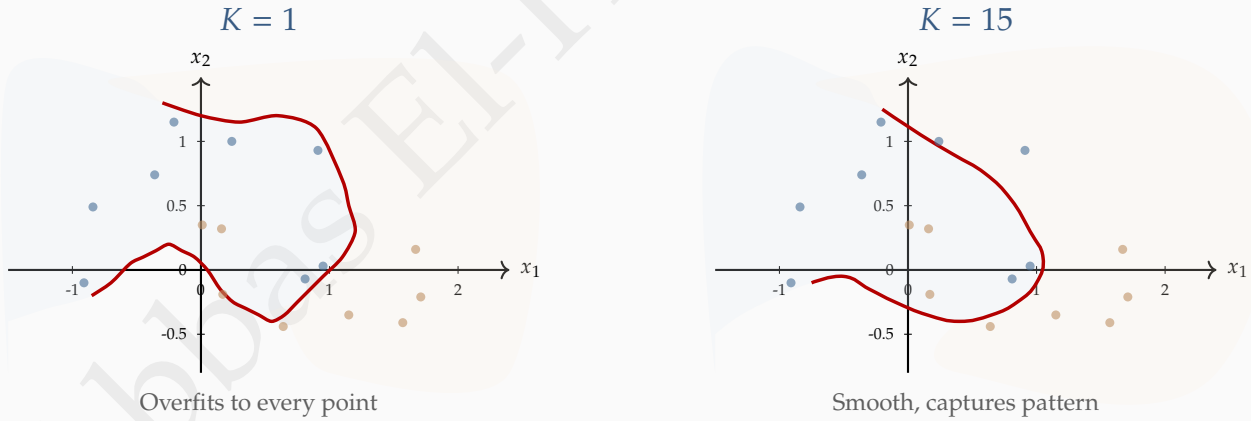


Figure 8: *K **transforms the decision landscape.** Both trained on identical data (16 of 200 points shown). **Left (**K = 1**):** Erratic boundary overfits to individual points. **Right (**K = 15**):** Smooth boundary captures true pattern. Red curves show decision boundaries.*

Performance Analysis: Two Moons

Cross-validation on the full 200-point dataset reveals the optimal K:

K	Test Accuracy	Interpretation
1	98.0%	Overfits—jagged boundaries
3	99.0%	Excellent balance
5	99.0%	Optimal—smooth yet responsive
10	**99.5%**	**Peak performance**
15	98.0%	Slight underfit, still captures pattern
31	95.5%	Too smooth—misses fine structure

Key insight: The Two Moons problem demonstrates K-NN's remarkable ability to learn **complex non-linear boundaries without feature engineering**. A linear classifier would fail (50% accuracy), but K-NN naturally discovers the curved separation through local voting.
This is why K-NN remains relevant: when you have non-linear patterns and sufficient data, its geometric approach can outperform sophisticated parametric models.

8 Weighted Voting: Distance-Aware Predictions

Standard K-NN gives every neighbor an equal vote. But intuitively, closer points should matter more: a neighbor at distance 0.1 provides stronger evidence than one at distance 5.0.

8.1 The Motivation for Weighting

Consider classifying an email as spam or ham using $K = 5$ nearest neighbors:

Neighbor	Distance	Class
1	0.3	Spam
2	0.5	Spam
3	2.1	Ham
4	2.8	Spam
5	3.2	Ham

The unweighted prediction counts votes: 3 for Spam and 2 for Ham, so we predict Spam. But notice that the two closest neighbors (distances 0.3 and 0.5) are both Spam, while the Ham votes come from much more distant points (distances 2.1 and 3.2). This suggests that weighting by proximity might give a more confident prediction.

8.2 Inverse Distance Weighting

Weighted K-NN

Assign each neighbor i a weight inversely proportional to squared distance:

$$w_i = \frac{1}{d_i^2 + \epsilon}, \qquad \epsilon = 10^{-6} \text{ (prevents division by zero)}$$

Classification: Sum weights for each class, choose the class with highest total

$$\hat{y}_q = \arg\max_c \sum_{i \in \mathcal{N}_K} w_i \mathbb{1}[y^{(i)} = c]$$

Regression: Weighted average of neighbor values

$$\hat{y}_q = \frac{\sum_{i \in \mathcal{N}_K} w_i \, y^{(i)}}{\sum_{i \in \mathcal{N}_K} w_i}$$

The inverse square law creates natural decay: doubling distance reduces weight by 4×, tripling reduces weight by 9×.

Spam Classification with Weights

We compute weights for each neighbor using $w_i = 1/d_i^2$:

Neighbor	Distance	Calculation	Weight	Class
1	0.3	$1/0.3^2$	11.11	Spam
2	0.5	$1/0.5^2$	4.00	Spam
3	2.1	$1/2.1^2$	0.23	Ham
4	2.8	$1/2.8^2$	0.13	Spam
5	3.2	$1/3.2^2$	0.10	Ham

Now we sum weights for each class:

$$\begin{aligned} \text{Spam:} \quad & 11.11 + 4.00 + 0.13 = 15.24 \\ \text{Ham:} \quad & 0.23 + 0.10 = 0.33 \end{aligned}$$

The weighted prediction is Spam with a score of 15.24 versus 0.33 for Ham. This represents 46× stronger evidence than the unweighted 3-versus-2 vote, making the prediction far more confident because the closest neighbors dominate the decision.

House Price Regression

Suppose we want to predict the price of a 2000 sq ft house using $K = 3$ neighbors:

House	Distance	Calculation	Weight	Price ($k)
A	0.8	$1/0.8^2$	1.56	350
B	1.2	$1/1.2^2$	0.69	380
C	1.5	$1/1.5^2$	0.44	420

The unweighted prediction averages all three prices equally:

$$\hat{y}_{\text{unweighted}} = \frac{350 + 380 + 420}{3} = \frac{1150}{3} = 383.3 \text{ thousand}$$

For the weighted prediction, we compute:

$$\hat{y}_{\text{weighted}} = \frac{1.56 \times 350 + 0.69 \times 380 + 0.44 \times 420}{1.56 + 0.69 + 0.44}$$
$$= \frac{546 + 262.2 + 184.8}{2.69}$$
$$= \frac{993}{2.69} = 369.1 \text{ thousand}$$

The weighted prediction is \$369k, which is \$14k lower than the unweighted \$383k. This makes sense because the closest house (distance 0.8) costs only \$350k, and weighting gives it more influence. The weighted approach is typically more accurate when neighbors are at varying distances.

Weighting helps most when distances vary widely or when data is irregularly distributed. Standard voting is simpler and sufficient when neighbors are roughly equidistant or when computational speed is critical. Most machine learning libraries, including scikit-learn's `KNeighborsClassifier`, support distance weighting through a `weights='distance'` parameter.

9 Computational Considerations and Limitations

K-NN is conceptually simple but becomes computationally expensive at prediction time, especially as datasets grow.

9.1 The Training–Prediction Trade-off

K-NN Complexity

Given m training examples in n dimensions:

Phase	Time Complexity	Space Complexity
Training	$O(1)$	$O(mn)$
Prediction	$O(mn)$	—

K-NN is a **lazy learner**: it stores all training data and defers computation until prediction time, when it must compute distances to every training point.

This is the opposite of **eager learners** like logistic regression, which invest time during training to build a model, then make fast predictions. K-NN's instant training becomes a liability when you need real-time predictions at scale.

Data structures like KD-Trees and Ball Trees can accelerate search, but they work best only in low dimensions ($n < 20$). For web-scale applications, approximate methods trade exactness for speed.

9.2 The Curse of Dimensionality

As dimensionality increases, a devastating phenomenon emerges: distances between points converge, and the notion of "nearest neighbor" loses meaning.

Distance Concentration in High Dimensions

Sample 1000 random points from the unit hypercube $[0, 1]^n$ and measure distances:

Dimensions	Min Distance	Max Distance	Ratio
2	0.008	1.392	174×
10	1.256	2.886	2.3×
100	8.012	11.523	1.4×

In 2 dimensions, the farthest point is 174 times farther than the nearest—clear proximity. In 100 dimensions, this ratio collapses to just 1.4×. When everything is equidistant, K-NN cannot distinguish signal from noise.

The data needed to maintain useful density grows exponentially as $O(2^n)$.
Practical rule: K-NN works well up to about 20 dimensions, becomes questionable beyond 50, and fails beyond 100.
Mitigation strategies: Remove irrelevant features, reduce dimensions using techniques like PCA, or switch to algorithms designed for high dimensions, such as tree-based methods.

10 When to Use K-NN: A Practical Guide

The K-NN Decision Guide

Use K-NN when:

- Dataset is small-to-medium ($m < 100{,}000$) with few dimensions ($n < 20$)
- Decision boundaries are complex and non-linear
- You need interpretable predictions ("these similar examples support this")
- You are building a quick baseline or prototype

Avoid K-NN when:

- Dimensionality is high ($n > 50$) without dimensionality reduction
- Dataset is large ($m > 1{,}000{,}000$) and requires fast predictions
- Memory is limited (K-NN must store all training data)
- Data is very noisy with many outliers

Successful application domains: Recommender systems, anomaly detection, image classification (with feature engineering), medical diagnosis.

Despite being introduced in 1951, K-NN remains valuable as a baseline method and for problems with irregular, complex boundaries. Modern applications often combine K-NN with

other techniques—for instance, using neural networks to first reduce dimensions, then applying K-NN in that simpler space.

11 Chapter Summary

Essential K-NN Concepts

The algorithm in three steps:

1. Compute distances from the query to all training points
2. Select the K nearest neighbors
3. Aggregate via voting (classification) or averaging (regression)

Critical implementation choices:

- **Distance metric:** Euclidean for most cases; Manhattan for outlier robustness
- **Feature scaling:** Always standardize—K-NN is extremely sensitive to scale
- **Hyperparameter K:** Cross-validate; start around $\sqrt{m}$
- **Weighting:** Use inverse-distance weights when neighbors vary in proximity

Key trade-offs:

- Small $K \rightarrow$ overfitting (high variance); Large $K \rightarrow$ underfitting (high bias)
- Zero training time $\rightarrow$ expensive predictions
- Maximum flexibility $\rightarrow$ only practical for low dimensions and modest data

The curse of dimensionality: Beyond 20–50 dimensions, distances become meaningless and K-NN fails. Always check your dimensionality before choosing K-NN.

K-NN teaches a fundamental lesson: sometimes the simplest approach—"find what's similar and copy it"—works remarkably well. But simplicity has costs. You now understand when those costs are worth paying and when to reach for more sophisticated tools. That judgment is what makes you a practitioner, not just a formula-follower.