Instance-Based Learning

What Is Instance-Based Learning?

- A **lazy learning** method: stores instances instead of building a model
- Predicts by comparing new cases to stored examples
- Most common method: **k-Nearest Neighbor (kNN)**
- Uses **distance functions** to measure similarity

Why Use Instance-Based Learning?

- Conceptually simple
- No training time — just store data!
- Handles complex, irregular decision boundaries
- Works naturally for multi-class problems

Core Idea

- Given a new instance $\mathbf{x}$:
  1. Compute distance to all stored instances
  2. Select nearest neighbor(s)
  3. Predict class by neighbor class labels

Distance Functions

- Euclidean (most common):

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

- Other metrics:
  - Manhattan distance
  - Minkowski distance
  - Hamming distance (symbolic features)

Normalization Is Essential

- Different scales can distort distances.
- Normalize numeric attributes to **[0,1]**:

$$a' = \frac{v - min}{max - min}$$

- Prevents large-scale attributes from dominating.

Handling Nominal Attributes

- If values match -> distance $= 0$
- If values differ -> distance $= 1$
- Missing nominal values -> treat as maximally different (distance $= 1$)

Handling Missing Numeric Values

- Both missing -> distance = 1
- One missing -> difference = max(value, 1 - value)
- Assumes missing means maximum uncertainty

k-Nearest Neighbor (kNN)

- Uses **k nearest neighbors** instead of just one.
  - Majority vote for classification
  - Typical values: $k = 3, 5, 7$
  - Reduces sensitivity to noise

Weighted kNN

- Closer neighbors get more influence:

$$\mathit{weight} = \frac{1}{\mathit{distance}}$$

- Improves predictions when distances vary widely.

Efficiency Challenges

- Naive kNN prediction -> $\mathcal{O}(n)$ per query
- Slow for large datasets.
- Solution: accelerate using spatial data structures.

kD-Trees Overview

- Binary tree that partitions data along axes.
    - Splits on attribute with greatest variance
    - Uses median value for balanced tree
    - Efficient in **low-dimensional** spaces

Building a kD-Tree

1. Choose attribute with largest variance
2. Split at median
3. Recursively partition subsets

- Produces well-shaped (non-skinny) regions.

kD-Tree Search

1. Start at root
2. Descend to best leaf
3. Record best candidate
4. Backtrack
5. Sibling region worth exploring?
   - Yes: search sibling
   - No: stop

When kD-Trees Break Down

- Become ineffective when:
  - Dimensionality is high
  - Data is skewed
  - Rectangular splits poorly model true neighborhoods

Ball Trees Overview

- Use **hyperspherical partitions** instead of rectangles.
  - Each node stores center + radius
  - Better for high-dimensional or skewed data

Ball Tree Search Steps

1. Descend into leaf containing target region
2. Record nearest candidate
3. Backtrack
4. Skip nodes whose balls lie outside search radius
5. Explore only necessary regions

Sensitivity to Noise

- Instance-based methods sensitive to:
  - Outliers
  - Duplicate conflicting points
  - Mislabeled instances
- Mitigation:
  - Use k $>$ 1
  - Prune noisy exemplars

Voting Feature Intervals

- Fast approximate method:
  - Convert numeric attributes to intervals
  - For each interval, store class counts
  - Classify by voting across intervals
- Useful for large datasets.

Strengths

- Simple and intuitive
- No training time
- Flexible decision shapes
- Works well with mixed data types

Weaknesses

- Slow classification for large datasets
- Must store entire dataset
- Sensitive to irrelevant features
- Struggles in high dimensions

Summary

- Instance-based learning stores examples and compares new ones directly
- kNN $\rightarrow$ majority or weighted voting
- Distance metrics and normalization crucial
- Data structures (kD-trees, ball trees) improve speed
- Best for low-dimensional, clean datasets