Supervised Learning: k-Nearest Neighbors

Distance Metrics, Curse of Dimensionality, and Choosing k

Sarwan Ali

Department of Computer Science
Georgia State University

Understanding k-Nearest Neighbors

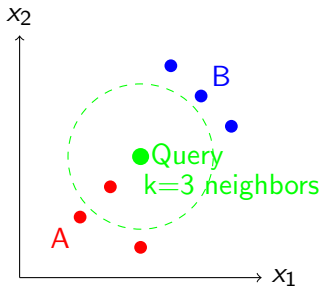
Today's Learning Journey

What is k-Nearest Neighbors (k-NN)?

k-Nearest Neighbors is a simple, intuitive supervised learning algorithm that:

- Makes predictions based on the k closest training examples
- Uses lazy learning - no explicit training phase
- Stores all training data and defers computation until prediction
- Works for both classification and regression

Key Idea: Similar inputs should produce similar outputs



k-NN Algorithm Steps

1. **Store** all training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$
2. For a new query point $\mathbf{x}_q$:
   1. **Calculate** distance from $\mathbf{x}_q$ to all training points
   2. **Find** the k closest neighbors
   3. **Predict** based on these k neighbors:
      - Classification: Majority vote
      - Regression: Average of neighbor values

Key Parameters

- **k**: Number of neighbors to consider
- **Distance metric**: How to measure similarity

Distance Metrics: The Foundation of k-NN

Distance metrics determine how we measure similarity between data points.

For two points $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_d)$:

1. Euclidean Distance (L2)

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

2. Manhattan Distance (L1)

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

3. Minkowski Distance

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

4. Cosine Distance

$$d(\mathbf{x}, \mathbf{y}) = 1 - \frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}||\mathbf{y}|}$$

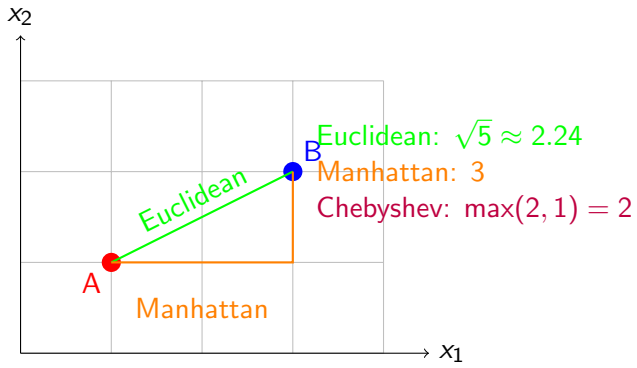
5. Hamming Distance

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

Special cases:

- $p = 1$: Manhattan
- $p = 2$: Euclidean
- $p = \infty$: Chebyshev

Visual Comparison of Distance Metrics



When to use which?

- **Euclidean**: Continuous features, geometric interpretation
- **Manhattan**: Sparse data, when features have different scales
- **Cosine**: High-dimensional data, text analysis
- **Hamming**: Categorical/binary features

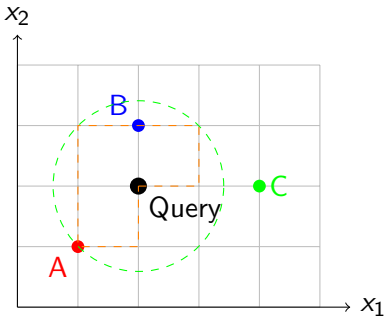
Impact of Distance Metrics on k-NN

Example Dataset:

Point	x_1	x_2
A	1	1
B	2	3
C	4	2
Query	2	2

Distances to Query (2,2):

- Euclidean: A(1.41), B(1.41), C(2.24)
- Manhattan: A(2), B(1), C(2)
- Chebyshev: A(1), B(1), C(2)



Different metrics give different nearest neighbors!

The Curse of Dimensionality

Definition: As the number of dimensions increases, the performance of k-NN degrades significantly.

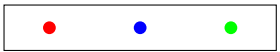
Why does this happen?

1. **Distance becomes meaningless**
   - All points become equidistant
   - Nearest and farthest neighbors have similar distances
2. **Sparsity of data**
   - Volume of high-dimensional space grows exponentially
   - Data becomes sparse, neighborhoods become empty
3. **Computational complexity**
   - Distance computation becomes expensive
   - Memory requirements increase

Low Dimensions

Clear neighborhoods

High Dimensions



All points equidistant

Mathematical Insight: Distance Concentration

Distance Concentration Phenomenon:

In high dimensions, the ratio of distances approaches 1: $\lim_{d\to\infty} \frac{D_{\text{min}}}{D_{\text{max}}} \to 1$

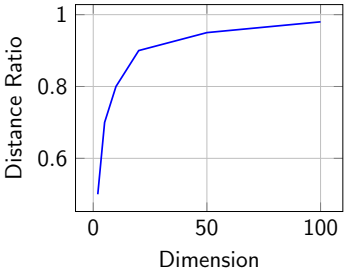
where D_{min} and D_{max} are the minimum and maximum distances to a query point.

Demonstration:

Consider uniformly distributed points in a unit hypercube $[0, 1]^d$.

As d increases:

- Expected distance $\approx \frac{\sqrt{d}}{6}$
- Variance decreases relative to mean
- All distances become similar



Implication

In high dimensions, the concept of "nearest" neighbor loses meaning!

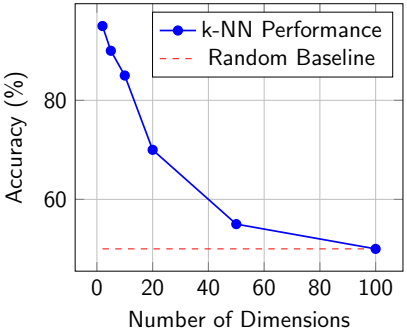
Effects of Curse of Dimensionality on k-NN

Performance Degradation:

- Accuracy drops as dimensions increase
- Overfitting becomes more likely
- Comput. cost increases exponentially
- Memory usage becomes prohibitive

Empirical Evidence:

- Optimal performance typically at $d < 10$
- Significant degradation at $d > 20$
- Nearly random performance at $d > 100$



Solutions

- **Dimensionality Reduction:** PCA, t-SNE, feature selection
- **Feature Engineering:** Create meaningful low-dimensional features
- **Distance Metric Learning:** Learn appropriate distance functions
- **Locality Sensitive Hashing:** Approximate nearest neighbors

Choosing k: The Bias-Variance Tradeoff

The choice of k significantly affects k-NN performance!

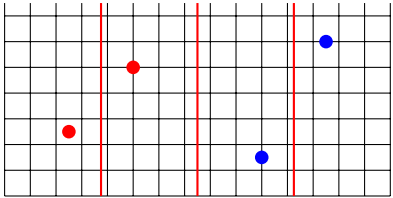
Small k (e.g., k=1):

- Low bias, high variance
- Sensitive to noise and outliers
- Complex decision boundaries
- Prone to overfitting

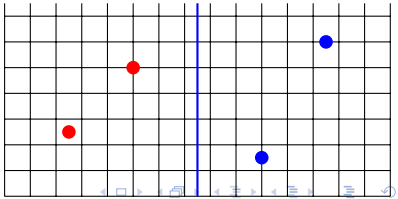
Large k:

- High bias, low variance
- Smooth decision boundaries
- Less sensitive to noise
- May underfit the data

k=1 (Overfitting)



k=large (Underfitting)



Methods for Choosing k

1. Cross-Validation

- Try different values of k (typically odd numbers)
- Use k-fold cross-validation to estimate performance
- Choose k that minimizes validation error

2. Rule of Thumb

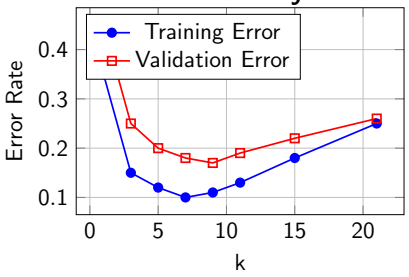
- $k = \sqrt{n}$ where n is the number of training samples
- Choose odd k to avoid ties in classification
- Start with $k \in \{1, 3, 5, 7, 9, \ldots\}$

3. Error Analysis

- Plot training and validation error vs. k
- Look for the "sweet spot" where both errors are minimized
- Consider computational constraints

k Selection: Practical Example

Validation Curve Analysis:



Optimal k = 9 (lowest validation error)

Guidelines:

1. **Start simple:** Try k = 1, 3, 5
2. **Use cross-validation:** 5-fold or 10-fold
3. **Consider class balance:**
   - For imbalanced data, use smaller k
   - Weight neighbors by distance
4. **Computational budget:** Larger k means more computation
5. **Domain knowledge:** Some applications prefer certain k values

Pro Tip

Always use odd k for binary classification to avoid ties!

k-NN: Advantages and Disadvantages

Advantages:

- ✔ Simple and intuitive
- ✔ No assumptions about data distribution
- ✔ Works with classification and regression
- ✔ Can capture complex patterns
- ✔ Naturally handles multi-class problems
- ✔ Can be used for anomaly detection

Disadvantages:

- ✘ Computationally expensive at prediction time
- ✘ Sensitive to irrelevant features
- ✘ Suffers from curse of dimensionality
- ✘ Sensitive to scale of features
- ✘ Memory intensive (stores all data)
- ✘ Poor performance with imbalanced data

When to Use k-NN

- Small to medium-sized datasets
- Low-dimensional feature spaces (d ¡ 20)
- Irregular decision boundaries
- When interpretability is important
- As a baseline method for comparison

Improving k-NN Performance

Preprocessing Techniques:

1. **Feature Scaling:**
   - Standardization: $z = \frac{x-\mu}{\sigma}$
   - Min-Max normalization: $x' = \frac{x-x_{\text{min}}}{x_{\text{max}}-x_{\text{min}}}$
2. **Feature Selection:**
   - Remove irrelevant features
   - Use correlation analysis, mutual information
3. **Dimensionality Reduction:**
   - PCA, LDA, t-SNE
   - Preserve most important information

Algorithmic Improvements:

- **Weighted k-NN:** Give closer neighbors more influence
- **Approximate methods:** LSH, random sampling
- **Efficient data structures:** KD-trees, Ball trees
- **Distance metric learning:** Learn optimal distance function

Key Takeaways

1. **k-NN Fundamentals:**
   - Simple, lazy learning algorithm
   - Makes predictions based on k closest neighbors
   - Works for both classification and regression
2. **Distance Metrics Matter:**
   - Choice affects which points are considered neighbors
   - Euclidean for continuous, Manhattan for sparse data
   - Consider data characteristics when choosing
3. **Curse of Dimensionality:**
   - Performance degrades significantly in high dimensions
   - Distance becomes meaningless, data becomes sparse
   - Use dimensionality reduction techniques
4. **Choosing k:**
   - Bias-variance tradeoff: small k (high variance), large k (high bias)
   - Use cross-validation to find optimal k
   - Consider odd values to avoid ties

Next Steps

What's Coming Next:

- Support Vector Machines (SVMs)
- Ensemble Methods (Random Forest, Boosting)
- Neural Networks and Deep Learning
- Unsupervised Learning Methods

Practice Exercises:

- Implement k-NN from scratch
- Experiment with different distance metrics
- Apply dimensionality reduction techniques
- Compare performance across different values of k
- Handle imbalanced datasets with weighted k-NN

Questions?

Remember: The best algorithm is the one that works best for *your* specific problem!

Bonus: Mathematical Formulation

k-NN Decision Rule:
For classification with query point $\mathbf{x}_q$:

$$\hat{y} = \arg\max_c \sum_{i \in N_k(\mathbf{x}_q)} \mathbf{1}_{y_i = c}$$

For regression:

$$\hat{y} = \frac{1}{k} \sum_{i \in N_k(\mathbf{x}_q)} y_i$$

Weighted k-NN:

$$\hat{y} = \frac{\sum_{i \in N_k(\mathbf{x}_q)} w_i y_i}{\sum_{i \in N_k(\mathbf{x}_q)} w_i}$$

where $w_i = \frac{1}{d(\mathbf{x}_q, \mathbf{x}_i) + \epsilon}$ and $N_k(\mathbf{x}_q)$ denotes the k nearest neighbors of $\mathbf{x}_q$.

Note

The small constant ϵ prevents division by zero when query point equals a training point.