

K-Nearest Neighbors (KNN) Algorithm: A Detailed Theoretical Tutorial

@S S Roy





What is KNN?

K-Nearest Neighbors (KNN) is a **non-parametric, instance-based (lazy) learning algorithm** used for **classification and regression** tasks in Machine Learning.

- **"Non-parametric"** means it doesn't make any assumptions about the underlying data distribution.
- **"Instance-based"** means it stores the entire training dataset and performs computation only during the prediction phase.

*The **K-Nearest Neighbors (KNN)** algorithm was first introduced by **Evelyn Fix and Joseph Hodges in 1951**. Their work, conducted for the US military, focused on discriminant analysis as a non-parametric classification method.*

Historical Background

-  **Inventor:** Thomas M. Cover and Peter E. Hart
 -  **Year:** 1967 (LATER ON)
 -  **Paper:** "Nearest Neighbor Pattern Classification", published in *IEEE Transactions on Information Theory*
 -  **Significance:**
 - This foundational paper introduced the **nearest neighbor rule** and provided **theoretical guarantees** on its performance.
 - It showed that the 1-NN classifier's error is never worse than **twice** the Bayes optimal error rate, making it **statistically consistent**.
 - KNN laid the groundwork for modern **pattern recognition, decision boundaries, and non-parametric methods**.
-

Core Intuition

KNN predicts the class (or value) of a new data point by **looking at the “K” closest points** in the training set and **taking a majority vote (for classification)** or **averaging the values (for regression)**.

The KNN Classification Algorithm (Step-by-Step)

1. Choose the number of neighbors, K.
 2. Compute the distance between the new point and all training samples using a chosen distance metric.
 3. Sort the distances in ascending order.
 4. Select the K-nearest neighbors.
 5. Classify the new point by the **majority class** among the neighbors.
-

Distance Metrics Used in KNN

1. Euclidean Distance

- Formula:

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

- **Used When:** Features are continuous and the input space is isotropic (no dominant axis).
- **Popular In:** Image recognition, general-purpose KNN.

2. Manhattan Distance (a.k.a. L1 Norm or Taxicab Distance)

- Formula:

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

- **Used When:** Data lies in a **grid-like topology** (e.g., city blocks, discrete units), or when features have **unequal variances**.
- **Popular In:** Natural language processing, time-series classification.

✓ 3. Minkowski Distance (Generalized Distance)

- **Formula:**

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

- $p = 1$: Manhattan Distance
- $p = 2$: Euclidean Distance
- Can be tuned for custom distance behavior.

✓ 4. Cosine Similarity

- Measures **angle** between vectors rather than magnitude.
- **Used When:** The direction of the data points matters more than their magnitude (e.g., text data, document similarity).

✓ 5. Hamming Distance

- Used for **categorical** or **binary features**.
- Counts the number of differing attributes.

✓ 6. Chebyshev Distance

- **Formula:**


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

- Sensitive to the **largest** difference in any feature.

🤝 Are Both Euclidean and Manhattan Used in Practice?

Yes — absolutely. Both are widely used, depending on the data characteristics:

Feature Type	Suitable Distance
Continuous	Euclidean, Manhattan
Grid-like Data	Manhattan
High-Dimensional	Manhattan often preferred (less affected by distance concentration)
Normalized Data	Euclidean (after scaling)

 **Important Note:** Always **scale your features** (e.g., Min-Max or Z-score normalization) before using Euclidean or Manhattan distances, especially if features are on different scales.

Theoretical Guarantees and Properties

Property	Description
Bayes Optimal Bound	1-NN classification error $\leq 2 \times$ Bayes error rate
Consistency	As data size $\rightarrow \infty$, KNN prediction \rightarrow Bayes optimal prediction
Curse of Dimensionality	In high dimensions, distance becomes less meaningful (affects all metrics)

Example (With Euclidean Distance)

Dataset:

Brightness	Saturation	Class
40	20	Red
50	50	Blue
60	90	Blue
10	25	Red
70	70	Blue
60	10	Red
25	80	Blue

New Entry: (20, 35)

Using Euclidean formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

(You can refer to the earlier part for full computations.)

Top 5 Nearest Neighbors (sorted):

- (10, 25) → Red
- (40, 20) → Red
- (50, 50) → Blue
- (25, 80) → Blue
- (60, 10) → Red

Result: Majority class = Red

👉 New entry classified as **Red**

How to Choose the Value of K

Strategy	Reason
Odd K (3, 5, 7)	Avoids ties in binary classification
Cross-validation	Empirical tuning for best K
Avoid K = 1	Sensitive to noise and outliers
Rule of Thumb	$K \approx \sqrt{n}$, where n = size of training set

✅ Advantages of KNN

- Simple and easy to implement
 - No model training needed
 - Naturally supports **multi-class classification**
 - Versatile with different distance metrics
-

❌ Disadvantages of KNN

- **High prediction time** for large datasets
 - **Memory intensive:** stores entire training data
 - **Sensitive to irrelevant features**
 - Requires **scaling** and careful **K selection**
 - Struggles with high-dimensional data (curse of dimensionality)
-

🧰 Best Practices for Using KNN

1. **Feature Scaling** is a must (Z-score or Min-Max)
 2. Use **dimensionality reduction** (e.g., **PCA**) for high-dimensional data
 3. Apply **feature selection** to reduce noise
 4. Use **KD-Trees** or **Ball Trees** for speed optimization
 5. Consider using **weighted KNN** (closer neighbors have more weight)
-



Summary

Aspect	Description
Invented by	Thomas Cover & Peter Hart (1967)
Type	Non-parametric, Instance-based
Common Use Cases	Classification, regression, recommendation, anomaly detection
Popular Distances	Euclidean, Manhattan, Minkowski, Cosine, Hamming
Key Limitation	Scalability & curse of dimensionality
Real-World Examples	Handwriting recognition, recommendation engines, customer segmentation

Here's a **complete and detailed KNN example** with **all distance calculations shown step-by-step**, using the dataset from the earlier discussion and a new data point. We'll use the **Euclidean Distance** and classify based on **K = 5 nearest neighbors**.

KNN Classification Example – Full Distance Calculations

Training Dataset

Index	Brightness (X)	Saturation (Y)	Class
1	40	20	Red
2	50	50	Blue
3	60	90	Blue
4	10	25	Red
5	70	70	Blue
6	60	10	Red
7	25	80	Blue

New Data Point

Brightness	Saturation	Class
20	35	?

We need to classify this point.



Step 1: Use Euclidean Distance Formula

$$\text{Distance} = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

Where:

- $X_2 = 20$ and $Y_2 = 35$ (new point)
 - (X_1, Y_1) is each point from the training dataset
-



Step 2: Calculate Distances

◆ Distance to Point 1 (40, 20) – Red

$$d_1 = \sqrt{(20 - 40)^2 + (35 - 20)^2} = \sqrt{(-20)^2 + (15)^2} = \sqrt{400 + 225} = \sqrt{625} = \boxed{25}$$

◆ Distance to Point 2 (50, 50) – Blue

$$d_2 = \sqrt{(20 - 50)^2 + (35 - 50)^2} = \sqrt{(-30)^2 + (-15)^2} = \sqrt{900 + 225} = \sqrt{1125} \approx \boxed{33.54}$$

◆ Distance to Point 3 (60, 90) – Blue

$$d_3 = \sqrt{(20 - 60)^2 + (35 - 90)^2} = \sqrt{(-40)^2 + (-55)^2} = \sqrt{1600 + 3025} = \sqrt{4625} \approx \boxed{68.0}$$

◆ Distance to Point 4 (10, 25) – Red

$$d_4 = \sqrt{(20 - 10)^2 + (35 - 25)^2} = \sqrt{(10)^2 + (10)^2} = \sqrt{100 + 100} = \sqrt{200} \approx \boxed{14.14}$$

◆ Distance to Point 5 (70, 70) – Blue

$$d_5 = \sqrt{(20 - 70)^2 + (35 - 70)^2} = \sqrt{(-50)^2 + (-35)^2} = \sqrt{2500 + 1225} = \sqrt{3725} \approx \boxed{61.03}$$

◆ Distance to Point 6 (60, 10) – Red

$$d_6 = \sqrt{(20 - 60)^2 + (35 - 10)^2} = \sqrt{(-40)^2 + (25)^2} = \sqrt{1600 + 625} = \sqrt{2225} \approx \boxed{47.17}$$

◆ Distance to Point 7 (25, 80) – Blue

$$d_7 = \sqrt{(20 - 25)^2 + (35 - 80)^2} = \sqrt{(-5)^2 + (-45)^2} = \sqrt{25 + 2025} = \sqrt{2050} \approx \boxed{45.28}$$



Step 3: Tabulate Distances

Index	Point (X, Y)	Class	Distance
1	(40, 20)	Red	25
2	(50, 50)	Blue	33.54
3	(60, 90)	Blue	68.01
4	(10, 25)	Red	14.14
5	(70, 70)	Blue	61.03
6	(60, 10)	Red	47.17
7	(25, 80)	Blue	45.28



Step 4: Sort by Distance

Rank	Point (X, Y)	Class	Distance
1	(10, 25)	Red	14.14

Rank	Point (X, Y)	Class	Distance
2	(40, 20)	Red	25
3	(50, 50)	Blue	33.54
4	(25, 80)	Blue	45.28
5	(60, 10)	Red	47.17
6	(70, 70)	Blue	61.03
7	(60, 90)	Blue	68.01



Step 5: Pick Top K = 5 Nearest Neighbors

Neighbor	Class
1	Red
2	Red
3	Blue
4	Blue
5	Red



Final Step: Majority Voting

- Red: 3 neighbors
- Blue: 2 neighbors



Predicted Class = Red

✓ Updated Dataset (After Classification)

Brightness	Saturation	Class
40	20	Red
50	50	Blue
60	90	Blue
10	25	Red
70	70	Blue
60	10	Red
25	80	Blue
20	35	Red (Predicted)