UNIT-II

UNIT-II: Nearest Neighbor-Based Models: Introduction to Proximity Measures, Distance Measures, Non-Metric Similarity Functions, Proximity Between Binary Patterns, Different Classification Algorithms Based on the Distance Measures, K-Nearest Neighbor Classifier, Radius Distance Nearest Neighbor Algorithm, KNN Regression, Performance of Classifiers, Performance of Regression Algorithms.

INTRODUCTION:

- Nearest neighbor-based models, especially the **k-nearest neighbors (k-NN)** algorithm, are a fundamental class of ML techniques that rely on the idea of **proximity or similarity between data points for making predictions**.

- These models are used in both **classification** and **regression** tasks and are based on the idea that similar points are likely to have similar labels or values.

- These are **non-parametric** (don't have a fixed number of parameters) machine learning techniques that classify data points or predict values based on the **similarity (or distance)** to nearby data points.

  - Nearest Neighbor methods rely on the idea that similar instances have similar outputs.
  - Predictions are based on the proximity of a query point to data points in the training set.

Applications of Nearest Neighbor Models

Pattern Recognition: Handwriting and facial recognition.

Recommender Systems: Collaborative filtering.

Anomaly Detection: Identifying outliers.

Medical Diagnosis: Classifying symptoms.

Introduction to Proximity Measures

• Proximity measures are used to **quantify the degree of similarity or dissimilarity** between two or more pattern vectors.

• These pattern vectors can represent documents, images, or even entire audio or video files.

• These are often used by ML algorithms to **compare and classify or group or make predictions** using these patterns.

• These are **mathematical techniques** used to calculate the similarity or dissimilarity between data points, which can facilitate clustering, classification, and other analytical tasks.

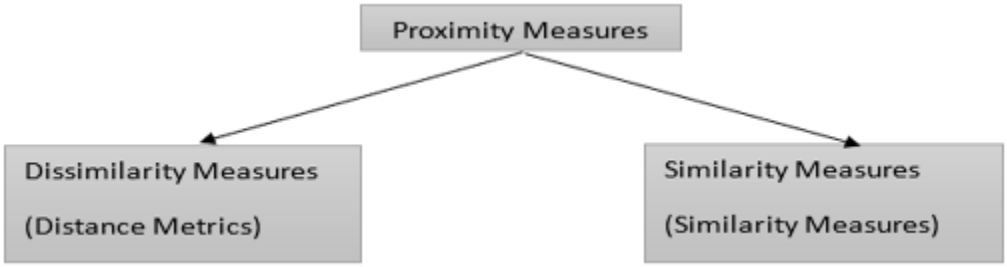
• A **'Proximity Measure'** is a function that calculates the similarity or dissimilarity between vectors or subsets of a dataset based on certain criteria such as **distance or similarity levels**.

• These measures can be applied to vectors or subsets of datasets to derive a **similarity score** that informs the relationships among the data points.

• The idea is that if two data points are close to each other in the feature space, they are more likely to belong to the same class (in classification) or have similar values (in regression).

• These proximity measures are essential for nearest neighbor algorithms.

Types of Proximity Measures:



Distance Measures/Dissimilarity Measures:

- A dissimilarity measure is a mathematical function that quantifies the degree of dissimilarity between two objects or data points. It is a numerical score measuring how different two data points are.
- It takes two data points as input and produces a dissimilarity score as output, ranging from 0 (identical or perfectly similar) to 1 (completely dissimilar). A few dissimilarity measures also have infinity as their upper limit.
- A dissimilarity measure can be obtained by using different techniques such as Euclidean distance, Manhattan distance, and Hamming distance.
- Dissimilarity measures are often used in identifying outliers, anomalies, or clusters.
- Smaller distance = higher similarity, larger distance = higher dissimilarity.

the most commonly used distance measures in machine learning are as follows:

1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance
4. Hamming Distance
5. ChebyShev Distance

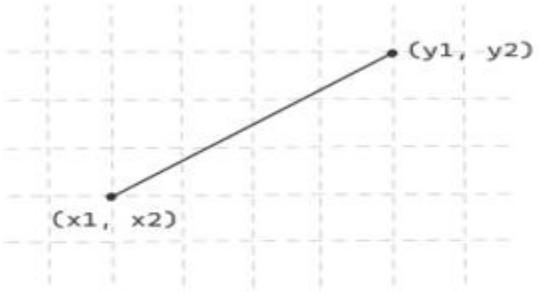
1. Euclidean Distance:

Euclidean distance is considered the traditional metric for problems with geometry. It can be simply explained as the ordinary distance between two points. It is one of the most used algorithms in the cluster analysis.

Formula:

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

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



Euclidean Distance in 2D Plane

- **Best for**: Continuous numerical data (when features are normalized).
- **Example**: Distance between two cities on a 2D map.

Example:

x = (2,3), y = (5,7)

$$d = \sqrt{(2-5)^2 + (3-7)^2} = \sqrt{9+16} = 5$$

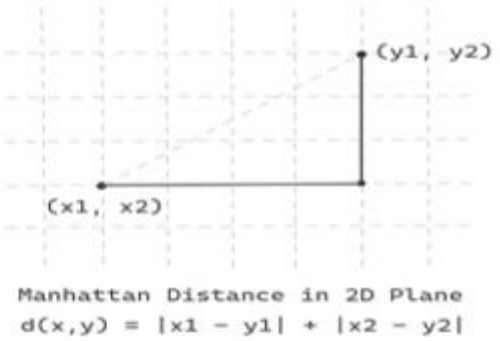
2. Manhattan Distance:

The Manhattan distance, also called **taxicab** distance or **city block** distance, is another popular distance metric. Suppose you're inside a two-dimensional plane and you can move only along the axes as shown:

Formula:

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

$$d(\mathbf{x}, \mathbf{y}) = |x_1 - y_1| + |x_2 - y_2|$$



- **Best for**: High-dimensional data or when diagonal movement has no meaning.
- **Example**: Distance between blocks in a city (taxicab geometry).

Example:

x = (2,3), y = (5,7)

$$d = |2 - 5| + |3 - 7| = 3 + 4 = 7$$

3. Minkowski Distance:

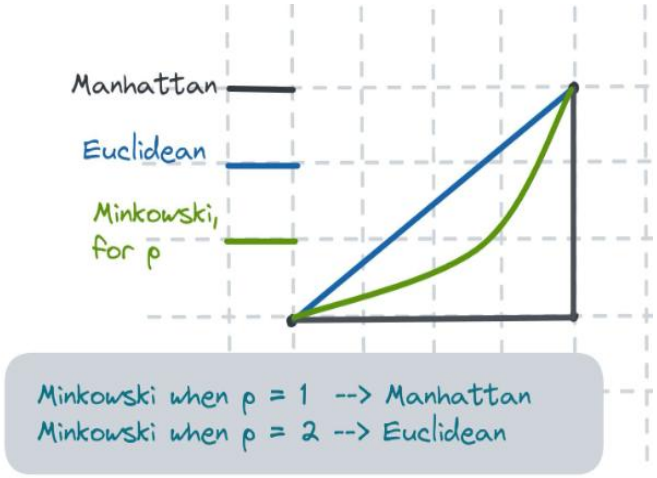
Minkowski distance is a generalized distance measure that includes both Euclidean and Manhattan distances as special cases, controlled by a parameter p.

Formula:

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

p=1 → Manhattan

p=2 → Euclidean



- **Best for**: Flexible distance calculations where p is tuned.
- **Example**: For p=1, it becomes Manhattan; for p=2, it becomes Euclidean.

It is pretty straightforward to see that for **p = 1**, the Minkowski distance equation takes the same form as that of Manhattan distance:

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

Similarly, for **p = 2**, the Minkowski distance is equivalent to the Euclidean distance:

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^2 \right)^{1/2}$$

Manhattan Distance

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

Euclidean Distance

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^2 \right)^{\frac{1}{2}} \quad (p=2)$$

Chebyshev Distance

$$\lim_{P \to \infty} \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

◆ **Example**

Let

$$x = (2, 4), \quad y = (6, 8)$$

Case 1 : p = 1 → Manhattan Distance

$$d = (|2 - 6| + |4 - 8|) = 4 + 4 = \boxed{8}$$

Case 2 : p = 2 → Euclidean Distance

$$d = \sqrt{(2-6)^2 + (4-8)^2} = \sqrt{16 + 16} = \sqrt{32} = \boxed{5.66}$$

Case 3 : p = 3 → Minkowski with p=3

$$d = \sqrt[3]{|2-6|^3 + |4-8|^3} = \sqrt[3]{64 + 64} = \sqrt[3]{128} = \boxed{5.04}$$

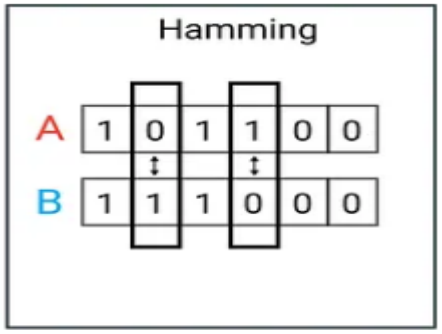
4. Hamming Distance:

The number of positions where two strings (of equal length) differ. Commonly used for error detection and sequence comparison.

Formula:

$$d(x,y) = \sum_{i=1}^{n} [x_i \neq y_i]$$

where [$[x_i \neq y_i] = 1$ if symbols differ, else 0.



- **Best for**: Binary strings, DNA sequences, error correction.
- **Example**: Hamming distance between “karolin” and “kathrin” = 3.

Example:

x = 101110

y = 100100

Differences = 2 → **Hamming Distance = 2**

5. **ChebyShev Distance:**

Chebyshev Distance is a metric used to calculate the distance between two points in a space where movement can occur in any direction including diagonally. It is defined as the maximum absolute difference between the corresponding coordinates of two points.

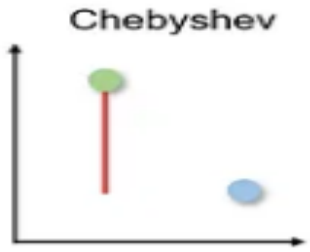
Formula:

$$D_{\text{Chebyshev}}(A,B) = \max_{i=1}^{n} |x_i - y_i|$$

Example:

x=(2,3), y=(5,7)

→ max(3,4)=4



Similarity Measures/Similarity Metrics:

- A similarity measure is a mathematical function that quantifies the degree of similarity between two objects or data points. It is a numerical score measuring how alike two data points are.
- It takes two data points as input and produces a similarity score as output, typically ranging from 0 (completely dissimilar) to 1 (identical or perfectly similar).
- A similarity measure can be based on various mathematical techniques such as Cosine similarity, Jaccard similarity, and Pearson correlation coefficient.

- Similarity measures are generally used to identify duplicate records, equivalent instances, or identifying clusters.

the most commonly used distance measures in machine learning are as follows:

1. Cosine Similarity
2. Jaccard Similarity
3. Hamming Similarity

1. Cosine similarity:

Cosine similarity is a metric that measures the similarity between two non-zero vectors of numbers by calculating the cosine of the angle between them. It is used to determine if two vectors are pointing in the same general direction, irrespective of their magnitude (length).

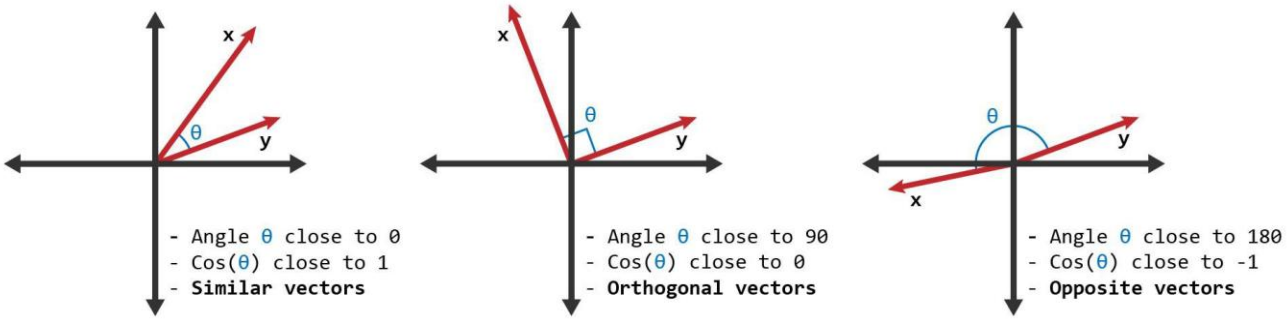
The result is a value between **-1 and 1**:

- A value of **1** means the vectors point in the exact same direction (perfectly similar).
- A value of **0** means the vectors are orthogonal (at a 90-degree angle), indicating no relationship.
- A value of **-1** means the vectors point in exactly opposite directions (perfectly dissimilar).

Formula

The mathematical formula for the cosine similarity between two vectors, **A** and **B**, is:

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \times ||\mathbf{B}||}$$



Let's calculate the cosine similarity between two simple 3-dimensional vectors:

- **Vector A:** [1,2,3]
- **Vector B:** [2,3,4]

Here is the step-by-step calculation:

1. **Calculate the dot product (A · B):**

$$(1 \times 2) + (2 \times 3) + (3 \times 4) = 2 + 6 + 12 = 20 \text{ [0.1.8]}$$

2. **Calculate the magnitude of Vector A (||A||):**

$$\sqrt{1^2 + 2^2 + 3^2} = \sqrt{1 + 4 + 9} = \sqrt{14} \approx 3.74 \text{ [0.1.17]}$$

3. **Calculate the magnitude of Vector B (||B||):**

$$\sqrt{2^2 + 3^2 + 4^2} = \sqrt{4 + 9 + 16} = \sqrt{29} \approx 5.39 \text{ [0.1.8]}$$

4. **Compute the cosine similarity:**

$$\text{Cosine Similarity} = \frac{20}{\sqrt{14} \times \sqrt{29}} \approx \frac{20}{3.74 \times 5.39} \approx \frac{20}{20.15} \approx 0.992$$

The resulting cosine similarity of approximately **0.992** indicates that the two vectors are highly similar and point in almost the exact same direction.

2. Jaccard Similarity:

Jaccard similarity, also known as the Jaccard index or Jaccard coefficient, is a measure of similarity between two sets. It is defined as the ratio of the intersection of the sets to their union

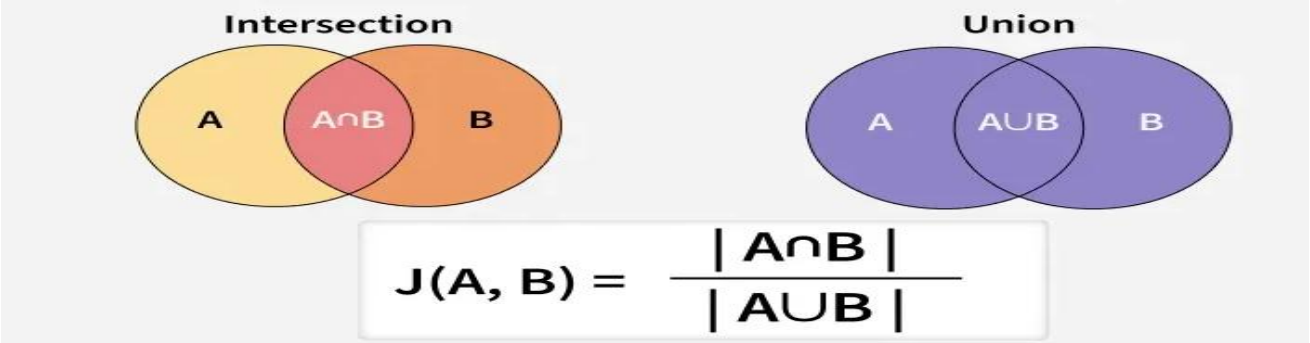
Formula:

For two sets A and B:

$$\text{Jaccard Similarity}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Where:

- $|A \cap B|$ = number of common elements
- $|A \cup B|$ = total unique elements in both sets



Example:

A = {1, 2, 3, 4} B = {3, 4, 5, 6}

- **Step 1: Find Intersection**

$$A \cap B = \{3, 4\}$$

So, $|A \cap B| = 2$

- **Step 2: Find Union**

$$A \cup B = \{1, 2, 3, 4, 5, 6\}$$

So, $|A \cup B| = 6$

- **Step 3: Apply Formula**

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{2}{6} = \boxed{0.33}$$

Jaccard Similarity = **0.33** → A and B share some elements, but are not very similar.

3. **Hamming Similarity:**

Hamming Similarity measures how **similar two strings or binary vectors** are by counting the number of **matching positions** and dividing by the total length.

$$\text{Hamming Similarity} = \frac{\text{Number of matching positions}}{\text{Total number of positions}}$$

It is related to Hamming Distance:

$$\text{Hamming Similarity} = 1 - \frac{\text{Hamming Distance}}{n}$$

Where n = length of the strings/vectors.

Example:

Given Binary Vectors

x = **1 0 1 1 0** y = **1 1 1 0 0**

Step 1: Compare each position

```
Position: 1  2  3  4  5
x       : 1  0  1  1  0
y       : 1  1  1  0  0
Match?  : ✓  X  ✓  X  ✓
```

Step 2: Count Matches

Matching positions = **3**

Total positions = **5**

Step 3: Hamming Similarity

$$\text{Hamming Similarity} = \frac{3}{5} = \boxed{0.6}$$

Hamming similarity = 0.6 → the two binary strings are 60% similar.

Distance Measures:

A **distance measure** computes how *dissimilar* two data points are.

- ✓ Smaller distance → more similar
- ✓ Larger distance → more different

The Distance Measures are:

1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance
4. Hamming Distance
5. ChebyShev Distance

Already explained in previous concept

Non-Metric Similarity Functions:

A **non-metric similarity function** measures how similar two objects are but **does NOT satisfy the properties of a metric distance**.

A metric distance must satisfy:

1. Non-negativity: $d(x,y) \geq 0$
2. Identity: $d(x,y)=0 \Leftrightarrow x=y$

3. Symmetry: d(x,y)=d(y,x)
4. Triangle inequality

Non-metric similarities violate one or more of these properties. They give **similarity scores**, not true distances.

Common Non-Metric Similarity Functions

1. **Cosine Similarity**

$$\text{sim}(x, y) = \frac{x \cdot y}{\|x\| \|y\|}$$

- ✓ Based on **angle**, not distance
- ✓ Not a true metric (violates triangle inequality)

2. **Jaccard Similarity**

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- ✓ Measures **overlap**
- ✓ Not metric in similarity form

3. **Hamming Similarity**

$$\text{Hamming Similarity}(x, y) = \frac{\text{Number of matching positions}}{n} = 1 - \frac{d_H(x, y)}{n}$$

- ✓ Measures position-wise similarity
- ✓ Value range:
  - **1** → exactly identical
  - **0** → completely different

4. **Dice (Sørensen) Similarity**

$$Dice(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

- ✓ Emphasizes common elements

5. **Pearson Correlation Similarity**

$$r(x, y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

- ✓ Measures **linear relationship**

6. **Simple Matching Coefficient (SMC)**

$$SMC = \frac{M_{11} + M_{00}}{M_{11} + M_{10} + M_{01} + M_{00}}$$

✓ Used for binary attributes

7. **Edit Similarity (from Edit Distance)**

$$Sim = 1 - \frac{\text{EditDistance}}{\max(|x|, |y|)}$$

✓ Used for strings

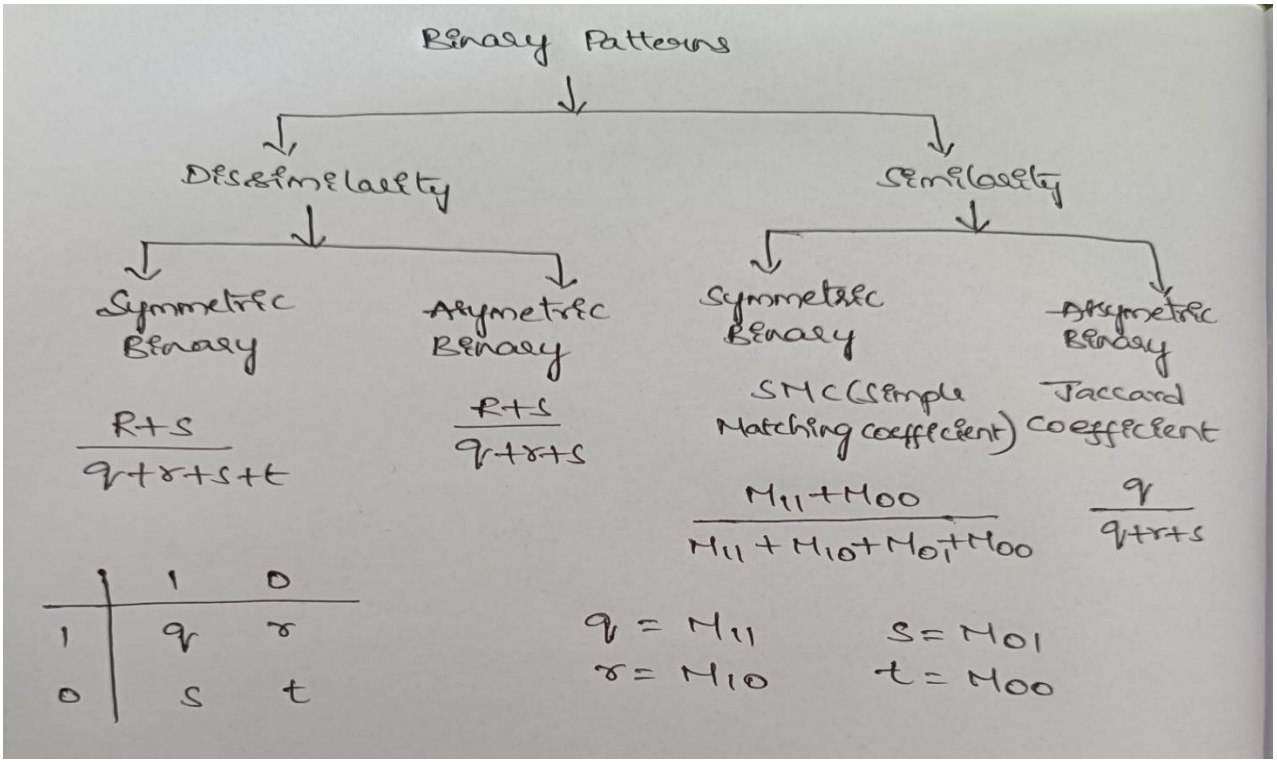
Proximity Between Binary Patterns:

A binary pattern is a pattern (data object) in which all attributes are binary-valued, i.e., each attribute can take only two states such as 0 or 1.

✓ 1 = presence / true / yes
✓ 0 = absence / false / no

So, a binary pattern is simply a vector made of 0s and 1s.

Classification of Binary Patterns:



Dissimilarity-Symmetric Binary Variables

A binary variable is symmetric if both of its states are equally valuable and carry the same weight. There is no preference for either state.

Dissimilarity that is based on symmetric binary variables is called symmetric binary dissimilarity.

Symmetric Binary

$$\text{Dissimilarity} = \frac{r + s}{q + r + s + t}$$

Dissimilarity-Asymmetric Binary Variables

A binary variable is asymmetric if the outcomes of the states are not equally important. Example: Medical data.

The dissimilarity based on such variables is called asymmetric binary dissimilarity, where the number of negative matches, t, is considered unimportant and thus is ignored in the computation.

Asymmetric Binary

$$\text{Dissimilarity} = \frac{r + s}{q + r + s}$$

Similarity-Symmetric Binary Variables

A symmetric binary similarity measure is used when both states of a binary variable (0 and 1) are equally important. It measures how similar two objects are by considering both the number of matching 1s and matching 0s.

When **0 and 1 are equally valuable**, we count:

- Matches of (1,1)
- Matches of (0,0)

to measure similarity.

1. Symmetric Binary → SCM (Simple Matching Coefficient)

$$\text{SCM} = \frac{m_{11} + m_{00}}{m_{11} + m_{10} + m_{01} + m_{00}}$$

Similarity-Asymmetric Binary Variables

Similarity for asymmetric binary variables is a measure used when only the presence (1) of an attribute is important and the absence (0) is not. It evaluates how similar two objects are based only on the number of attributes where both have value 1.

We **ignore 0–0 matches** and only count:
✓ Matches of (1,1)

2. Asymmetric Binary → Jaccard Coefficient

$$\text{Jaccard} = \frac{q}{q + r + s}$$

Example:

Let X= [1,0,1,1,0]

Y= [1,1,0,1,0]

Total attributes:

$n=m_{11}+m_{10}+m_{01}+m_{00}=2+1+1+1=5$

Dissimilarity Measures:

1. **Symmetric Binary Dissimilarity**

$$D_{sym} = \frac{m_{10} + m_{01}}{n}$$

$$D_{sym} = \frac{1 + 1}{5} = 0.4$$

2. **Asymmetric Binary Dissimilarity**

$$D_{asym} = \frac{m_{10} + m_{01}}{m_{11} + m_{10} + m_{01}}$$

$$D_{asym} = \frac{1 + 1}{2 + 1 + 1} = \frac{2}{4} = 0.5$$

Similarity Measures:

1. **Symmetric Binary Similarity – Simple Matching Coefficient (SMC)**
   ✓ Both 1–1 and 0–0 matches matter.

$$\text{SMC} = \frac{m_{11} + m_{00}}{n}$$

$$SMC = \frac{2+1}{5} = 0.6$$

2. **Asymmetric Binary Similarity – Jaccard Coefficient**
   ✓ Only 1–1 matches matter.
   ✓ 0–0 matches are ignored.

$$J = \frac{m_{11}}{m_{11} + m_{10} + m_{01}}$$

$$J = \frac{2}{2+1+1} = \frac{2}{4} = 0.5$$

Different Classification Algorithms Based on the Distance Measures:

Distance-based classification algorithms rely on calculating the proximity (distance or similarity) between data points to assign labels or predict outcomes. These algorithms are intuitive and widely used due to their simplicity and effectiveness in various scenarios.

Key Distance-Based Classification Algorithms

1. **K-Nearest Neighbors (KNN)**

   Concept:
   KNN is a lazy learning algorithm that assigns a label to a test instance based on the labels of its *k*-nearest neighbors in the feature space.

   Distance Measures:

   - Euclidean Distance (default for continuous data).
   - Manhattan Distance (when features differ significantly in magnitude).
   - Minkowski Distance (a generalized form of Euclidean and Manhattan distances).
   - Cosine Similarity (for high-dimensional sparse data like text).

Steps:

1. Compute the distance between the test point and all training points.
2. Identify the *k*-nearest neighbors.
3. Perform majority voting (for classification) or compute the average (for regression).

Applications:
Text classification, recommendation systems, and image recognition.

2. Support Vector Machines (SVM) with Kernel Trick

Concept:
SVM is a classifier that finds the optimal hyperplane to separate data points from different classes.

Distance and Similarity:

- Linear kernel uses Euclidean geometry to maximize margin.
- Non-linear kernels (e.g., RBF kernel) use distance or similarity metrics to map data into higher-dimensional spaces.

Applications:
Handwriting recognition, image classification, and bioinformatics.

3. Distance-Weighted KNN

Concept:
An extension of KNN where the contribution of neighbors is weighted by their distance from the test point. Closer neighbors have more influence.

Weighting Formula:

$$w_i = \frac{1}{d(x, x_i)^p} \quad (p > 0)$$

Where $d(x, x_i)$ is the distance between the test point x and the neighbor x_i.

Applications: Real-time systems where the proximity of data points is critical.

4. Centroid-Based Classifiers

Concept:
Assigns labels to test data based on their proximity to class centroids (mean vectors for each class).

Algorithms:

- **Nearest Centroid Classifier (NCC):** Assigns a test instance to the nearest class centroid.
- **Rocchio Classifier:** A variant of NCC used in text categorization, based on prototype vectors.

Distance Measures: Euclidean distance, cosine similarity, or Mahala Nobis distance.
Applications: Text classification, anomaly detection, and clustering-based classification.

Key Features of Distance-Based Classification Algorithms

- **Interpretability:** Easy to understand and implement.
- **Flexibility:** Compatible with various distance/similarity measures based on data type (e.g., numerical, categorical, or mixed).
- **Non-parametric Nature:** No assumption about the underlying data distribution.
- **Scalability Issues:** May become computationally expensive for large datasets due to distance calculations.

Applications

- **Healthcare:** Disease prediction using patient similarity.
- **Finance:** Customer segmentation and credit scoring.
- **Image and Video Analysis:** Object recognition and scene classification.
- **Text Mining:** Document classification and spam detection.

K-Nearest Neighbor Classifier:

K-Nearest Neighbor (KNN) is a **supervised learning algorithm** used for both **classification** and **regression**. It is non-parametric, meaning it doesn't make any assumptions about the underlying data distribution, which makes it versatile for various applications. KNN works by analyzing the proximity or "closeness" of data points based on specific distance metrics.

In **classification**, KNN assigns a class label to a new data point based on the majority class of its nearest neighbors. For instance, if a data point has five nearest neighbors, and three of them belong to class A while two belong to class B, the algorithm will classify the point as class A.

Types of Problems Solved by KNN

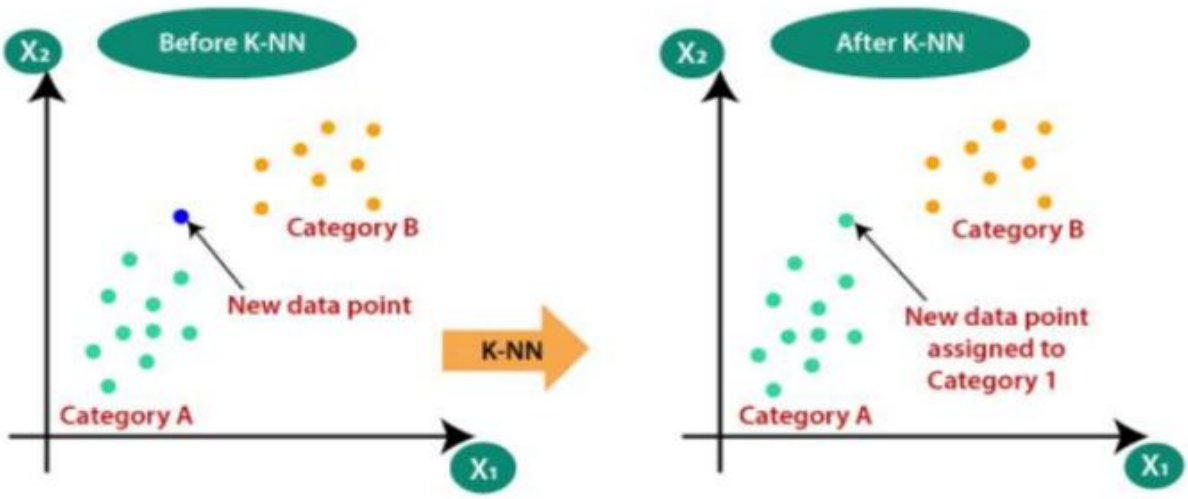
- **Classification**: Identifying which category a new observation belongs to.
- **Regression**: Predicting a continuous outcome based on similar observations.

KNN is widely used due to its simplicity and effectiveness in both small datasets and non-linear data distributions.

Algorithm Steps:

1. Select the number of neighbors **k**.
2. Calculate the distance between the query point and all other points in the dataset using a chosen distance metric.
3. Sort the distances in ascending order and select the top **k-nearest neighbors**.
4. **For classification**: Assign the query point the class of the majority of its neighbors.
5. **For regression**: Predict the value of the query point as the average of the k-nearest neighbors.

Working of KNN:



How to Select the Value of K in the K-NN Algorithm?

Choosing the correct value of **k** is critical to the performance of the KNN algorithm. A small k value can make the model too sensitive to noise, resulting in **overfitting**, while a large k value can oversimplify the model, causing **underfitting**.

Methods to Select the Optimal k:

- **Cross-Validation**: By splitting the dataset into training and validation sets and evaluating model performance across different values of k, the optimal k value can be determined based on which k produces the lowest error rate.
- **Common Values of k**: In practice, values of k such as 3, 5, or 7 are typically chosen. Smaller values of k allow the model to capture local patterns, while larger k values generalize better across the dataset.

Impact on Model Performance:

- **Too small k (e.g., k = 1)**: The model is highly sensitive to individual data points, leading to overfitting, as the prediction is based on just one point.
- **Too large k**: When k is too large, the model can become too smooth, losing important patterns in the data, leading to underfitting.

selecting an appropriate k value ensures the balance between model complexity and predictive accuracy.

Training Data:

Point	**(x, y)**	**Class**
A	(1, 1)	Red
B	(2, 1)	Red
C	(4, 3)	Blue
D	(5, 4)	Blue

Query point:

Q=(3,2)

Let **k = 3** and use **Euclidean distance**.

- **Step 1: Compute Distances**

$$d(Q, A) = \sqrt{(3-1)^2 + (2-1)^2} = \sqrt{4+1} = \sqrt{5} \approx 2.24$$
$$d(Q, B) = \sqrt{(3-2)^2 + (2-1)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$$
$$d(Q, C) = \sqrt{(3-4)^2 + (2-3)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.41$$
$$d(Q, D) = \sqrt{(3-5)^2 + (2-4)^2} = \sqrt{4+4} = \sqrt{8} \approx 2.83$$

🔹 Step 2: Choose k Nearest Neighbors

Distances (small → large):

- B: 1.41 → Red
- C: 1.41 → Blue
- A: 2.24 → Red
- D: 2.83 → Blue

Nearest **k = 3** neighbors → **B, C, A**

Classes: Red, Blue, Red

🔹 Step 3: Majority Vote

Red = 2
Blue = 1

✅ Predicted class for Q = **Red**

Using KNN with **k = 3**, the query point Q (3,2) is classified as **Red** based on majority voting of its 3 nearest neighbors.

Distance Metrics Used in KNN Algorithm

1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance

Advantages of KNN Algorithm:

1. Simplicity and Ease of Implementation
2. Non-Parametric Nature
3. Versatility
4. Robustness to Noisy Data

Disadvantages of KNN Algorithm

1. Computational Inefficiency
2. Sensitivity to the Choice of k and Distance Metric
3. Storage-Intensive
4. Struggles with Imbalanced Data

Applications of KNN Algorithm in Machine Learning

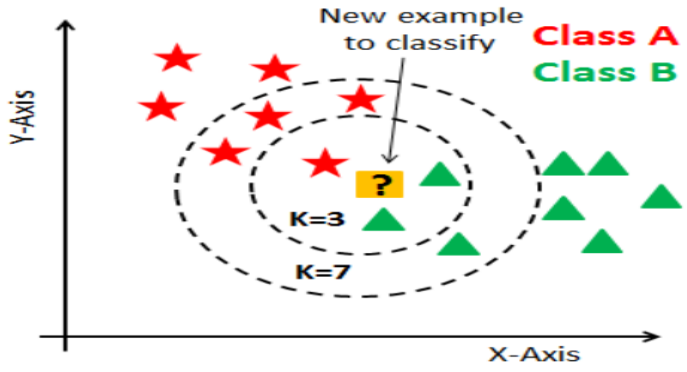
1. Text Classification
2. Image Recognition
3. Recommendation Systems
4. Medical Diagnosis

Radius Distance Nearest Neighbor Algorithm:

The Radius Distance Nearest Neighbor (RNN) algorithm is a variant of the traditional **K-Nearest Neighbors (KNN)** algorithm. Instead of using a fixed number of nearest neighbors, RNN classifies data points by considering all training instances within a given radius *r*. This radius defines a "neighborhood" around the query point. All neighbors whose distance from the query point is less than or equal to the radius *r* are included in the classification decision.

The key advantage of RNN over traditional KNN is its ability to adapt to varying data densities. In regions with higher data density, more neighbors will be considered, while in sparse regions, fewer neighbors will be considered.

r-Nearest neighbors are a modified version of the k-nearest neighbors. The issue with k-nearest neighbors is the choice of k. With a smaller k, the classifier would be more sensitive to outliers. If the value of k is large, then the classifier would be including many points from other classes. It is from this logic that we get the r near neighbors' algorithm.



- If k=3, new example is classified as green (Triangle) i.e class B
- If k=7, new example is classified as red (Star) i.e class A

Working of the Radius Distance Nearest Neighbor Algorithm

1. **Predefine the radius r:**
   The radius *r* is a user-defined parameter that specifies the maximum distance within which neighbors are considered for classification.

2. **Calculate the distance:**
   For each test point, compute the distance between the test point and all the training data points using a chosen distance metric (e.g., Euclidean, Manhattan).

3. **Identify neighbors:**
   Select all training data points whose distance from the test point is less than or equal to the radius *r*.

Assign class labels:

- If there are neighbors within the radius, perform majority voting to assign the most frequent class among the neighbors.
- If no neighbors are within the radius, the point may be classified as unclassifiable or assigned a default class.

Steps in RNN Classification

1. Choose the radius *r*.
2. Compute the distance between the query point and every training point.
3. Identify the neighbors: Include all training points whose distance to the query point is less than or equal to the radius *r*.
4. Perform majority voting for classification, or average the target values for regression.

KNN Regression:

KNN (K-Nearest Neighbors) regression is a type of instance-based learning or non-parametric regression algorithm. It uses the training data to make predictions. It does not learn a model that can be used to make predictions for new data points. The KNN algorithm doesn't make any assumptions about the training dataset.

It is a simple and straightforward approach to regression that can be used for both regression and classification problems. In KNN regression, we predict the value of a dependent variable for a data point based on the average or mean of the target values of its K nearest neighbors in the training data. The value of K is a user-defined hyperparameter. It determines the number of nearest neighbors used to make the prediction.

The K-Nearest Neighbors algorithm is a scalable approach to regression. This is a good choice for large datasets as well as small datasets. Also, we can use KNN regression for both continuous and categorical target variables (as in KNN classification). However, it is important to choose the right value of K, as a small value can lead to overfitting and a large value can lead to underfitting.

KNN Regression Algorithm

The KNN regression algorithm can be broken down into the following steps:

1. **Choose a value for K**: We first choose a value for K. This determines the number of nearest neighbors used to make the prediction.
2. **Calculate the distance**: After choosing K, we calculate the distance between each data point in the training set and the target data point for which a prediction is being made. For this, we can use a variety of distance metrics, including Euclidean distance, Manhattan distance, or Minkowski distance.
3. **Find the K nearest neighbors**: After calculating the distance between the existing data points and the new data point, we identify K nearest neighbors by selecting the K data points nearest to the new data point.
4. **Calculate the prediction**: After finding the neighbors, we calculate the value of the dependent variable for the new data point. For this, we take the average of the target values of the K nearest neighbors.

Hence, after executing the above steps, we can find the value of the dependent variable for any set of data points. Let us now discuss a numerical example to understand the KNN regression algorithm in a better way.

KNN Regression Numerical Example

To discuss the numerical example of N-Nearset Neighbors Regression, we will use the following dataset.

Length	Weight	Cost
7	9	33
6	11	35
9	7	32
5	8	30
5	10	30

Dataset for KNN Regression

In the above dataset, we have 5 data points. The dataset contains the length and weight of metal rods along with their cost. Now, suppose that we want to calculate the cost for a rod with a length of 7 and a weight of 8. For this, we will use the following steps.

First, we will decide on the value of K. We will take 3 as the number of closest neighbors used to decide the cost of the input data point.

Next, we will calculate the distance of the new data point i.e. (7, 8) to all the existing points in the dataset. Here, we will use the Euclidean distance measure

Point	Distance from (7,8)
(7, 9)	1.0
(6, 11)	3.16
(9, 7)	2.23
(5, 8)	2.0
(5, 10)	2.82

Distance Table

Now, we have found the distance of all the data points in the dataset from the point (7, 8). Next, we have to find the three closest points in the dataset. For this, we will sort the points according to their distances from (7, 8). The result is tabulated below.

Point	Distance from (7,8)
(7, 9)	1
(5, 8)	2
(9, 7)	2.23
(5, 10)	2.82
(6, 11)	3.16

Sorted Distance Table

In the above table, you can observe that the three points closest to (7, 8) are (7, 9), (5, 8), and (9, 7). These points have costs of 33, 30, and 32.

To calculate the cost of a rod with length 7 and weight 8, we can take the average of the above costs. Hence, the cost for (7, 8) will be 31.67.

Performance of Classifiers:

Classifier performance measures a machine learning model's predictive accuracy using metrics like precision, recall, and F1-score to determine how well it categorizes data into classes. Key metrics are derived from a confusion matrix—true positives, true negatives, false positives, and false negatives—which evaluate models beyond simple accuracy.

Confusion matrix:

Confusion matrix is a simple table used to measure how well a classification model is performing. It compares the predictions made by the model with the actual results and shows where the model was right or wrong. This helps you understand where the model is making mistakes so you can improve it. It breaks down the predictions into four categories:

- **True Positive (TP):** The model correctly predicted a positive outcome i.e the actual outcome was positive.
- **True Negative (TN):** The model correctly predicted a negative outcome i.e the actual outcome was negative.
- **False Positive (FP):** The model incorrectly predicted a positive outcome i.e the actual outcome was negative. It is also known as a Type I error.
- **False Negative (FN):** The model incorrectly predicted a negative outcome i.e the actual outcome was positive. It is also known as a Type II error.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Fig: Confusion matrix

It also helps calculate key measures like **accuracy**, **precision** and **recall** which give a better idea of performance especially when the data is imbalanced.

Classification problems aim to predict discrete categories. To evaluate the performance of classification models, we use the following metrics:

1. Accuracy

Accuracy shows how many predictions the model got right out of all the predictions. It gives idea of overall performance but it can be misleading when one class is more dominant over the other. For example, a model that predicts the majority class correctly most of the time might have high accuracy but still fail to capture important details about other classes. It can be calculated using the below formula:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

2. Precision

Precision focus on the quality of the model's positive predictions. It tells us how many of the "positive" predictions were actually correct. It is important in situations where false positives need to be minimized such as detecting spam emails or fraud. The formula of precision is:

$$\text{Precision} = \frac{TP}{TP+FP}$$

3. Recall

Recall measures how good the model is at predicting positives. It shows the proportion of true positives detected out of all the actual positive instances. High recall is essential when missing positive cases has significant consequences like in medical tests.

$$\text{Recall} = \frac{TP}{TP+FN}$$

4. F1-Score

F1-score combines precision and recall into a single metric to balance their trade-off. It provides a better sense of a model's overall performance particularly for imbalanced datasets. It is helpful when both false positives and false negatives are important though it assumes precision and recall are equally important but, in some situations, one might matter more than the other.

$$\text{F1-Score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Example:

Suppose we build a classifier to detect **Spam emails**.

Out of 100 emails:

- TP = 40 (Spam correctly detected)
- FP = 10 (Not-spam wrongly marked spam)
- FN = 5 (Spam wrongly marked not-spam)
- TN = 45 (Not-spam correctly detected)

Confusion Matrix

	Predicted: Spam	**Predicted: Not Spam**
Actual: Spam	TP = 40	FN = 5
Actual: Not Spam	FP = 10	TN = 45

1. **Accuracy:**

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Accuracy = \frac{40 + 45}{100} = 0.85 = 85\%$$

2. **Precision:** How many predicted positives are correct?

$$Precision = \frac{TP}{TP + FP}$$

$$Precision = \frac{40}{40 + 10} = 0.80 = 80\%$$

3. **Recall(sensitivity/TPR):** How many actual positives are found?

$$Recall = \frac{TP}{TP + FN}$$

$$Recall = \frac{40}{40 + 5} = 0.888 = 88.8\%$$

4. **F1-Score:**

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$F1 = 2 \times \frac{0.8 \times 0.888}{0.8 + 0.888} = 2 \times \frac{0.7104}{1.688} = 0.842$$

- If data is **imbalanced**, accuracy can be misleading.
  Example: If 95% emails are not spam, a dumb model saying "not spam" always gives 95% accuracy
- So, we use **Precision, Recall, F1-score, ROC-AUC**.

Performance of Regression Algorithms:

Regression metrics are used to evaluate models that predict continuous numerical values. The key metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2)

Types of Regression Metrics

Some common regression metrics in scikit-learn with examples

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- R-squared (R^2) Score
- Root Mean Squared Error (RMSE)

1. **Mean Absolute Error (MAE):** Represents the average magnitude of errors between actual and predicted values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

2. **Mean Squared Error (MSE):** The average of the squared errors, emphasizing larger errors.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

3. **Root Mean Squared Error (RMSE):** The square root of MSE, putting the error back into the original units for easier interpretation.

$$RMSE = \sqrt{MSE}$$

4. **R-squared (R^2 or Coefficient of Determination):** Indicates the proportion of the variance in the dependent variable explained by the model, ranging from 0 to 1.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Where:

- $SS_{res} = \sum(y_i - \hat{y}_i)^2$
- $SS_{tot} = \sum(y_i - \bar{y})^2$

✓ Range: $(-\infty, 1]$

✓ $R^2 = 1$ → perfect fit, $R^2 = 0$ → predicts mean only

Example data Set:

Data Point	Actual Value (y)	Predicted Value (y^)	|y-y^|
1	10	12	2
2	15	14	1
3	20	18	2
4	25	27	2
5	30	28	2

1. **Mean Absolute Error (MAE)**

$$MAE = \frac{1}{n}\sum|y - \hat{y}| = \frac{2+1+2+2+2}{5} = \frac{9}{5} = \boxed{1.8}$$

➢ Avg. absolute error

2. **Mean Squared Error (MSE)**

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2 = \frac{4+1+4+4+4}{5} = \frac{17}{5} = \boxed{3.4}$$

➢ Avg. squared error

3. **Root Mean Squared Error (RMSE)**

$$RMSE = \sqrt{MSE} = \sqrt{3.4} \approx \boxed{1.84}$$

- Error in same units

4. **R^2 Score (Coefficient of Determination)**

First find the mean of actual values:

$$\bar{y} = \frac{10 + 15 + 20 + 25 + 30}{5} = 20$$

Now:

$$SS_{res} = \sum (y - \hat{y})^2 = 17$$

$$SS_{tot} = \sum (y - \bar{y})^2 = (10-20)^2 + (15-20)^2 + (20-20)^2 + (25-20)^2 + (30-20)^2 = 100 + 25 + 0 + 25 + 100 = 250$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{17}{250} = 1 - 0.068 = \boxed{0.932}$$

- Model explains **93.2%** of the variance