# Measures of Distance in Data Mining

How similar are two data points?

# Outline

# Why Measure Distance?

# Why Measure Distance? (Theory)

**Distance measures** quantify the similarity or dissimilarity between data points. They are foundational in:

- Clustering (e.g., K-Means, Hierarchical)
- Classification (e.g., K-Nearest Neighbors)
- Anomaly detection
- Information retrieval and recommender systems

Choosing the right distance measure affects model accuracy and interpretability.

# Euclidean Distance

# Euclidean Distance (Theory)

**Euclidean distance** is the most common measure of straight-line distance in continuous space.

- Sensitive to magnitude and scale
- Assumes continuous, real-valued attributes

# Euclidean Distance (Math)

Given two points $A = (x_1, x_2, \ldots, x_n)$ and $B = (y_1, y_2, \ldots, y_n)$, the Euclidean distance is:

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

# Euclidean Distance (Example)

Given $A = (2, 4)$, $B = (5, 8)$:

$$d(A, B) = \sqrt{(2-5)^2 + (4-8)^2} = \sqrt{(-3)^2 + (-4)^2} = \sqrt{9 + 16} = \sqrt{25}$$

# Euclidean Distance (Use Case)

**Use Case:** K-Means Clustering

- Used to assign points to the nearest centroid
- Ideal when clusters are spherical and scales are normalized

# Manhattan Distance

# Manhattan Distance (Theory)

Also called **Taxicab** or **City Block Distance**:

- Measures distance by summing absolute differences
- Better for high-dimensional, sparse data

# Manhattan Distance (Math)

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

Where $A = (x_1, \ldots, x_n)$, $B = (y_1, \ldots, y_n)$

# Manhattan Distance (Example)

Given $A = (2, 4)$, $B = (5, 8)$:

$$d(A, B) = |2 - 5| + |4 - 8| = 3 + 4 = 7$$

# Manhattan Distance (Use Case)

**Use Case:** L1-Regularized models (e.g., Lasso Regression)

- Promotes sparsity
- Preferred when feature differences are linear or additive

# Jaccard Index

# Jaccard Index (Theory)

**Jaccard Similarity** measures overlap between sets.

- Works with binary or categorical data
- Good for market basket analysis, document similarity

# Jaccard Index (Math)

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

$$d(A, B) = 1 - J(A, B)$$

# Jaccard Index (Example)

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

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

$$J(A, B) = \frac{2}{6} = 0.33, \quad d(A, B) = 1 - 0.33 = 0.67$$

# Jaccard Index (Use Case)

**Use Case:** Recommender Systems

- Used to compare user interests (e.g., liked items)
- Suitable for sparse, binary user-item matrices

# Minkowski Distance

# Minkowski Distance (Theory)

Generalized distance metric that includes:

- Euclidean distance when $p = 2$
- Manhattan distance when $p = 1$

# Minkowski Distance (Math)

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

Where $p \in \mathbb{R},\ p \geq 1$

# Minkowski Distance (Example)

Given $A = (1, 2)$, $B = (4, 6)$, with $p = 3$:

$$d(A, B) = \left(|1 - 4|^3 + |2 - 6|^3\right)^{1/3} = (27 + 64)^{1/3} = (91)^{1/3} \approx 4.481$$

## Minkowski Distance (Use Case)

**Use Case:** Customizable distance metric in KNN

- Choose $p$ based on desired sensitivity
- Flexible for tuning similarity in various data distributions

# Cosine Similarity

# Cosine Similarity (Theory)

**Cosine Similarity** measures the cosine of the angle between two vectors.

- Focuses on orientation, not magnitude
- Ideal for high-dimensional, sparse data like text

# Cosine Similarity (Math)

$$\text{Cosine}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum x_i^2} \cdot \sqrt{\sum y_i^2}}$$

# Cosine Similarity (Example)

**Example 1: Similar Vectors**

Let $A_1 = (1, 2, 3)$, $B_1 = (2, 4, 6)$

$$\cos(\theta) = \frac{1 \cdot 2 + 2 \cdot 4 + 3 \cdot 6}{\sqrt{1^2 + 2^2 + 3^2} \cdot \sqrt{2^2 + 4^2 + 6^2}} = \frac{28}{\sqrt{14} \cdot \sqrt{56}} = 1$$

*Perfect similarity — linearly dependent or colinear (same direction)*

**Example 2: Dissimilar Vectors**

Let $A_2 = (1, 0)$, $B_2 = (0, 1)$

$$\cos(\theta) = \frac{1 \cdot 0 + 0 \cdot 1}{\sqrt{1^2 + 0^2} \cdot \sqrt{0^2 + 1^2}} = 0$$

*No similarity — orthogonal vectors.*

# Cosine Similarity (Use Case)

**Use Case:** Document Similarity in NLP

- Used to compare TF-IDF or word embeddings
- Common in search engines, chatbots, and plagiarism detection

# Summary Table

| Metric | Data Type | Math Form | Use Case |
|---|---|---|---|
| Euclidean | Numeric | $\sqrt{\sum(x_i - y_i)^2}$ | K-Means Clustering |
| Manhattan | Numeric | $\sum \lvert x_i - y_i \rvert$ | Lasso Regression |
| Jaccard | Set/Binary | $\frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$ | Recommender Systems |
| Minkowski | Numeric | $(\sum \lvert x_i - y_i \rvert^p)^{1/p}$ | Custom KNN |
| Cosine | Vector/High-dim | $\frac{A \cdot B}{\lVert A \rVert \lVert B \rVert}$ | Text Similarity |

# Some More Distance Measures

## What is Mahalanobis Distance?

- Measures distance between a point and a distribution
- Accounts for correlation between variables
- Scale-invariant
- More robust than Euclidean distance for multivariate data

# Intuition

- Euclidean distance treats all variables equally
- Mahalanobis considers variable correlation and scale
- Tells how many standard deviations a point is from the mean
- Adapts to the shape and orientation of the data

# Mathematical Formulation

Given:

- Point: $\mathbf{x}$
- Mean: $\boldsymbol{\mu}$
- Covariance: $\Sigma$

$$D_M(x) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})}$$

- $\Sigma^{-1}$: Inverse of covariance matrix
- Output is a scalar distance

# Properties

- Accounts for scale and correlation of features
- Reduces to Euclidean distance if $\Sigma = I$
- $D_M(x) = 0$: Point at mean
- $D_M(x) = 1$: One standard deviation away
- Useful for identifying multivariate outliers

# Applications

- Outlier detection
- Clustering (e.g., GMMs)
- Multivariate hypothesis testing
- Pattern recognition (e.g., face recognition)
- Quality control in industrial settings

# Example Calculation

- Mean: $\boldsymbol{\mu} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$
- Covariance: $\Sigma = \begin{bmatrix} 4 & 2 \\ 2 & 3 \end{bmatrix}$
- Point: $\mathbf{x} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$

$$D_M(x) = \sqrt{(x - \mu)^T \Sigma^{-1} (x - \mu)}$$

## Python Implementation

```
import numpy as np
from scipy.spatial import distance

data = np.array([[2, 3], [3, 5], [4, 8], [5, 11]])
x = np.array([3, 7])
mean = np.mean(data, axis=0)
cov = np.cov(data, rowvar=False)
inv_cov = np.linalg.inv(cov)

# Manual calculation
diff = x - mean
d_mahal = np.sqrt(diff.T @ inv_cov @ diff)

# Using scipy
d_scipy = distance.mahalanobis(x, mean, inv_cov)
```

# Tips & Best Practices

- Ensure data follows approximately multivariate normal distribution
- Handle singular covariance matrices (regularization, PCA)
- Normalize data if using with other distance metrics
- Be cautious with high-dimensional data (curse of dimensionality)

# What is MCC?

- Metric for evaluating binary (and multiclass) classification
- Considers: TP, TN, FP, FN
- Balanced measure, even with class imbalance
- Interpreted as a correlation coefficient

## Intuition

- Accuracy fails in imbalanced datasets
- MCC reflects the relationship between predictions and ground truth
- Robust and symmetric

**Values:**

- 1 = Perfect prediction
- 0 = No better than random
- -1 = Total disagreement

# MCC Formula

$$\text{MCC} = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

- If denominator $= 0$, MCC is defined as 0
- Normalized between -1 and 1

# Properties

- $-1 \leq \text{MCC} \leq 1$
- Handles class imbalance
- Invariant to label flipping
- More reliable than accuracy or F1-score alone

# Applications

- Medical diagnostics
- Fraud detection
- Binary classifiers in NLP, vision
- Model selection and benchmarking
- Extended to multiclass and multilabel problems

# Example Calculation

**Confusion Matrix:**

- TP = 70, TN = 50
- FP = 10, FN = 5

$$\text{MCC} = \frac{(70 \cdot 50) - (10 \cdot 5)}{\sqrt{(70+10)(70+5)(50+10)(50+5)}}$$
$$= \frac{3450}{\sqrt{19800000}} \approx 0.776$$

# Python Implementation

```
from sklearn.metrics import matthews_corrcoef

y_true = [1, 1, 0, 1, 0, 0, 1]
y_pred = [1, 0, 0, 1, 0, 1, 1]

mcc = matthews_corrcoef(y_true, y_pred)
print("MCC:", mcc)
```

# Manual Calculation

```
from math import sqrt

TP, TN, FP, FN = 70, 50, 10, 5
numerator = (TP * TN) - (FP * FN)
denominator = sqrt((TP + FP)*(TP + FN)*(TN + FP)*(TN + FN))
mcc = numerator / denominator if denominator != 0 else 0
print("MCC (manual):", mcc)
```

# Tips & Best Practices

- Use MCC for imbalanced datasets
- Report MCC with other metrics (e.g., AUC, F1)
- Useful in production to track model drift
- Preferable when false positives/negatives carry different costs

**Thank you!**