Distance Metrics in Machine Learning

Created by Daniel Zaldaña
https://x.com/ZaldanaDaniel

Euclidean Distance

Formula

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

When to use:

- Continuous data in low-dimensional space
- When scale and magnitude matter
- Default choice for clustering algorithms

Properties:

- Symmetric: $d(x, y) = d(y, x)$
- Non-negative: $d(x, y) \geq 0$
- Sensitive to outliers and scale

```
from sklearn.metrics.pairwise import (
    euclidean_distances,
    manhattan_distances,
    cosine_distances
)
import numpy as np
from typing import ndarray

def euclidean(
    x: ndarray,
    y: ndarray
) -> float:
    """Calculate Euclidean distance using sklearn."""
    # Reshape if needed for single vectors
    X = x.reshape(1, -1) if x.ndim == 1 else x
    Y = y.reshape(1, -1) if y.ndim == 1 else y
    return euclidean_distances(X, Y)[0, 0]
```

Manhattan Distance

Formula

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

When to use:

- Grid-like patterns (e.g., city blocks)
- When diagonal movement costs more
- Robust to outliers

```
def manhattan(
    x: ndarray,
    y: ndarray
) -> float:
    """Calculate Manhattan distance using sklearn."""
    X = x.reshape(1, -1) if x.ndim == 1 else x
    Y = y.reshape(1, -1) if y.ndim == 1 else y
    return manhattan_distances(X, Y)[0, 0]
```

Cosine Similarity

Formula

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

When to use:

- Text analysis and document similarity
- High-dimensional sparse data
- When direction matters more than magnitude

```
def cosine_similarity(
    x: ndarray,
    y: ndarray
) -> float:
    """Calculate cosine similarity using sklearn."""
    X = x.reshape(1, -1) if x.ndim == 1 else x
    Y = y.reshape(1, -1) if y.ndim == 1 else y
    return 1 - cosine_distances(X, Y)[0, 0]
```

Mahalanobis Distance

Formula

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})}$$

When to use:

- Correlated features
- Anomaly detection
- Scale-invariant clustering

```
from sklearn.covariance import EmpiricalCovariance

def mahalanobis(
    x: ndarray,
    y: ndarray,
    cov: ndarray = None
) -> float:
    """Calculate Mahalanobis distance using sklearn."""
```

```python
    X = x.reshape(1, -1) if x.ndim == 1 else x
    Y = y.reshape(1, -1) if y.ndim == 1 else y

    if cov is None:
        # Estimate covariance from data
        cov_estimator = EmpiricalCovariance()
        cov_estimator.fit(np.vstack([X, Y]))
        cov = cov_estimator.covariance_

    diff = X - Y
    inv_covmat = np.linalg.inv(cov)
    return np.sqrt(
        diff.dot(inv_covmat).dot(diff.T)
    )[0, 0]
```

Minkowski Distance

Formula

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

When to use:

- Generalizing distance metrics
- When you need to tune the influence of large differences
- Experimenting with different p-norms

```python
from sklearn.metrics import pairwise_distances

def minkowski(
    x: ndarray,
    y: ndarray,
    p: float = 2
) -> float:
    """Calculate Minkowski distance using sklearn."""
    X = x.reshape(1, -1) if x.ndim == 1 else x
    Y = y.reshape(1, -1) if y.ndim == 1 else y
    return pairwise_distances(
        X, Y, metric='minkowski', p=p
    )[0, 0]
```

Jaccard Distance

Formula

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

When to use:

- Binary or set-based data
- Comparing discrete features
- Document similarity with word sets

```python
def jaccard(
    x: ndarray,
    y: ndarray
) -> float:
    """Calculate Jaccard distance using sklearn."""
    X = x.reshape(1, -1) if x.ndim == 1 else x
    Y = y.reshape(1, -1) if y.ndim == 1 else y
    return pairwise_distances(
        X, Y, metric='jaccard'
    )[0, 0]
```

Hamming Distance

Formula

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

When to use:

- Categorical data
- Error detection in communication
- Comparing equal-length strings

```python
def hamming(
    x: ndarray,
    y: ndarray
) -> float:
    """Calculate Hamming distance using sklearn."""
    X = x.reshape(1, -1) if x.ndim == 1 else x
    Y = y.reshape(1, -1) if y.ndim == 1 else y
    return pairwise_distances(
        X, Y, metric='hamming'
    )[0, 0]
```