

## Lecture 2

Distance measurements

Scaling methods

### Distance Metrics

In machine learning models like k-nearest neighbors (KNN) or k-means clustering, distance metrics play a crucial role in measuring the similarity or dissimilarity between data points. Commonly used distance metrics include:

*Euclidean Distance:* This is the most common distance metric, measuring the straight-line distance between two points in Euclidean space.

*Manhattan Distance (Taxicab or City Block Distance):* Manhattan distance is the sum of the absolute differences between the coordinates of the points. It measures the distance along the grid-like paths formed by a city block.

*Minkowski Distance:* Minkowski distance is a generalization of both Euclidean and Manhattan distances. The parameter "p" determines the specific type of distance:

When  $p = 1$ , it reduces to Manhattan distance.

When  $p = 2$ , it reduces to Euclidean distance.

For other values of  $p$ , it's a generalization of both.

*Cosine Similarity:* Although not a distance metric per se, cosine similarity measures the cosine of the angle between two vectors, indicating their similarity in direction rather than magnitude.

*Hamming Distance:* Hamming distance measures the number of positions at which corresponding symbols are different between two strings of equal length. It is commonly used for categorical variables or binary data.

*Jaccard Distance:* Jaccard distance measures dissimilarity between two sets by dividing the size of their intersection by the size of their union. It is frequently used for comparing binary data or sets.

Less common distance metrics include:










*Chebyshev Distance (Maximum Value Distance):* Chebyshev distance is the maximum absolute difference between the coordinates of the points along any dimension. It measures similarity based on the maximum difference in any dimension.

*Mahalanobis Distance:* Mahalanobis distance takes into account the correlations between variables and is scaled by the inverse of the covariance matrix. It is sensitive to the covariance structure of the data and is useful when the data is not spherical.

*Levenshtein Distance (Edit Distance):* Levenshtein distance measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into the other. It's commonly used in natural language processing for spell checking, DNA analysis, and other string similarity tasks.

*Earth Mover's Distance (Wasserstein Distance):* Earth Mover's Distance measures the minimum cost of transforming one distribution into another. It is used in applications such as image comparison, where it measures the minimum amount of work needed to transform one image's distribution of pixel values into another's.

These are just a few examples of common and less common distance metrics used in machine learning. The choice of distance metric depends on the characteristics of the data, the specific task at hand, and the underlying assumptions of the model being used.

Picture	Method	Application	Features	Disadvantages	Formula
	<b>Euclidean Distance</b>	General distance measurement, Clustering, Classification, Regression	Measures the straight line distance between two points in n-dimensional space.	Sensitive to outliers, Can be affected by scale differences	O(n) Fast
	<b>Manhattan Distance</b>	Distance on grid networks, Routing algorithms, Image processing	Measures the distance between two points on a grid network, where movement is limited.	Ignores diagonal movement, not useful for high-dimensional data,	O(n) Fast
	<b>Cosine Similarity</b>	Text document clustering, Text analysis, Recommendation systems	Measures the cosine of the angle between two vectors	Ignores magnitude of vectors, Not useful for negative values or high degree of correlation data	O(n) Fast
	<b>Minkowski Distance</b>	General distance measurement	Measures the distance between two points in n-dimensional space, where r determines the metric used.	Sensitive to outliers	O(n) Fast
	<b>Chebyshev Distance</b>	Measuring maximum difference, Clustering, Anomaly detection	Measures the maximum difference between corresponding components of two vectors	Only applicable for continuous data, Sensitive to outliers, may not be as useful for highly correlated data	O(n) Fast
	<b>Hamming Distance</b>	Measuring string similarity, Error-correcting codes, DNA sequencing	Measures the number of positions at which the corresponding symbols are different.	Only for same length strings, May not be as useful for continuous data	O(n) Fast
	<b>Levenshtein Distance</b>	Measuring string similarity	Measures the minimum number of single-character edits required to transform one string into another.	More expensive for long strings	O(n^2) Slow
	<b>Jaccard Similarity</b>	Set similarity measurement, Text analysis, recommendation systems	Measures the similarity between two sets by comparing their intersection and union.	Ignores magnitude of sets, May not be as useful for continuous data	O(n) Fast
	<b>Sorensen-Dice Index</b>	Measuring similarity of sets, Ecology, Biology, Genetics	Measures the similarity between two sets	May not be as useful for continuous data and Ignores magnitude of sets	O(n) Fast

Let's look at the code for some of these.

### Euclidean Distance

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

```
euclidean_distance <- function(x, y) {
```

```

sqrt(sum((x - y)^2))
}

```

```

# Select the first two rows (cars) and all columns except the 'cyl' column

```

```

car1 <- mtcars[1, -2]

```

```

car2 <- mtcars[2, -2]

```

```

# Calculate Euclidean distance

```

```

euclidean_distance(car1, car2)

```

### Manhattan Distance

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

```

manhattan_distance <- function(x, y) {
  sum(abs(x - y))
}

```

```

# Calculate Manhattan distance

```

```

manhattan_distance(car1, car2)

```

### Minkowski's distance

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

```

minkowski_distance <- function(x, y, p) {
  sum(abs(x - y)^p)^(1/p)
}

```

```

# Calculate Minkowski distance with p = 3

```

```

minkowski_distance(car1, car2, p = 3)

```

### Chebyshev's distance

$$d(x_i, z_j) = \max_{k=1,2,\dots,d} |x_{i,k} - z_{j,k}|$$

$$k = 1, 2, \dots, d$$

```

chebyshev_distance <- function(x, y) {
  max(abs(x - y))
}

```

```

# Calculate Chebyshev distance

```

```

chebyshev_distance(car1, car2)

```

### Cosine Similarity

$$\text{similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2 \sum_{i=1}^n B_i^2}}$$

```
cosine_similarity <- function(x, y) {
  sum(x * y) / (sqrt(sum(x^2)) * sqrt(sum(y^2)))
}
```

```
# Calculate Cosine Similarity
cosine_similarity(car1, car2)
```

## Mahalanobis Distance

$$\begin{aligned} d_M(x, y) &= \sqrt{(x - y)^T S^{-1} (x - y)} \\ &= \sqrt{\begin{bmatrix} x_1 - y_1 & x_2 - y_2 \end{bmatrix} \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 \\ 0 & \frac{1}{\sigma_2^2} \end{bmatrix} \begin{bmatrix} x_1 - y_1 \\ x_2 - y_2 \end{bmatrix}} \\ &= \sqrt{\begin{bmatrix} \frac{x_1 - y_1}{\sigma_1^2} & \frac{x_2 - y_2}{\sigma_2^2} \end{bmatrix} \begin{bmatrix} x_1 - y_1 \\ x_2 - y_2 \end{bmatrix}} \\ &= \sqrt{\frac{(x_1 - y_1)^2}{\sigma_1^2} + \frac{(x_2 - y_2)^2}{\sigma_2^2}} \end{aligned}$$

```
# Load the dataset
data("mtcars")
```

```
# Select the first two rows (cars) and all columns except the 'cyl' column
car1 <- as.numeric(mtcars[1, -2])
car2 <- as.numeric(mtcars[2, -2])
```

```
# Define Mahalanobis distance function
```

```
mahalanobis_distance <- function(x, y, cov_matrix) {
  diff <- as.matrix(x - y, ncol=1) # Convert the difference to a column matrix
  sqrt(t(diff) %*% solve(cov_matrix) %*% diff)
}
```

```
# Calculate the covariance matrix of the entire dataset (excluding 'cyl')
cov_matrix <- cov(mtcars[, -2])
```

```
# Calculate Mahalanobis distance
```

```
mahalanobis_distance(car1, car2, cov_matrix)
```

## Jaccard Distance

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

```
jaccard_distance <- function(x, y) {
  intersection <- sum(x & y)
  union <- sum(x | y)
  1 - (intersection / union)
}
```

```
}
```

```
# Create binary vectors based on mpg (above/below threshold)
```

```
binary_car1 <- as.numeric(mtcars[1, ] > median(mtcars$mpg))
```

```
binary_car2 <- as.numeric(mtcars[2, ] > median(mtcars$mpg))
```

```
# Calculate Jaccard distance
```

```
jaccard_distance(binary_car1, binary_car2)
```

## Hamming Distance

Hamming Distance

$$D_H = \sum_{i=1}^k |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

```
hamming_distance <- function(x, y) {  
  sum(x != y)  
}
```

We can apply one of the metrics to the entire dataset:

```
# Load the dataset
```

```
data("mtcars")
```

```
# Select only the numeric columns (excluding categorical or identifiers if necessary)
```

```
# Here we use the entire dataset without 'cyl' for example purposes.
```

```
mtcars_numeric <- mtcars[, -2]
```

```
# Define Euclidean distance function
```

```
euclidean_distance <- function(x, y) {
```

```
  sqrt(sum((x - y)^2))
```

```
}
```

```
# Initialize an empty matrix to store the distances
```

```
num_rows <- nrow(mtcars_numeric)
```

```
distance_matrix <- matrix(NA, nrow = num_rows, ncol = num_rows)
```

```
# Calculate Euclidean distance for each pair of rows (cars)
```

```
for (i in 1:num_rows) {
```

```
  for (j in 1:num_rows) {
```

```
    distance_matrix[i, j] <- euclidean_distance(mtcars_numeric[i, ], mtcars_numeric[j, ])
```

```
  }
```

```
}
```

```
# Optionally add row and column names to the matrix for easier interpretation
```

```
rownames(distance_matrix) <- rownames(mtcars_numeric)
```

```
colnames(distance_matrix) <- rownames(mtcars_numeric)

# View the distance matrix
print(distance_matrix)
```

## Scaling Methods

We want to also look at scaling methods in data preprocessing, why they are used, and how to implement them from scratch in R using the mtcars dataset. We'll cover the following scaling methods:

- Min-Max Scaling (Normalization)
- Z-Score Scaling (Standardization)
- Max Abs Scaling
- Robust Scaling
- Decimal Scaling

**What is Scaling?** Scaling is a preprocessing step in machine learning where we adjust the range or distribution of features so they are comparable and suitable for algorithms sensitive to feature magnitudes. This step is particularly important for distance-based models like KNN or SVM, where feature magnitude can heavily influence the model.

## Why Scale Data?

- Ensures that features contribute equally to the model.
- Improves convergence speed for gradient-based algorithms.
- Reduces the impact of units of measurement.

## Min-Max Scaling (Normalization)

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

This scaling method transforms the data into the range [0,1].

```
# Function to perform Min-Max Scaling
min_max_scaling <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

# Apply to mtcars dataset
mtcars_min_max <- as.data.frame(lapply(mtcars, min_max_scaling))

# View scaled data
head(mtcars_min_max)
```

## Z-Score Scaling (Standardization)

$$z = \frac{(x - \bar{x})}{s}$$

This is slightly misnamed, since we are using sample means and standard deviations, not populations means (so this is really t-score scaling), but if you know the mean and standard deviations of the population, you can use that.

# Function to perform Z-Score Scaling

```
z_score_scaling <- function(x) {  
  (x - mean(x)) / sd(x)  
}
```

# Apply to mtcars dataset

```
mtcars_z_scaled <- as.data.frame(lapply(mtcars, z_score_scaling))
```

# View scaled data

```
head(mtcars_z_scaled)
```

### Max Abs Scaling

$$X'_i = \frac{X_i}{\text{abs}(X_{\max})}$$

# Function to perform Max Abs Scaling

```
max_abs_scaling <- function(x) {  
  x / max(abs(x))  
}
```

# Apply to mtcars dataset

```
mtcars_max_abs <- as.data.frame(lapply(mtcars, max_abs_scaling))
```

# View scaled data

```
head(mtcars_max_abs)
```

### Robust Scaling

$$X_{\text{new}} = \frac{X - X_{\text{median}}}{\text{IQR}}$$

# Function to perform Robust Scaling

```
robust_scaling <- function(x) {  
  (x - median(x)) / IQR(x)  
}
```

# Apply to mtcars dataset

```
mtcars_robust_scaled <- as.data.frame(lapply(mtcars, robust_scaling))
```

# View scaled data

```
head(mtcars_robust_scaled)
```

### Decimal Scaling

$$v' = \frac{v}{10^j},$$

- where  $j$  is the smallest integer such that  $\text{new-max}_A < 1$ .

# Function to perform Decimal Scaling

```
decimal_scaling <- function(x) {
```

```
j <- ceiling(log10(max(abs(x))))
x / 10^j
}

# Apply to mtcars dataset
mtcars_decimal_scaled <- as.data.frame(lapply(mtcars, decimal_scaling))

# View scaled data
head(mtcars_decimal_scaled)
```

### Practical Considerations

When to Use Each Method:

*Min-Max Scaling*: Use when features are on different scales, but you want to preserve the relationships between values.

*Z-Score Scaling*: Use when data follows a normal distribution and you want to standardize it.

*Max Abs Scaling*: Use when data contains both positive and negative values.

*Robust Scaling*: Use when your data contains outliers.

*Decimal Scaling*: Use when you need a quick normalization without worrying about the distribution of values.

### Impact on Machine Learning Models:

- Distance-based Models (e.g., KNN): Scaling is crucial as these models are sensitive to feature magnitudes.
- Gradient-based Models (e.g., Logistic Regression, Neural Networks): Scaling improves the convergence of optimization algorithms.

Resources:

1. <https://numerics.mathdotnet.com/Distance>
2. <https://medium.com/@eskandar.sahel/exploring-common-distance-measures-for-machine-learning-and-data-science-a-comparative-analysis-ea0216c93ba3>
3. <https://datatricks.co.uk/feature-scaling-in-r-five-simple-methods>