Distances and Time Series

We will start building some tools for making comparisons of data objects with particular attention to time series.

Working with data, we can encounter a wide variety of different data objects

Lecture Overview

We cover the following topics in today's lecture

Time Series Data

Some examples of time series data

Feature space representation

Usually a data object consists of a set of attributes.

These are also commonly called **features.**

- (“J. Smith”, 25, $ 200,000)
- (“M. Jones”, 47, $ 45,000)

If all d dimensions are real-valued then we can visualize each data object as a point in a d-dimensional vector space.

- `(25, USD 200000)` $\rightarrow \begin{bmatrix} 25 \\ 200000 \end{bmatrix}$.

Likewise If all features are binary then we can think of each data object as a binary vector in vector space.

The space is called **feature space.**

One-hot encoding

Vector spaces are such a useful tool that we often use them even for non-numeric data.

For example, consider a categorical variable that can be only one of “house”, “tree”, or “moon”. For such a variable, we can use a **one-hot** encoding.

Why not alternative encoding?

Why would we not encode as follows:

- `house`: $[1]$
- `tree`: $[2]$
- `moon`: $[3]$

🎯 Class Activity: Feature Encoding

Task: Work in pairs to encode the following data into feature vectors.

Data to encode:

- Person A: (age=30, salary=$75,000, city="New York", education="PhD")
- Person B: (age=25, salary=$50,000, city="Boston", education="Masters")
- Person C: (age=35, salary=$90,000, city="New York", education="Bachelors")

Instructions:

1. Create one-hot encodings for categorical variables (city: NY, Boston, LA; education: Bachelors, Masters, PhD)
2. Normalize the numerical features (age: divide by 100, salary: divide by 100,000)
3. Write out the complete feature vectors for all three people
4. Calculate the Euclidean distance between Person A and Person B

Discussion: Why might we normalize numerical features? What does the distance tell us

Encodings

We will see many other encodings that take non-numeric data and encode them into vectors or matrices.

For example, there are vector or matrix encodings for

Matrix representation of data

We generally store data in a matrix form as

$$\overbrace{\begin{bmatrix} x_{11} & \cdots & x_{1j} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i1} & \cdots & x_{ij} & \cdots & x_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mj} & \cdots & x_{mn} \end{bmatrix}}^{n \text{ features}} \left\{ \vphantom{\begin{bmatrix} x_{11} & \cdots & x_{1j} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i1} & \cdots & x_{ij} & \cdots & x_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mj} & \cdots & x_{mn} \end{bmatrix}} \right. m \text{ data objects}$$

The number of rows is denoted by m and the number of columns by n. The rows are instances or records of data and the columns are the features.

Metrics

A metric is a function $d(x, y)$ that satisfies the following properties.

Distance Matrices

The distance matrix is defined as

$$
\begin{array}{c}
\\
m \text{ data objects}
\end{array}
\overbrace{\left\{\begin{array}{c}
\\
\begin{bmatrix}
0 & & & & \\
d(x_1, x_2) & 0 & & & \\
d(x_1, x_3) & d(x_2, x_3) & 0 & & \\
\vdots & \vdots & \vdots & \ddots & \\
d(x_1, x_m) & d(x_2, x_m) & \cdots & d(x_{m-1}, x_m)
\end{bmatrix}
\end{array}\right.}^{m \text{ data objects}}
$$

where x_i denotes the i-th row of the data matrix X.

Norms

Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ and $a \in \mathbb{R}$. The vector function $p(\mathbf{v})$ is called a **norm** if

ℓ_p norm

A general class of norms are called ℓ_p norms, where $p \geq 1$.

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{d} |x_i|^p \right)^{\frac{1}{p}}.$$

The corresponding distance that an ℓ_p norm defines is called the *Minkowski distance.*

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

ℓ_2 norm

A special – but very important – case is the ℓ_2 norm.

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{d} |x_i|^2}.$$

We've already mentioned it: it is the **Euclidean** norm.

The distance defined by the ℓ_2 norm is the same as the Euclidean distance between two vectors $\mathbf{x}, \mathbf{y}$.

$$\|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}.$$

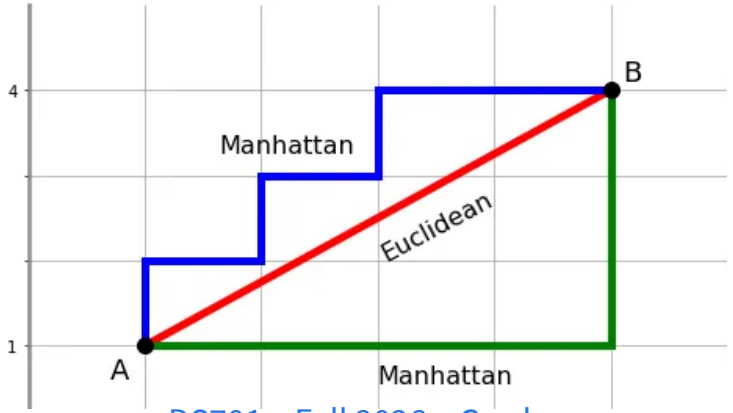
ℓ_1 norm

Another important special case is the ℓ_1 norm.

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{d} |x_i|.$$

This defines the **Manhattan** distance, or (for binary vectors), the **Hamming** distance:

$$\|\mathbf{x} - \mathbf{y}\|_1 = \sum_{i=1} |x_i - y_i|.$$



ℓ_∞ norm

If we take the limit of the ℓ_p norm as p gets large we get the ℓ_∞ norm.

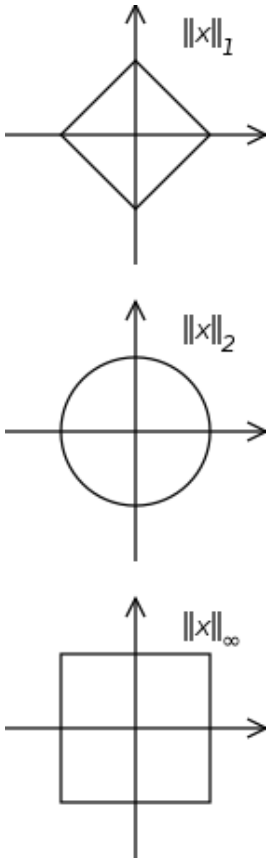
We have that

$$\|\mathbf{x}\|_\infty = \max_i |x_i|.$$

Visualizing norms

Here is the notion of a “circle” under each of three norms.

That is, for each norm, the set of vectors having norm 1, or distance 1 from the origin.



$$|x_1| + |x_2| = 1$$

$$\sqrt{x_1^2 + x_2^2} = 1$$

$$\max(|x_1|, |x_2|) = 1$$

What norm should I use?

The choice of norm depends on the characteristics of your data and the problem you're trying to solve.

ℓ_1 norm

- Use when your data is sparse.
- Robust to outliers.

ℓ_2 norm

- Use when measuring distances in Euclidean space.
- Smooth and differentiable.

ℓ_∞ norm

- Use when you need uniform bounds.
- Limits maximum deviation.

🎯 Class Activity: Norm Comparison

Task: Compare how different norms measure distance between the same two points.

Given: Two 2D points A = (3, 4) and B = (1, 1)

Instructions:

1. Calculate the distance between A and B using:
   - ℓ_1 norm (Manhattan distance)
   - ℓ_2 norm (Euclidean distance)
   - ℓ_∞ norm (Chebyshev distance)
2. Now consider points C = (0, 0) and D = (1, 0)
   - Calculate all three distances again
   - Which norm gives the smallest distance? Largest?
3. **Think-pair-share**:
   - In what real-world scenarios would you prefer each norm?

Similarity and Dissimilarity Functions

Similarity functions quantify how similar two objects are. The higher the similarity score, the more alike the objects.

Dissimilarity functions quantifies the difference between two objects. The higher the dissimilarity score, the more different the objects are.

Similarity

We know that the inner product of two vectors can be used to compute the **cosine of the angle** between them

$$\cos(\theta) = \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} \equiv \cos(\mathbf{x}, \mathbf{y}).$$

This value is

- close to 1 when $\mathbf{x} \approx \mathbf{y}$ ($\theta \approx 0$) and
- close to 0 when $\mathbf{x}$ and $\mathbf{y}$ are orthogonal ($\theta \approx 90°$).

We can use this formula to define a **similarity** function called the **cosine similarity** $\cos(\mathbf{x}, \mathbf{y})$.

Abuse of notation? 🤔

Dissimilarity

🎯 Class Activity: Similarity Calculations

Task: Calculate different similarity measures between two documents represented as word frequency vectors.

Given: Two documents with word counts:

- Document A: [cat: 3, dog: 2, bird: 1, fish: 0]
- Document B: [cat: 1, dog: 3, bird: 0, fish: 2]

Instructions:

1. Convert to normalized vectors (divide by total word count)
2. Calculate cosine similarity between the documents
3. Calculate the dissimilarity using: d = 1 - cosine_similarity
4. **Quick discussion**:
    - What does a cosine similarity of 0.5 mean?

Solution

▶ Code

```
Document Similarity Analysis
====================================
Original word frequency vectors:
Document A: [3 2 1 0] (cat: 3, dog: 2, bird: 1, fish: 0)
Document B: [1 3 0 2] (cat: 1, dog: 3, bird: 0, fish: 2)

Step 1: Normalized vectors (divided by total word count)
Document A normalized: [0.5        0.33333333 0.16666667 0.        ]
Document B normalized: [0.16666667 0.5        0.         0.33333333]
Sum of normalized A: 1.000000
Sum of normalized B: 1.000000

Step 2: Cosine similarity calculation
Cosine similarity: 0.642857

Manual verification:
Dot product: 0.250000
Norm of A: 0.623610
Norm of B: 0.623610
Cosine similarity (manual): 0.642857
```

Discussion points

Bit vectors and Sets

When working with bit vectors, the ℓ_1 metric is commonly used and is called the **Hamming** distance.

x	0	1	0	0	1	0	0	1	0
y	1	0	0	0	0	1	0	1	1

$$L_1(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i| \right)$$

This has a natural interpretation: “how well do the two vectors match?”

Or: “What is the smallest number of bit flips that will convert one vector into the other?”

Hamming distance

x	0	1	0	0	1	0	0	1	0
y	1	0	0	0	0	1	0	1	1

$$L_1(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i| \right)$$

Hamming distance with sets

Consider the case in which the bit vector is being used to represent a *set*.

Definition: A *set* is an unordered collection of distinct elements.

In that case, Hamming distance measures the **size of the set difference.**

For example, consider two documents. We will use bit vectors to represent the sets of words in each document.

Jaccard similarity

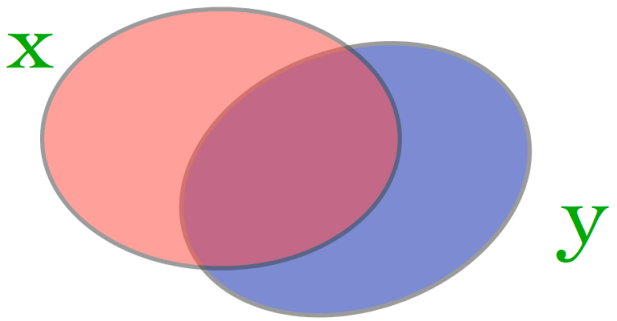
This leads to the *Jaccard* similarity or *Intersection over Union (IoU)*:

$$J_{Sim}(\mathbf{x}, \mathbf{y}) = \frac{|\mathbf{x} \cap \mathbf{y}|}{|\mathbf{x} \cup \mathbf{y}|}.$$

This takes on values from 0 to 1, so a natural dissimilarity metric is $1 - J_{Sim}()$.

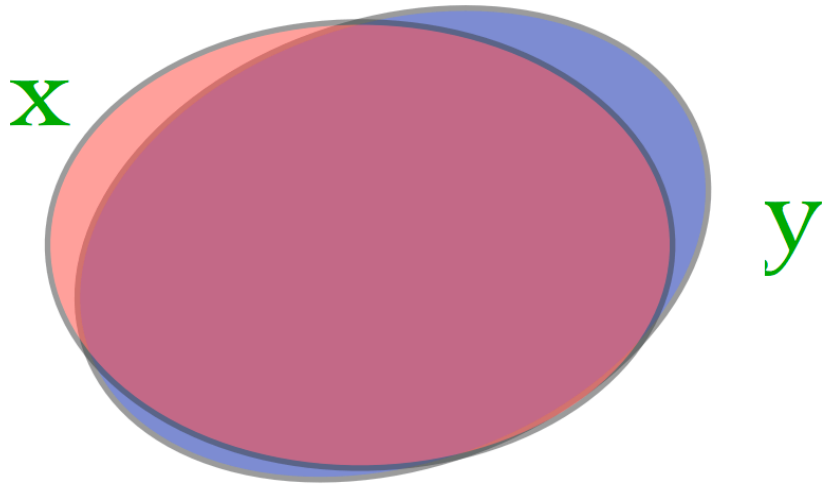
In fact, this is a **metric!**

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



Jaccard Similarity Example 1

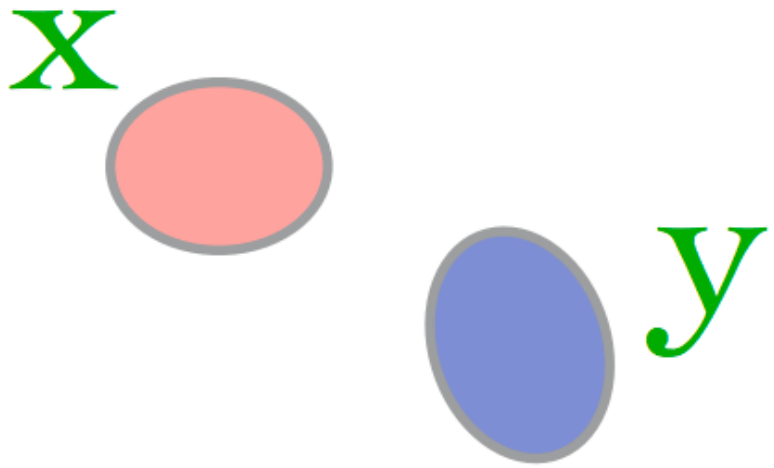
Case 1: Very large almost identical documents.



Here $J_{Sim}(\mathbf{x}, \mathbf{y})$ is almost 1.

Jaccard Similarity Example 2

Case 2: Very small disjoint documents.



Here $J_{Sim}(\mathbf{x}, \mathbf{y})$ is 0.

🎯 Class Activity: Jaccard Similarity Practice

Task: Calculate Jaccard similarity for different document scenarios.

Given: Three documents represented as sets of unique words:

- Document 1: {apple, banana, cherry, date}
- Document 2: {apple, banana, cherry, elderberry}
- Document 3: {grape, kiwi, lemon, mango}

Instructions:

1. Calculate Jaccard similarity between:
    - Document 1 and Document 2
    - Document 1 and Document 3
    - Document 2 and Document 3
2. **Think-pair-share:**

Time Series

A time series is a sequence of real numbers, representing the measurements of a real variable at (possibly equal) time intervals.

Some examples are

Similarity of Time Series

Suppose we wish to compare the following time series.

Norm-based Similarity Measures

We could just view each sequence as a vector.

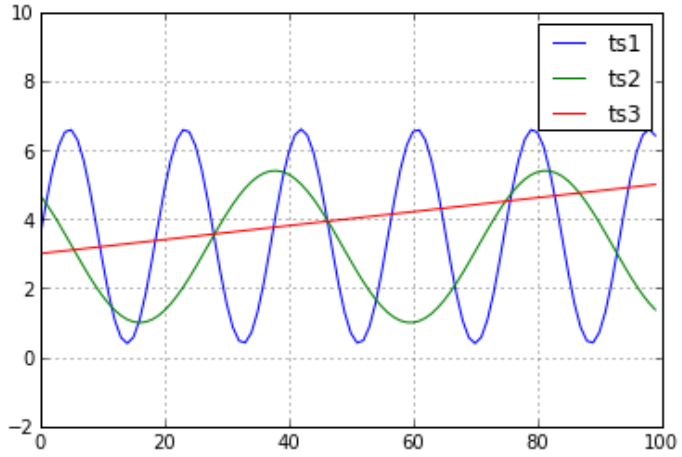
Then we could use a p-norm, e.g., ℓ_1, ℓ_2, or ℓ_p to measure similarity.

Beware of Norm-based Similarity

Disadvantage

1. May not be **meaningful!**

We may believe that $\mathbf{ts1}$ and $\mathbf{ts2}$ are the most “similar” pair of time series.

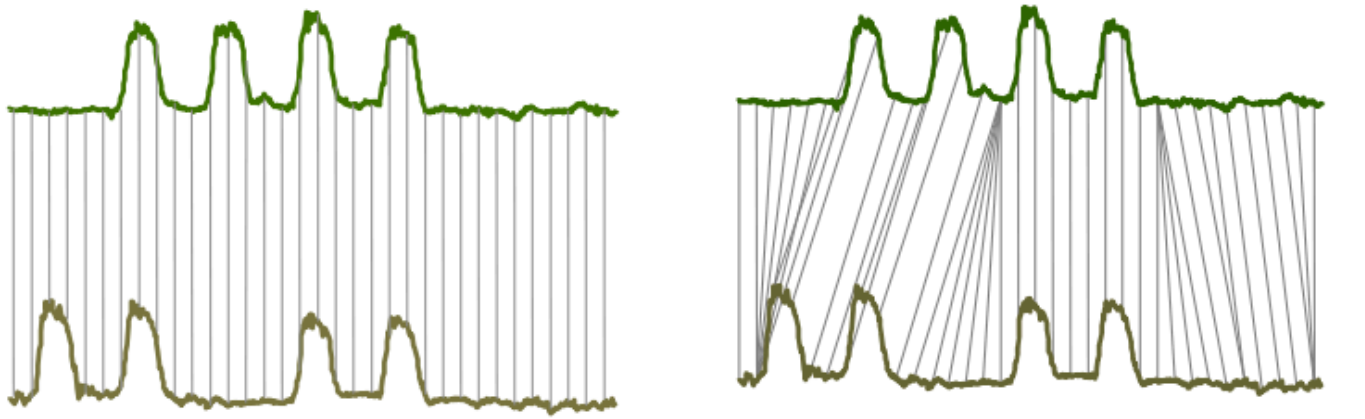


Feature Engineering

Feature engineering is the art of computing some derived measure from your data object that makes the important properties usable in a subsequent step.

Bump Hunting

One case that arises often is something like the following: “bump hunting”



Images from: Eamonn Keogh’s slides

Both time series have the same key characteristics: four bumps.

But a one-to-one match (ala Euclidean distance) will not detect the similarity.

A solution to this is called **dynamic time warping.**

Dynamic Time Warping

The basic idea is to allow stretching or compressing of signals along the time dimension.

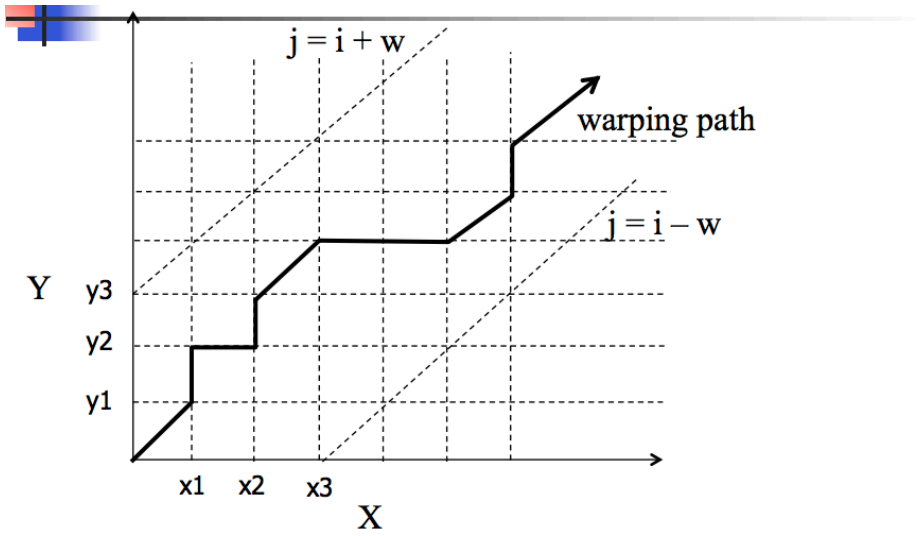
Visualizing DTW

There is a simple way to visualize this algorithm.

Consider a matrix M where $M_{ij} = |x_i - y_j|$.

M measures the amount of error we get if we match x_i with y_j.

So we seek a **path, starting from lower left, through M that minimizes the total error.**



DTW restrictions

The basic restrictions on path are:

Monotonicity

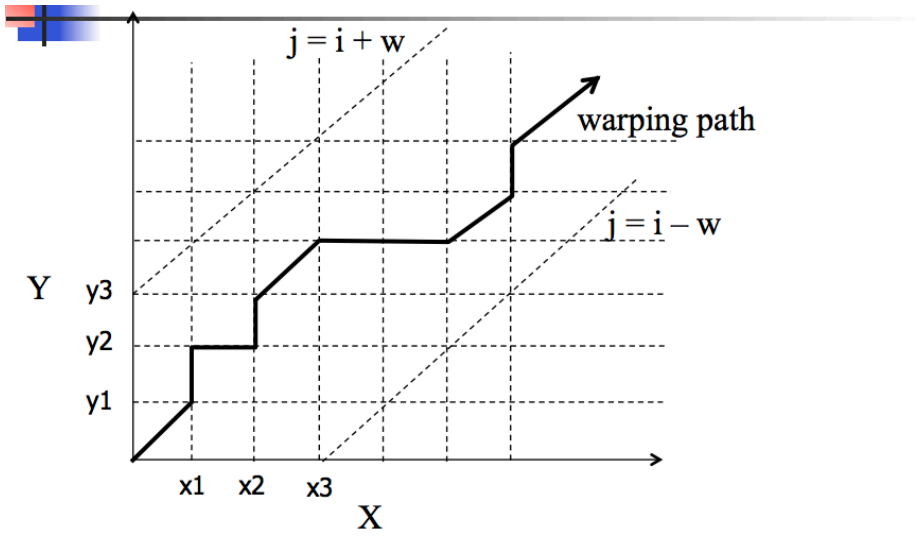
- The path should not go down or to the left.

Continuity

- No elements may be skipped in a sequence.

Restrict amount of deviation from diagonal (Sakoe-Chiba constraint)

This can be solved via dynamic programming.



Dynamic Time Warping Example

▶ Code

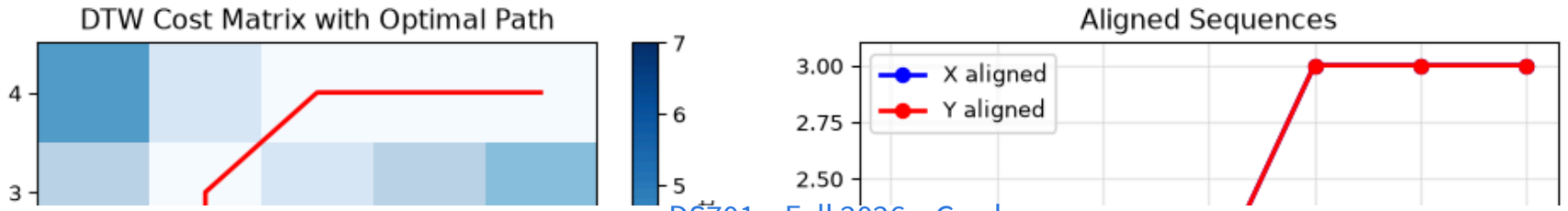
```
Original sequences:
X = [1, 1, 2, 2, 3]
Y = [1, 2, 3, 3, 3]

DTW distance = 0.0

Optimal alignment path (0-indexed): [(0, 0), (1, 0), (2, 1), (3, 1), (4, 2), (4, 3), (4, 4)]

Fully aligned sequences:
X_aligned:  1  1  2  2  3  3  3
Y_aligned:  1  1  2  2  3  3  3

Alignment mapping:
Step 0: X[0] = 1 ↔ Y[0] = 1
Step 1: X[1] = 1 ↔ Y[0] = 1
Step 2: X[2] = 2 ↔ Y[1] = 2
Step 3: X[3] = 2 ↔ Y[1] = 2
Step 4: X[4] = 3 ↔ Y[2] = 3
Step 5: X[4] = 3 ↔ Y[3] = 3
Step 6: X[4] = 3 ↔ Y[4] = 3
```



To explore further

See

T. Giorgino, “Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package”, Journal of Statistical Software, 2003. link

And python packages such as:

- dtw-python
- dtaidistance

🎯 Class Activity: DTW Concept Exploration

Task: Explore why DTW is better than Euclidean distance for time series with temporal shifts.

Given: Two simple time series:

- Series A: [1, 2, 3, 2, 1] (peak at position 3)
- Series B: [0, 1, 2, 3, 2, 1, 0] (peak at position 4)

Instructions:

1. **Calculate Euclidean distance** between the series (pad with zeros if needed)
2. **Sketch the DTW alignment** by hand:
    - Expand series A to better align with series B
3. **Calculate the Euclidean distance of the aligned series**
4. **Group discussion**:

From Time series to Strings

A closely related idea concerns strings.

The key point is that, like time series, strings are **sequences**.

Given two strings, one way to define a ‘distance’ between them is:

- the minimum number of **edit operations** that are needed to transform one string into the other.

Edit operations are insertion, deletion, and substitution of single characters.

This is called **edit distance** or **Levenshtein distance.**

Example

For example, given strings: `s = VIVALASVEGAS` and `t = VIVADAVIS`

we would like to

- compute the edit distance, and
- obtain the optimal **alignment**.

$$A(\mathbf{s},\mathbf{t}) = \begin{array}{cccccccccccc} \mathbf{V} & \mathbf{I} & \mathbf{V} & \mathbf{A} & \mathbf{L} & \mathbf{A} & \mathbf{S} & \mathbf{V} & \mathbf{E} & \mathbf{G} & \mathbf{A} & \mathbf{S} \\ | & | & | & | & & | & & | & & & & | \\ \mathbf{V} & \mathbf{I} & \mathbf{V} & \mathbf{A} & \mathbf{D} & \mathbf{A} & - & \mathbf{V} & - & - & \mathbf{I} & \mathbf{S} \end{array}$$

A dynamic programming algorithm can also be used to find this distance, and it is **very similar to dynamic time-warping.**

In bioinformatics this algorithm is called **"Smith-Waterman" sequence alignment.**

Recap

We covered the following topics

- reviewed representations of data,
- discussed metrics and norms,
- discussed similarity and dissimilarity functions,
- introduced time series,
- feature engineering, and
- dynamic time warping.