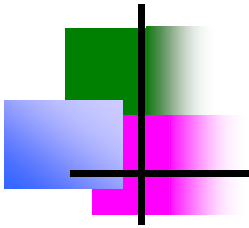


Machine Learning and Computational Intelligence
Lecture 4

Sanjeeb Prasad Panday, PhD
Associate Professor
Dept. of Electronics and Computer Engineering
Director (ICTC)
IOE, TU

Dimensionality Reduction

- Data representation

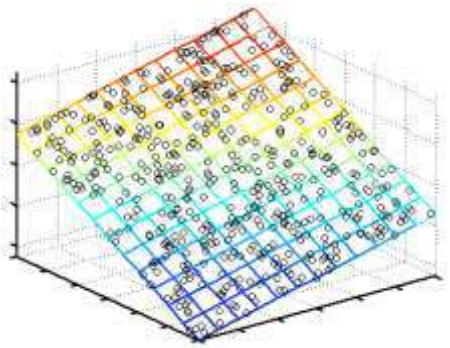
  Inputs are real-valued vectors in a high dimensional space.

- Linear structure

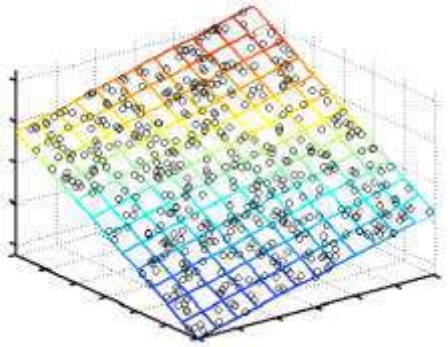
  Does the data live in a low dimensional subspace?

- Nonlinear structure

  Does the data live on a low dimensional submanifold?

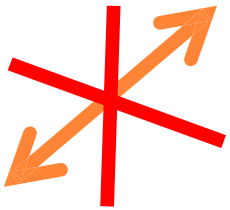


Dimensionality Reduction so far



PCA

Manifold learning methods



for non linear structure

Kernel PCA

but does not unfold the data

Notations

- Inputs (**high dimensional**)

  $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$ points in R^D

- Outputs **(low dimensional)**

  $\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_n}$ points in R^d ($d << D$)

Example

Assume the decision boundary is given by the quadratic function

$$f(\boldsymbol{x}) = a_0 + a_1 {x_1}^2 + a_2 {x_2}^2 + a_3 x_1 x_2 + a_4 x_1 + a_5 x_2 .$$

Obviously this is not a linear decision boundary.

By the following mapping, we get features that have a linear decision boundary:

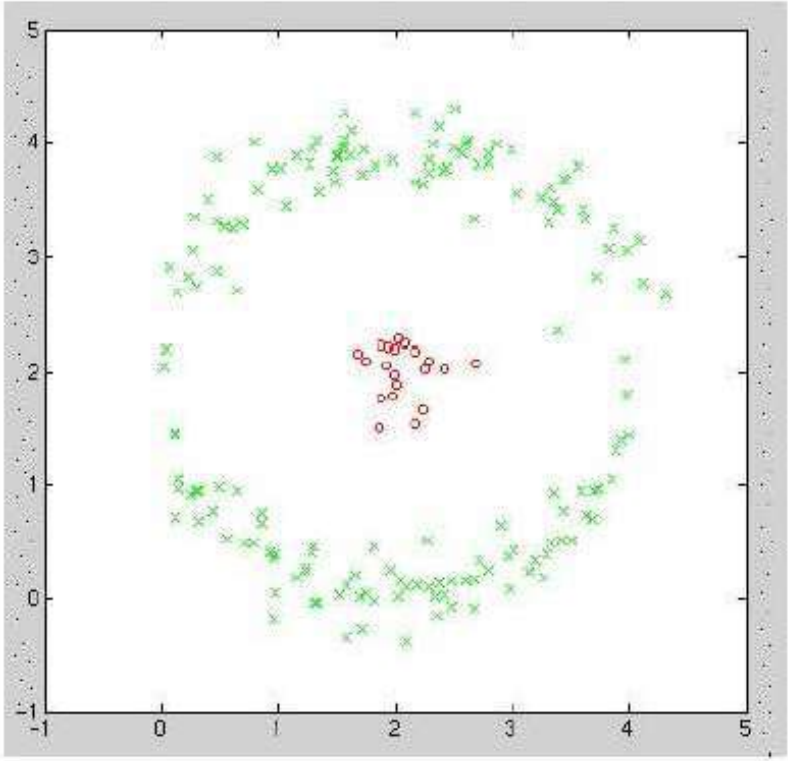
$$\phi(\boldsymbol{x}) = \begin{pmatrix} {x_1}^2 \\ {x_2}^2 \\ x_1 \cdot x_2 \\ x_1 \\ x_2 \end{pmatrix}$$

The “magic” of high dimensions

- Given some problem, how do we know what classes of functions are capable of solving that problem?

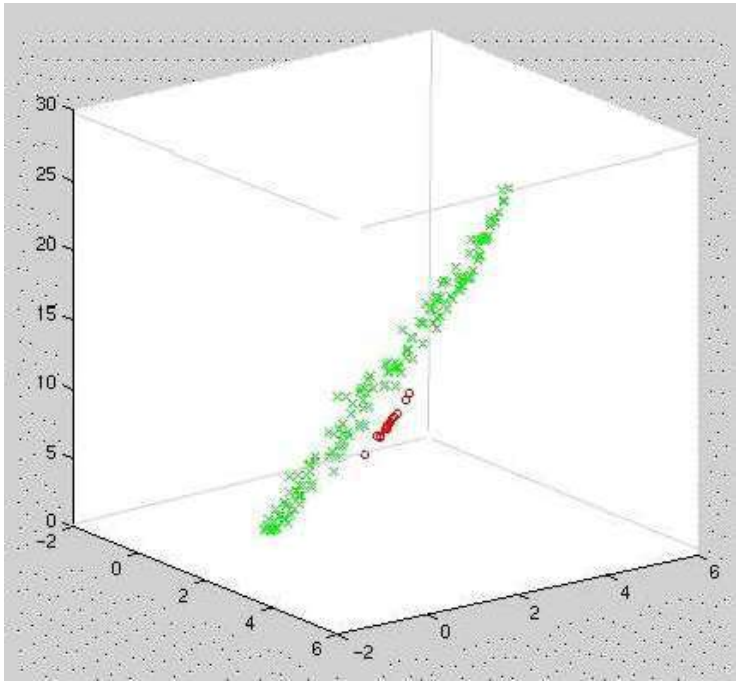
- VC (Vapnik-Chervonenkis) theory tells us that often mappings which take us into a higher dimensional space than the dimension of the input space provide us with greater classification power.

Example in $\mathbb{R}^2$



These classes are linearly inseparable in the input space.

Example: High-Dimensional Mapping



We can make the problem linearly separable by a simple mapping

$$\Phi : \mathbf{R}^2 \rightarrow \mathbf{R}^3$$

$$(x_1, x_2) \rightarrow (x_1, x_2, x_1^2 + x_2^2)$$

Kernel Trick

- High-dimensional mapping can seriously increase computation time.
- Can we get around this problem and still get the benefit of high-D?
- Yes! Kernel Trick

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

- Given *any* algorithm that can be expressed solely in terms of dot products, this trick allows us to construct different nonlinear versions of it.

Popular Kernels

Gaussian $K(\vec{x},\vec{x}') = \exp\left(-\beta \| \vec{x} - \vec{x}' \|^2\right)$

Polynomial $K(\vec{x},\vec{x}') = (1 + \vec{x} \cdot \vec{x}')^p$

Hyperbolic tangent $K(\vec{x},\vec{x}') = \tanh(\vec{x} \cdot \vec{x}' + \delta)$

Definition

A *kernel function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function that maps a pair of features to a real number. For a kernel function the following property holds:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$$

for any feature mapping ϕ.

Note:

Usually the evaluation of the kernel function is much easier than the computation of transformed features followed by the inner product.

Definition

For a given set of feature vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m$, we define the *kernel matrix*

$$\boldsymbol{K} \quad = \quad [K_{i,j}]_{i,j=1,2,\ldots,m}\,, \quad \text{where} \quad K_{i,j} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle.$$

Note:

The entries of the matrix are similarity measures for transformed feature pairs.

Kernel Principal Component Analysis (KPCA)

- Extends conventional principal component analysis (PCA) to a high dimensional feature space using the “kernel trick”.
- Can extract up to n (number of samples) nonlinear principal components without expensive computations.

Making PCA Non-Linear

- Suppose that instead of using the points x_i we would first map them to some nonlinear feature space $\phi(x_i)$

  E.g. using polar coordinates instead of cartesian coordinates would help us deal with the circle.

- Extract principal component in that space (PCA)
- The result will be non-linear in the original data space!

Derivation

- Suppose that the mean of the data in the feature space is

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\phi(x_i) = 0$$

- Covariance:

$$C = \frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T$$

- Eigenvectors

$$Cv = \lambda v$$

Derivation Cont.

- Eigenvectors can be expressed as linear combination of features:

$$v = \sum_{i=1}^{n} \alpha_i \phi(x_i)$$

- Proof:

$$Cv = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) \phi(x_i)^T v = \lambda v$$

thus

$$v = \frac{1}{\lambda n} \sum_{i=1}^{n} \phi(x_i) \phi(x_i)^T v = \frac{1}{\lambda n} \sum_{i=1}^{n} (\phi(x_i) \cdot v) \phi(x_i)^T$$

Showing that $xx^T v = (x \cdot v)x^T$

$$
\begin{aligned}
(\boldsymbol{x}\boldsymbol{x}^T)\boldsymbol{v} &= \begin{pmatrix} x_1x_1 & x_1x_2 & \dots & x_1x_M \\ x_2x_1 & x_2x_2 & \dots & x_2x_M \\ \vdots & \vdots & \ddots & \vdots \\ x_Mx_1 & x_Mx_2 & \dots & x_Mx_M \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_M \end{pmatrix} \\
&= \begin{pmatrix} x_1x_1v_1 + x_1x_2v_2 + \ldots + x_1x_Mv_M \\ x_2x_1v_1 + x_2x_2v_2 + \ldots + x_2x_Mv_M \\ \vdots \\ x_Mx_1v_1 + x_Mx_2v_2 + \ldots + x_Mx_Mv_M \end{pmatrix}
\end{aligned}
$$

Showing that $xx^T v = (x \cdot v)x^T$

$$= \begin{pmatrix} (x_1v_1 + x_2v_2 + \ldots + x_Mv_M)\, x_1 \\ (x_1v_1 + x_2v_2 + \ldots + x_Mv_M)\, x_2 \\ \vdots \\ (x_1v_1 + x_2v_2 + \ldots + x_Mv_M)\, x_M \end{pmatrix}$$

$$= \begin{pmatrix} x_1v_1 + x_2v_2 + \ldots + x_Mv_M \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix}$$

$$= (\boldsymbol{x} \cdot \boldsymbol{v})\boldsymbol{x} \qquad \square$$

Derivation Cont.

- So, from before we had,

$$v = \frac{1}{n\lambda}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T v = \frac{1}{n\lambda}\sum_{i=1}^{n}(\phi(x_i)\cdot v)\phi(x_i)^T$$

just a scalar

- this means that all solutions v with $= 0$ lie in the span of $\phi(x_1),..,\phi(x_n)$, i.e.,

$$v = \sum_{i=1}^{n}\alpha_i\phi(x_i)$$

- Finding the eigenvectors is equivalent to finding the coefficients α_i

Derivation Cont.

- By substituting this back into the equation we get:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T\left(\sum_{l=1}^{n}\alpha_{jl}\phi(x_l)\right)=\lambda_j\sum_{l=1}^{n}\alpha_{jl}\phi(x_l)$$

- We can rewrite it as

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\left(\sum_{l=1}^{n}\alpha_{jl}K(x_i,x_l)\right)=\lambda_j\sum_{l=1}^{n}\alpha_{jl}\phi(x_l)$$

- Multiply this by $\phi(x_k)$ from the left:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x_k)^T\phi(x_i)\left(\sum_{l=1}^{n}\alpha_{jl}K(x_i,x_l)\right)=\lambda_j\sum_{l=1}^{n}\alpha_{jl}\phi(x_k)^T\phi(x_l)$$

Derivation Cont.

- By plugging in the kernel and rearranging we get:

$$\mathrm{K}^2\alpha_j = n\lambda_j \mathrm{K}\alpha_j$$

We can remove a factor of K from both sides of the matrix (this will only affects the eigenvectors with zero eigenvalue, which will not be a principle component anyway):

$$\mathrm{K}\alpha_j = n\lambda_j\alpha_j$$

- We have a normalization condition for the α_j vectors:

$$v_j^T v_j = 1 \Rightarrow \sum_{k=1}^{n}\sum_{l=1}^{n}\alpha_{jl}\alpha_{jk}\phi(x_l)^T\phi(x_k) = 1 \Rightarrow \alpha_j^T \mathrm{K}\alpha_j = 1$$

Derivation Cont.

- By multiplying $\mathrm{K}\alpha_j = n\lambda_j\alpha_j$ by α_j and using the normalization condition we get:

$$\lambda_j n\alpha_j^T\alpha_j = 1, \quad \forall j$$

- For a new point x, its projection onto the principal components is:

$$\phi(x)^T v_j = \sum_{i=1}^{n} \alpha_{ji}\phi(x)^T\phi(x_i) = \sum_{i=1}^{n} \alpha_{ji}K(x, x_i)$$

Normalizing the feature space

- In general, $\phi(x_i)$ may not be zero mean.
- Centered features:

$$\tilde{\phi}(x_k) = \phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)$$

- The corresponding kernel is:

$$\tilde{K}(x_i, x_j) = \tilde{\phi}(x_i)^T \tilde{\phi}(x_j)$$

$$= \left(\phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)\right)^T \left(\phi(x_j) - \frac{1}{n}\sum_{k=1}^{n}\phi(x_k)\right)$$

$$= K(x_i, x_j) - \frac{1}{n}\sum_{k=1}^{n}K(x_i, x_k) - \frac{1}{n}\sum_{k=1}^{n}K(x_j, x_k) + \frac{1}{n^2}\sum_{l,k=1}^{n}K(x_l, x_k)$$

Normalizing the feature space (cont)

$$\tilde{K}(x_i, x_j) = K(x_i, x_j) - \frac{1}{n}\sum_{k=1}^{n} K(x_i, x_k) - \frac{1}{n}\sum_{k=1}^{n} K(x_j, x_k) + \frac{1}{n^2}\sum_{l,k=1}^{n} K(x_l, x_k)$$

- In a matrix form

$$\tilde{\mathrm{K}} = \mathrm{K} - 2\mathbf{1}_{1/n}\mathrm{K} + \mathbf{1}_{1/n}\mathrm{K}\mathbf{1}_{1/n}$$

- where $\mathbf{1}_{1/n}$ is a matrix with all elements 1/n.

Summary of kernel PCA

- Pick a kernel
- Construct the normalized kernel of the matrix data (dimension m x m):

$$\tilde{\mathrm{K}} = \mathrm{K} - 2\mathbf{1}_{1/\mathrm{n}}\mathrm{K} + \mathbf{1}_{1/\mathrm{n}}$$

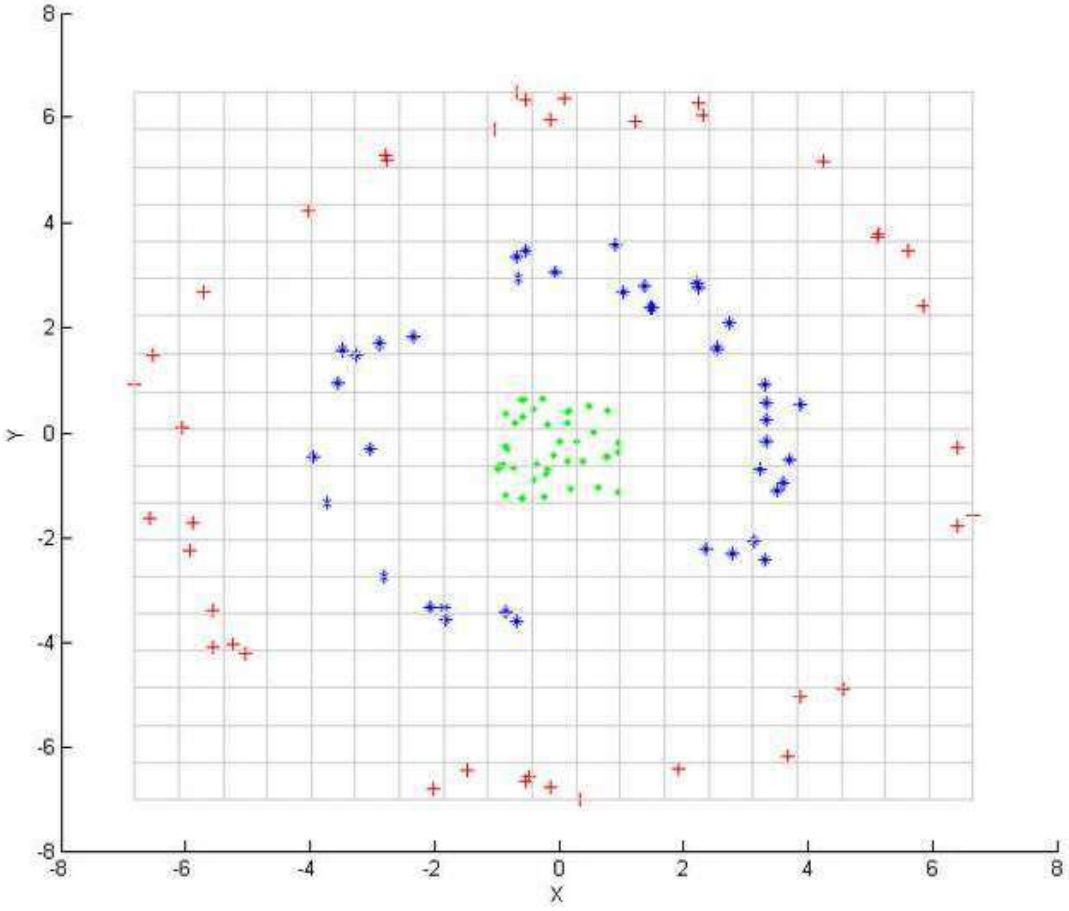
- Solve an eigenvalue problem:

$$\tilde{\mathrm{K}}\alpha_i = \lambda_i \alpha_i$$

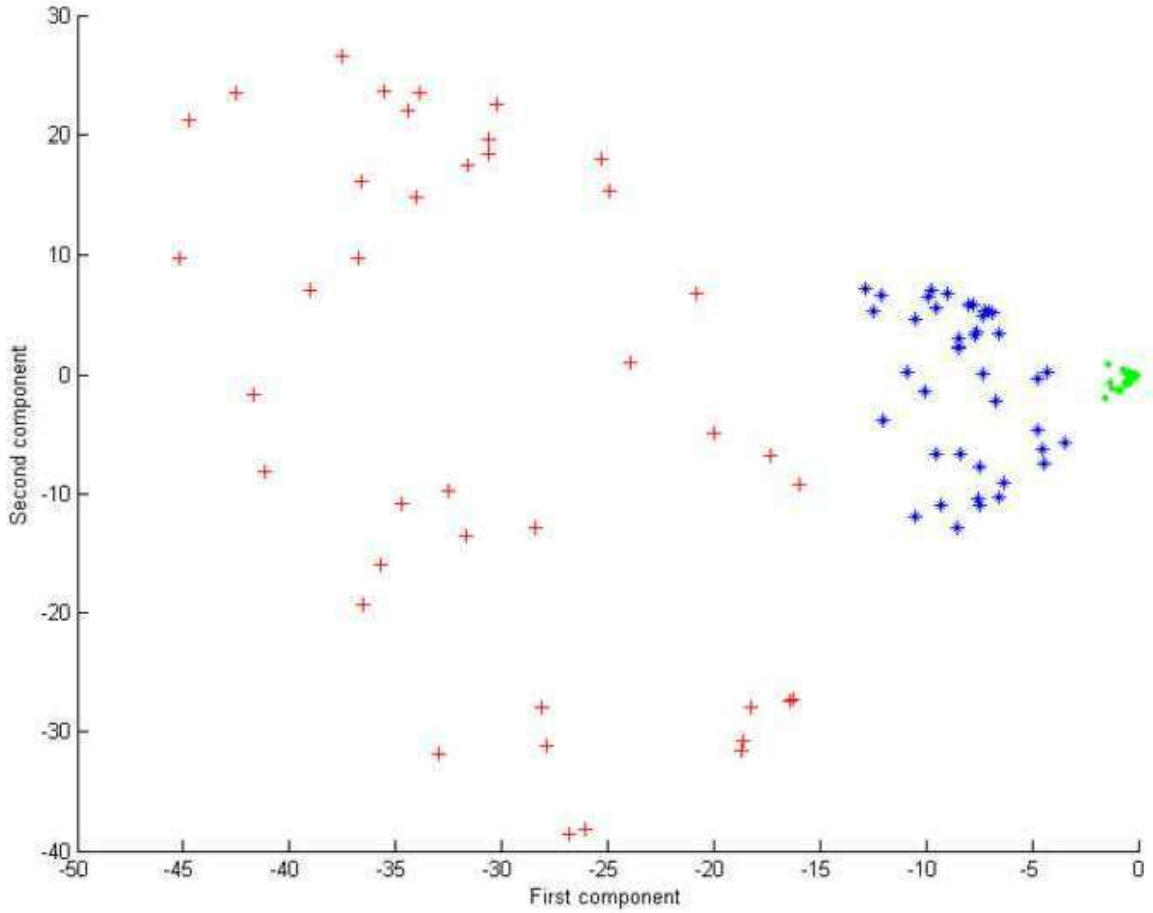
- For any data point (new or old), we can represent it as

$$y_j = \sum_{i=1}^{n} \alpha_{ji} K(x, x_i), \quad \mathrm{j} = 1,..,d$$

Example: Input Points



Example: KPCA



Example: De-noising images

Original data

Data corrupted with Gaussian noise

Result after linear PCA

Result after kernel PCA, Gaussian kernel

Properties of KPCA

- Kernel PCA can give a good re-encoding of the data when it lies along a non-linear manifold.
- The kernel matrix is n x n, so kernel PCA will have difficulties if we have lots of data points.

What is a Confusion Matrix?

A common method for describing the performance of a classification model consisting of true positives, true negatives, false positives, and false negatives.

It is called a confusion matrix because it shows how confused the model is between the classes.

Confusion Matrix

Type I error
(false positive)

Type II error
(false negative)

Confusion Matrix

- A confusion matrix for a two classes (+, -) is shown below.

	C1	C2
C1	True positive	False negative
C2	False positive	True negative

	+	-
+	++	+-
-	-+	--

- There are four quadrants in the confusion matrix, which are symbolized as below.
  - True Positive (TP: f_{++}) : The number of instances that were positive (+) and correctly classified as positive (+v).
  - False Negative (FN: f_{+-}): The number of instances that were positive (+) and incorrectly classified as negative (-). It is also known as **Type 2 Error**.
  - False Positive (FP: f_{-+}): The number of instances that were negative (-) and incorrectly classified as (+). This also known as **Type 1 Error**.
  - True Negative (TN: f_{--}): The number of instances that were negative (-) and correctly classified as (-).

Confusion Matrix

Note:

- N_p = TP (f_{++}) + FN (f_{+-})

  = is the total number of positive instances.

- N_n = FP(f_{-+}) + TN(f_{--})

  = is the total number of negative instances.

- $N = N_p + N_n$

  = is the total number of instances.

- (TP + TN) denotes the number of correct classification

- (FP + FN) denotes the number of errors in classification.

- For a perfect classifier FP = FN = 0, that is, there would be no Type 1 or Type 2 errors.

Confusion Matrix

Example 11.4: Confusion matrix

A classifier is built on a dataset regarding Good and Worst classes of stock markets. The model is then tested with a test set of 10000 unseen instances. The result is shown in the form of a confusion matrix. The result is self explanatory.

Class	Good	Worst	Total	Rate(%)
Good	6954	46	7000	99.34
Worst	412	2588	3000	86.27
Total	**7366**	**2634**	**10000**	**95.52**

Predictive accuracy?

Confusion Matrix for Multiclass Classifier

- Having m classes, confusion matrix is a table of size $m \times m$, where, element at (i, j) indicates the number of instances of class i but classified as class j.

- To have good accuracy for a classifier, ideally most diagonal entries should have large values with the rest of entries being close to zero.

- Confusion matrix may have additional rows or columns to provide total or recognition rates per class.

Confusion Matrix for Multiclass Classifier

Example 11.5: Confusion matrix with multiple class

Following table shows the confusion matrix of a classification problem with six classes labeled as C_1, C_2, C_3, C_4, C_5 and C_6.

Class	C1	C2	C3	C4	C5	C_6
C1	52	10	7	0	0	1
C2	15	50	6	2	1	2
C3	5	6	6	0	0	0
C4	0	2	0	10	0	1
C5	0	1	0	0	7	1
C_6	1	3	0	1	0	24

Predictive accuracy?

Confusion Matrix for Multiclass Classifier

- In case of multiclass classification, sometimes one class is important enough to be regarded as positive with all other classes combined together as negative.

- Thus a large confusion matrix of m*m can be concised into 2*2 matrix.

Example 11.6: *m*×*m* CM to 2×2 CM

- For example, the CM shown in Example 11.5 is transformed into a CM of size 2×2 considering the class C_1 as the positive class and classes C_2, C_3, C_4, C_5 and C_6 combined together as negative.

Class	+	-
+	52	18
-	21	123

How we can calculate the predictive accuracy of the classifier model in this case?

Are the predictive accuracy same in both Example 11.5 and Example 11.6?

Performance Evaluation Metrics

- We now define a number of metrics for the measurement of a classifier.
  - In our discussion, we shall make the assumptions that there are only two classes: + (positive) and – (negative)
  - Nevertheless, the metrics can easily be extended to multi-class classifiers (with some modifications)

- **True Positive Rate (TPR):** It is defined as the fraction of the positive examples predicted correctly by the classifier.

$$TPR = \frac{TP}{P} = \frac{TP}{TP+FN} = \frac{f_{++}}{f_{++}+f_{+-}}$$

  - This metrics is also known as ***Recall***, ***Sensitivity*** or ***Hit rate***.

- **False Positive Rate (FPR):** It is defined as the fraction of negative examples classified as positive class by the classifier.

$$FPR = \frac{FP}{N} = \frac{FP}{FP+TN} = \frac{f_{-+}}{f_{-+}+f_{--}}$$

  - This metric is also known as ***False Alarm Rate***.

Performance Evaluation Metrics

- **False Negative Rate (FNR):** It is defined as the fraction of positive examples classified as a negative class by the classifier.

$$FNR = \frac{FN}{P} = \frac{FN}{TP + FN} = \frac{f_{+-}}{f_{++} + f_{+-}}$$

- **True Negative Rate (TNR):** It is defined as the fraction of negative examples classified correctly by the classifier

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP} = \frac{f_{--}}{f_{--} + f_{-+}}$$

  - This metric is also known as ***Specificity***.

Performance Evaluation Metrics

- **Positive Predictive Value (PPV):** It is defined as the fraction of the positive examples classified as positive that are really positive

$$PPV = \frac{TP}{TP + FP} = \frac{f_{++}}{f_{++} + f_{-+}}$$

  - It is also known as *Precision*.

- **F_1 Score (F_1):** Recall (r) and Precision (p) are two widely used metrics employed in analysis, where detection of one of the classes is considered more significant than the others.
  - It is defined in terms of (r or TPR) and (p or PPV) as follows.

$$F_1 = \frac{2r.p}{r + p} = \frac{2TP}{2TP + FP + FN}$$

$$= \frac{2f_{++}}{2f_{++} + f_{+-} + f_{-+}} = \frac{2}{\frac{1}{r} + \frac{1}{p}}$$

Note

- F_1 represents the harmonic mean between recall and precision
- High value of F_1 score ensures that both Precision and Recall are reasonably high.

Performance Evaluation Metrics

- More generally, F_β score can be used to determine the trade-off between Recall and Precision as

$$F_\beta = \frac{(\beta + 1)rp}{r + \beta p} = \frac{(\beta + 1)TP}{(\beta + 1)TP + \beta FN + FP}$$

- Both, Precision and Recall are special cases of F_β when $\beta = 0$ and $\beta = 1$, respectively.

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

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

Performance Evaluation Metrics

- A more general metric that captures Recall, Precision as well as is defined in the following.

$$F_\omega = \frac{\omega_1 TP + \omega_4 TN}{\omega_1 TP + \omega_2 FP + \omega_3 FN + \omega_4 TN}$$

Metric	ω_1	ω_2	ω_3	ω_4
Recall	1	1	0	1
Precision	1	0	1	0
F_β	β+1	β	1	0

Note

- In fact, given *TPR*, *FPR*, *p* and *r*, we can derive all others measures.
- That is, these are the universal metrics.

Predictive Accuracy (ε)

- It is defined as the fraction of the number of examples that are correctly classified by the classifier to the total number of instances.

$$\varepsilon = \frac{TP + TN}{P + N}$$

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

$$= \frac{f_{++} + f_{--}}{f_{++} + f_{+-} + f_{\mp} + f_{--}}$$

- This accuracy is equivalent to F_w with $w_1 = w_2 = w_3 = w_4 = 1$.

Error Rate ($\bar{\varepsilon}$)

- The error rate $\bar{\varepsilon}$ is defined as the fraction of the examples that are incorrectly classified.

$$\bar{\varepsilon} = \frac{FP + FN}{P + N}$$

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

$$= \frac{f_{+-} + f_{-+}}{f_{++} + f_{+-} + f_{-+} + f_{--}}$$

Note

$\bar{\varepsilon} = 1 - \varepsilon$.

Accuracy, Sensitivity and Specificity

- Predictive accuracy (ε) can be expressed in terms of sensitivity and specificity.
- We can write

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

$$= \frac{TP + TN}{P + N}$$

$$\varepsilon = \frac{TP}{P} \times \frac{P}{P + N} + \frac{TN}{N} \times \frac{N}{P + N}$$

Thus,

$$\varepsilon = \text{Sensitivity} \times \frac{P}{P+N} + \text{Specificity} \times \frac{N}{P+N}$$

Analysis with Performance Measurement Metrics

- Based on the various performance metrics, we can characterize a classifier.
- We do it in terms of TPR, FPR, Precision and Recall and Accuracy
- **Case 1: Perfect Classifier**

When every instance is correctly classified, it is called the perfect classifier. In this case, $TP = P$, $TN = N$ and CM is

$$TPR = \frac{P}{P} = 1$$

$$FPR = \frac{0}{N} = 0$$

$$Precision = \frac{P}{P} = 1$$

$$F_1\ Score = \frac{2 \times 1}{1+1} = 1$$

$$Accuracy = \frac{P+N}{P+N} = 1$$

		Predicted Class	
		+	-
Actual class	+	P	0
	-	0	N

Analysis with Performance Measurement Metrics

- **Case 2: Worst Classifier**

When every instance is wrongly classified, it is called the worst classifier. In this case, $TP = 0$, $TN = 0$ and the CM is

$$TPR = \frac{0}{P} = 0$$

$$FPR = \frac{N}{N} = 1$$

$$Precision = \frac{0}{N} = 0$$

F_1 *Score* = Not applicable as *Recall* + *Precision* = 0

$$\text{Accuracy} = \frac{0}{P+N} = 0$$

		Predicted Class	
		+	-
Actual class	+	0	P
	-	N	0

Analysis with Performance Measurement Metrics

- **Case 3: Ultra-Liberal Classifier**

The classifier always predicts the + class correctly. Here, the False Negative (FN) and True Negative (TN) are zero. The CM is

		Predicted Class	
		+	-
Actual class	+	P	0
	-	N	0

$$TPR = \frac{P}{P} = 1$$

$$FPR = \frac{N}{N} = 1$$

$$Precision = \frac{P}{P+N}$$

$$F_1\ Score = \frac{2P}{2P+N}$$

$$\text{Accuracy} = \frac{P}{P+N} = 0$$

Analysis with Performance Measurement Metrics

- **Case 4: Ultra-Conservative Classifier**

This classifier always predicts the - class correctly. Here, the True Positive (TP) and False Positive (FP) are zero. The CM is

$$TPR = \frac{0}{P} = 0$$

$$FPR = \frac{0}{N} = 0$$

$$Precision = \text{Not applicable}$$

$$(\text{as } TP + FP = 0)$$

$$F_1\ Score = \text{Not applicable}$$

$$\text{Accuracy} = \frac{N}{P+N} = 0$$

		Predicted Class	
		+	-
Actual class	+	0	p
	-	0	N

Predictive Accuracy versus TPR and FPR

- One strength of characterizing a classifier by its *TPR* and *FPR* is that they do not depend on the relative size of *P* and *N*.
  - The same is also applicable for *FNR* and *TNR* and others measures from CM.
- In contrast, the *Predictive Accuracy*, *Precision*, *Error Rate*, F_1 *Score*, etc. are affected by the relative size of *P* and *N*.
- *FPR*, *TPR*, *FNR* and *TNR* are calculated from the different rows of the CM.
  - On the other hand Predictive Accuracy, etc. are derived from the values in both rows.
- This suggests that *FPR*, *TPR*, *FNR* and TNR are more effective than *Predictive Accuracy*, etc.

True Positives

		Predicted class		
		Apple	Orange	Pear
Actual class	Apple	50	5	50
	Orange	10	50	20
	Pear	5	5	0

The model correctly classified 50 apples and 50 oranges.

True Negatives for Apple

		Predicted class		
		Apple	Orange	Pear
Actual class	Apple	50	5	50
	Orange	10	50	20
	Pear	5	5	0

The model correctly classified 75 cases as not belonging to class apple.

True Negatives for Orange

		Predicted class		
		Apple	Orange	Pear
Actual class	Apple	50	5	50
	Orange	10	50	20
	Pear	5	5	0

The model correctly classified 105 cases as not belonging to class orange.

True Negatives for Pear

		Predicted class		
		Apple	Orange	Pear
Actual class	Apple	50	5	50
	Orange	10	50	20
	Pear	5	5	0

The model correctly classified 115 cases as not belonging to class pear.

False Positives for Apple

Actual class	Predicted class: Apple	Orange	Pear
Apple	50	5	50
Orange	10	50	20
Pear	5	5	0

The model incorrectly classified 15 cases as apples.

False Positives for Orange

		Predicted class		
		Apple	Orange	Pear
Actual class	Apple	50	5	50
	Orange	10	50	20
	Pear	5	5	0

The model incorrectly classified 10 cases as oranges.

False Positives for Pear

		Predicted class		
		Apple	Orange	Pear
Actual class	Apple	50	5	50
	Orange	10	50	20
	Pear	5	5	0

The model incorrectly classified 70 cases as pears.

False Negatives for Apple

		Predicted class		
		Apple	Orange	Pear
Actual class	Apple	50	5	50
	Orange	10	50	20
	Pear	5	5	0

The model incorrectly classified 55 cases as not belonging to class apple.

False Negatives for Orange

		Predicted class		
		Apple	Orange	Pear
Actual class	Apple	50	5	50
	Orange	10	50	20
	Pear	5	5	0

The model incorrectly classified 30 cases as not belonging to class orange.

False Negatives for Pear

		Predicted class		
		Apple	Orange	Pear
Actual class	Apple	50	5	50
	Orange	10	50	20
	Pear	5	5	0

The model incorrectly classified 10 cases as not belonging to class pears.

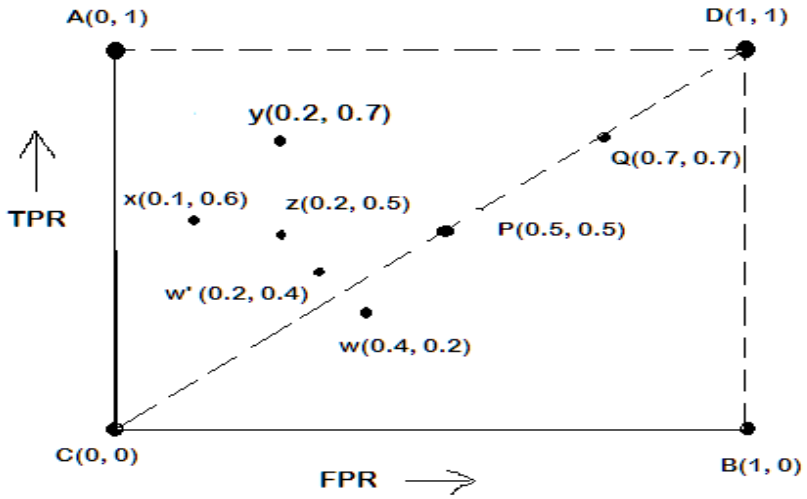
ROC Curves

ROC Curves

- ROC is an abbreviation of Receiver Operating Characteristic come from the signal detection theory, developed during World War 2 for analysis of radar images.
- In the context of classifier, ROC plot is a useful tool to study the behaviour of a classifier or comparing two or more classifiers.
- A ROC plot is a two-dimensional graph, where, X-axis represents FP rate (FPR) and Y-axis represents TP rate (TPR).
- Since, the values of FPR and TPR varies from 0 to 1 both inclusive, the two axes thus from 0 to 1 only.
- Each point *(x, y)* on the plot indicating that the FPR has value *x* and the TPR value *y*.

ROC Plot

- A typical look of ROC plot with few points in it is shown in the following figure.

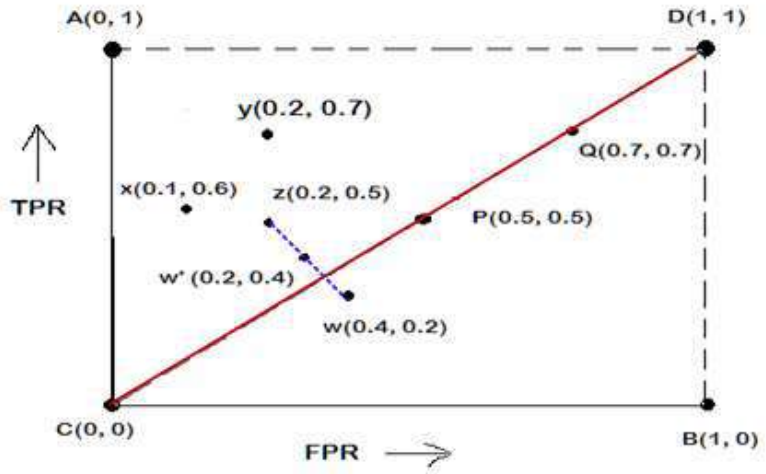


- Note the four cornered points are the four extreme cases of classifiers

Identify the four extreme classifiers.

Interpretation of Different Points in ROC Plot

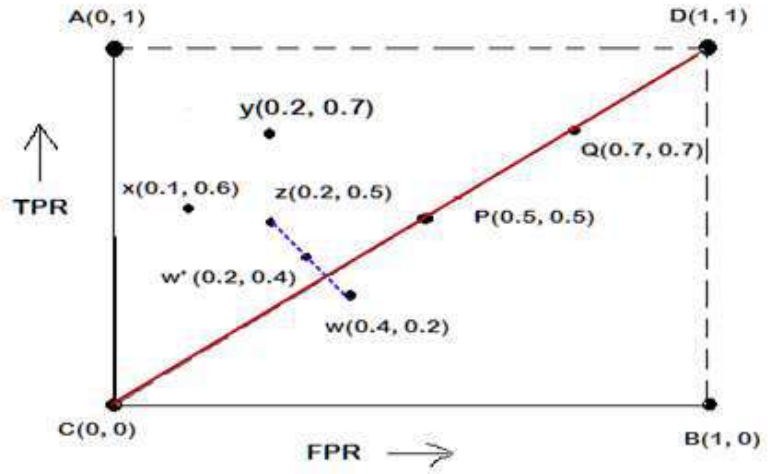
- Le us interpret the different points in the ROC plot.



- The four points (A, B, C, and D)
  - A: TPR = 1, FPR = 0, the ideal model, i.e., the perfect classifier, no false results
  - B: TPR = 0, FPR = 1, the worst classifier, not able to predict a single instance
  - C: TPR = 0, FPR = 0, the model predicts every instance to be a Negative class, i.e., it is an ultra-conservative classifier
  - D: TPR = 1, FPR = 1, the model predicts every instance to be a Positive class, i.e., it is an ultra-liberal classifier

Interpretation of Different Points in ROC Plot

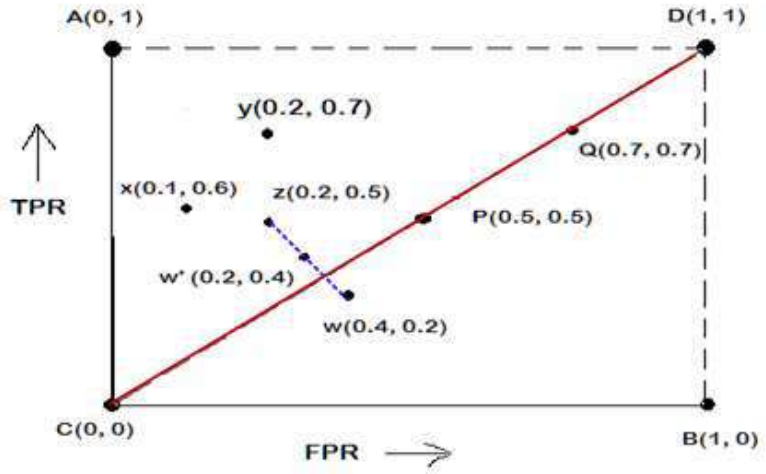
- Le us interpret the different points in the ROC plot.



- The points on diagonals
  - The diagonal line joining point C(0,0) and D(1,1) corresponds to random guessing
    - Random guessing means that a record is classified as positive (0r negative) with a certain probability
    - Suppose, a test set contacting N_+ positive and N_- negative instances. Suppose, the classifier guesses any instances with probability p
    - Thus, the random classifier is expected to correctly classify $p.N_+$ of the positive instances and $p.N_-$ of the negative instances
    - Hence, $TPR = FPR = p$
    - Since $TPR = FPR$, the random classifier results reside on the main diagonals

Interpretation of Different Points in ROC Plot

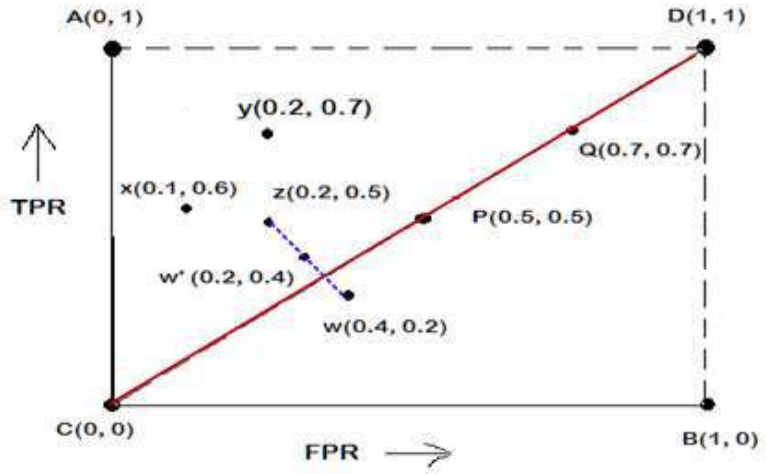
- Let us interpret the different points in the ROC plot.



- The points on the upper diagonal region
  - All points, which reside on upper-diagonal region are corresponding to classifiers "good" as their TPR is as good as FPR (i.e., FPRs are lower than TPRs)
  - Here, X is better than Z as X has higher TPR and lower FPR than Z.
  - If we compare X and Y, neither classifier is superior to the other

Interpretation of Different Points in ROC Plot

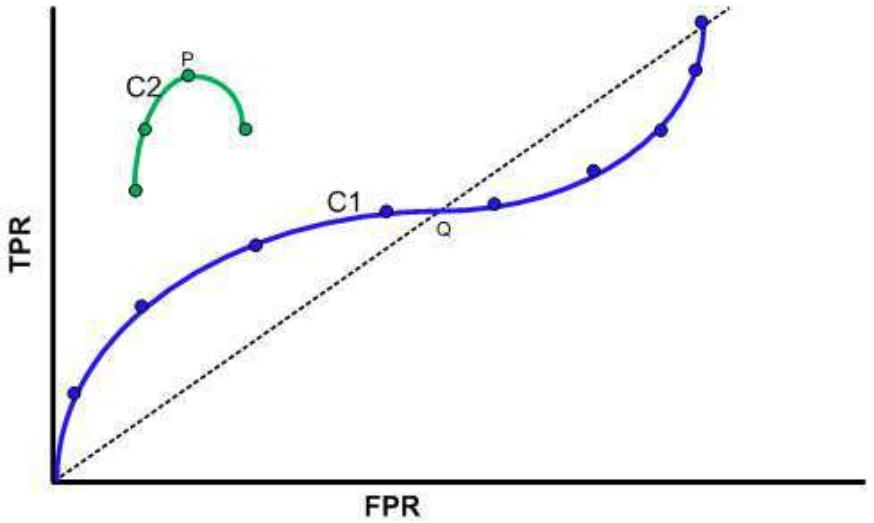
- Let us interpret the different points in the ROC plot.



- The points on the lower diagonal region
  - The Lower-diagonal triangle corresponds to the classifiers that are worst than random classifiers
  - Note: A classifier that is worst than random guessing, simply by reversing its prediction, we can get good results.
    - W'(0.2, 0.4) is the better version than W(0.4, 0.2), W' is a mirror reflection of W

Tuning a Classifier through ROC Plot

- Using ROC plot, we can compare two or more classifiers by their TPR and FPR values and this plot also depicts the trade-off between TPR and FPR of a classifier.

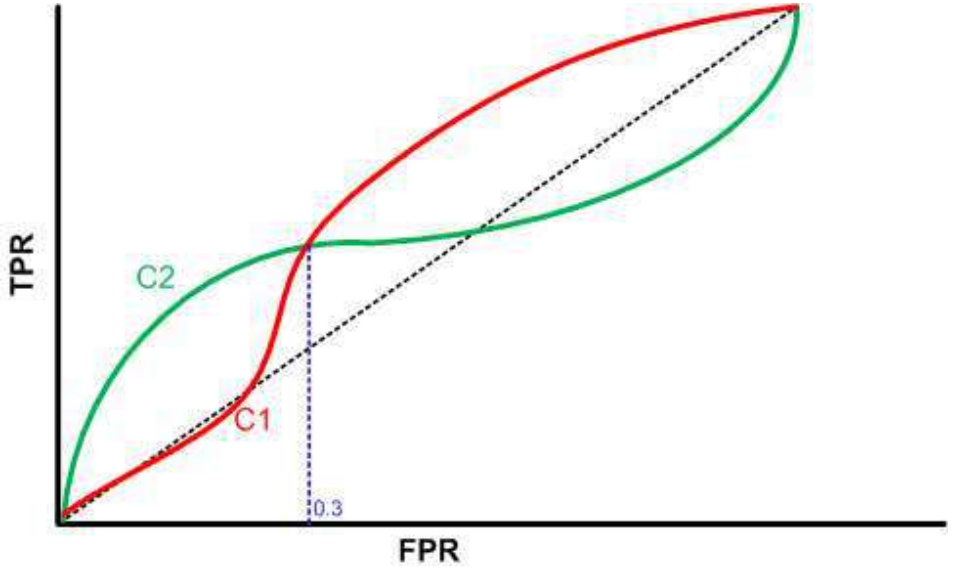


- Examining ROC curves can give insights into the best way of tuning parameters of classifier.

- For example, in the curve C2, the result is degraded after the point P. Similarly for the observation C1, beyond Q the settings are not acceptable.

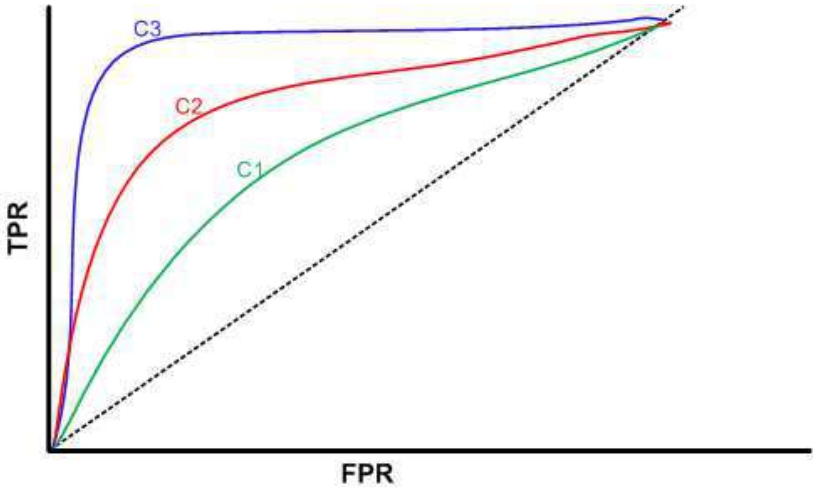
Comparing Classifiers trough ROC Plot

- Two curves C1 and C2 are corresponding to the experiments to choose two classifiers with their parameters.
- Here, C1 is better than C2 when FPR is greater than 0.3.
- However, C2 is better, when FPR is less than 0.3.
- Clearly, neither of these two classifiers dominates the other.



Comparing Classifiers trough ROC Plot

- We can use the concept of "area under curve" (AUC) as a better method to compare two or more classifiers.
- If a model is perfect, then its AUC = 1.
- If a model simply performs random guessing, then its AUC = 0.5
- A model that is strictly better than other, would have a larger value of AUC than the other.

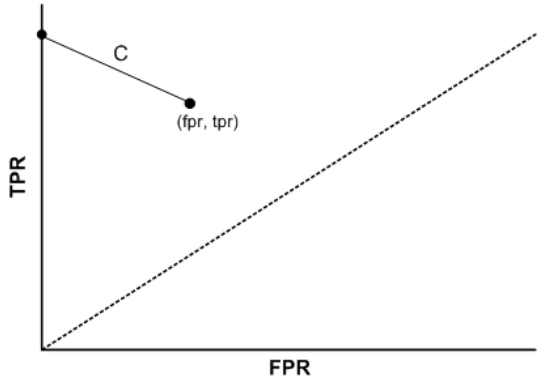


- Here, C3 is best, and C2 is better than C1 as AUC(C3)>AUC(C2)>AUC(C1).

A Quantitative Measure of a Classifier

- The concept of ROC plot can be extended to compare quantitatively using Euclidean distance measure.

- See the following figure for an explanation.



- Here, *C(fpr, tpr)* is a classifier and $\boldsymbol{\delta}$ denotes the Euclidean distance between the best classifier (0, 1) and C. That is,

  - $\boldsymbol{\delta} = \sqrt{fpr^2 + (1 - tpr)^2}$

A Quantitative Measure of a Classifier

- The smallest possible value of $\boldsymbol{\delta}$ is 0
- The largest possible values of $\boldsymbol{\delta}$ is $\sqrt{2}$ (when (fpr = 1 and tpr = 0).
- We could hypothesise that the smaller the value of $\boldsymbol{\delta}$, the better the classifier.
- $\boldsymbol{\delta}$ is a useful measure, but does not take into account the relative importance of true and false positive rates.
- We can specify the relative importance of making TPR as close to 1 and FPR as close 0 by a weight *w* between 0 to 1.
- We can define weighted $\boldsymbol{\delta}$ (denoted by $\boldsymbol{\delta}_w$) as

$$\boldsymbol{\delta}_w = \sqrt{(1-w)fpr^2 + w(1-tpr)^2}$$

Note

- If $w = 0$, it reduces to $\boldsymbol{\delta}_w$= fpr, i.e., FP Rate.
- If $w = 1$, it reduces to $\boldsymbol{\delta}_w$ = 1 – tpr, i.e., we are only interested to maximizing TP Rate.

Reference

- The detail material related to this lecture can be found in

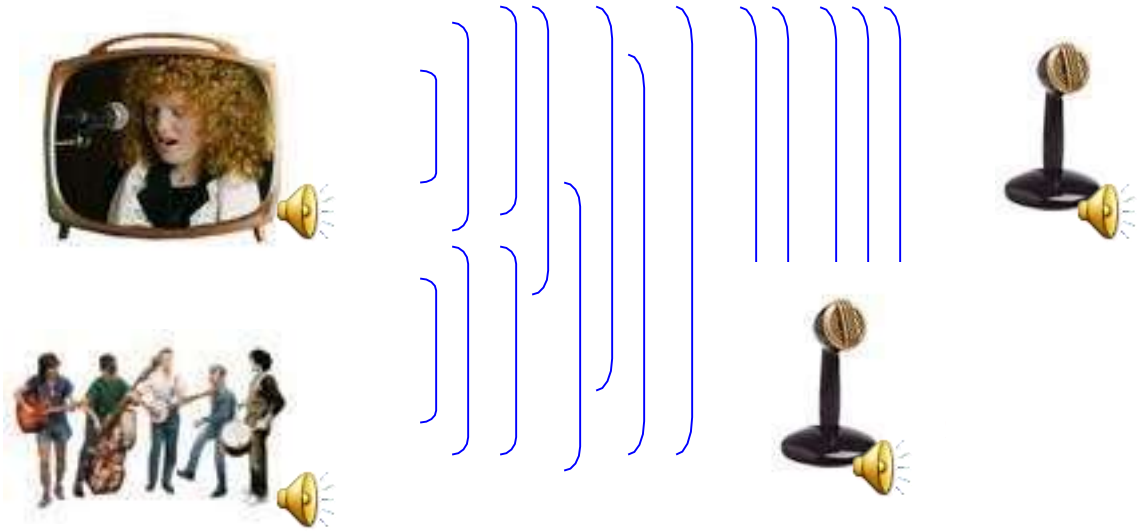
Data Mining: Concepts and Techniques, (3rd Edn.), Jiawei Han, Micheline Kamber, Morgan Kaufmann, 2015.

Introduction to Data Mining, Pang-Ning Tan, Michael Steinbach, and Vipin Kumar, Addison-Wesley, 2014

Independent Component Analysis

Motivation - Cocktail-Party Problem

- Simple scenario:
  - Two people speaking simultaneously in a room.
  - Speeches are recorded by two microphones in separate locations.



Motivation - Cocktail-Party Problem

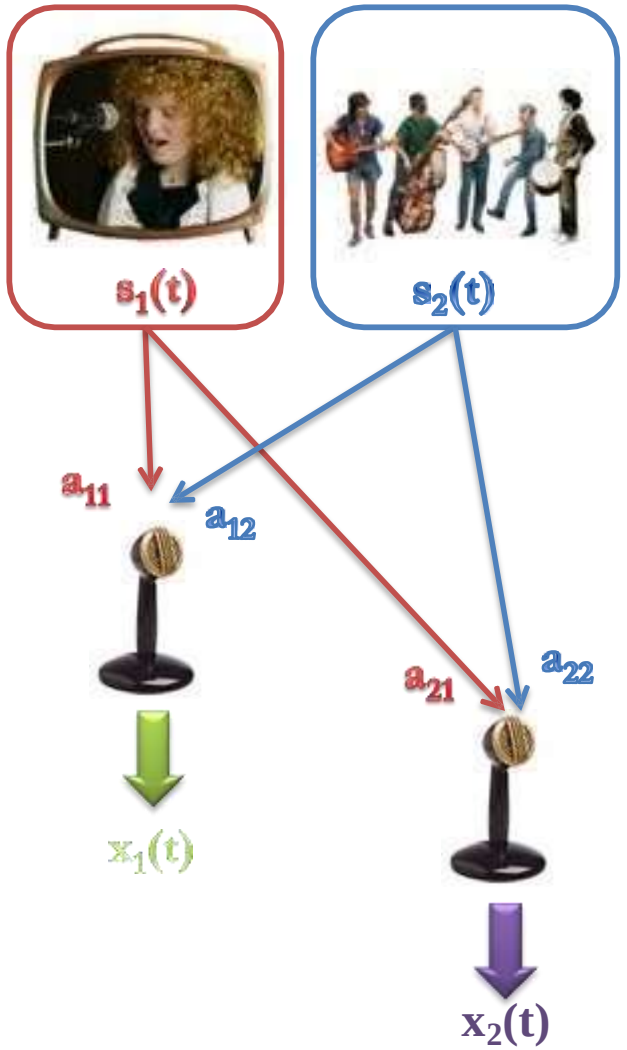
- Let $\mathbf{s_1(t)}$, $\mathbf{s_2(t)}$ be the speech signals emitted by the two speakers.
- Recorded time signals, by the two microphones, are denoted by $\mathbf{x_1(t)}$, $\mathbf{x_2(t)}$.
- The recorded time signals can be expressed as a linear equation:

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t)$$

$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t)$$

  where parameters in matrix **A** depend on distances of the microphones to the speaker, along with other microphone properties

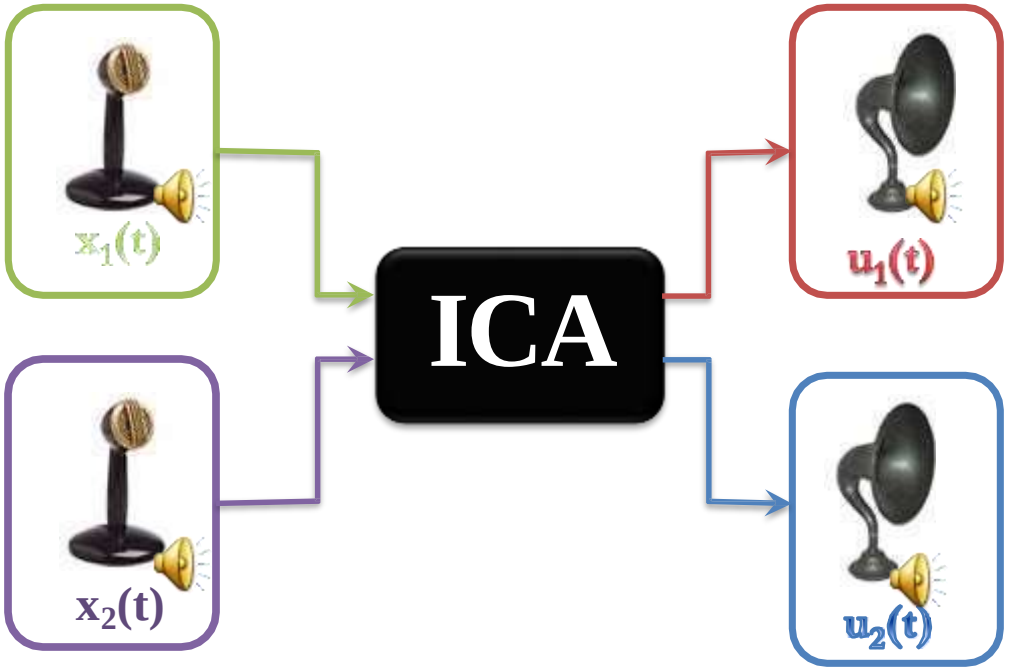
- Assume $\mathbf{s_1(t)}$ and $\mathbf{s_2(t)}$ are *statistically independent.*



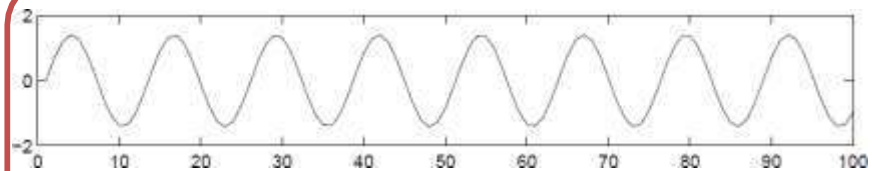
Motivation - Cocktail-Party Problem

Goal:

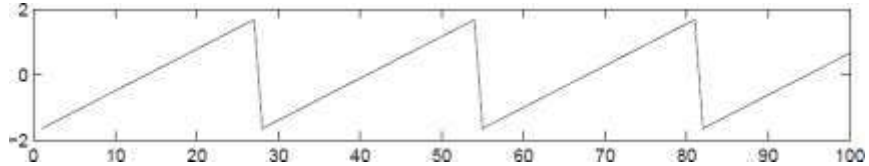
– Recover the unmixed speech signals, best estimate $\mathbf{u_i(t)}$, without knowing **A** or $\mathbf{s_i(t)}$.



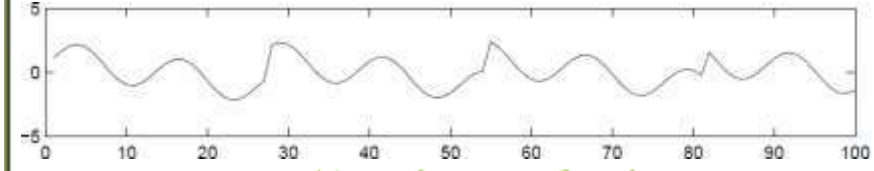
Motivation - Cocktail-Party Problem



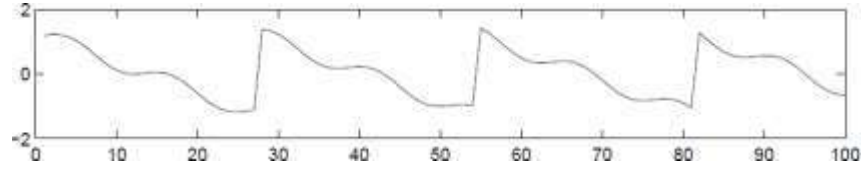
$s_1(t)$: original signal



$s_2(t)$: original signal

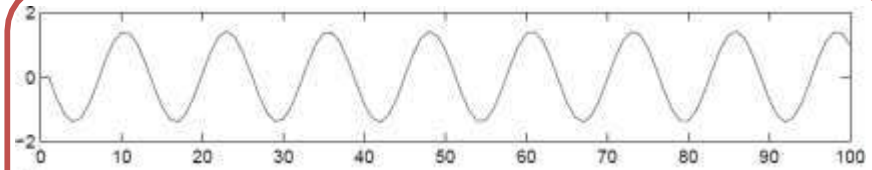


$x_1(t)$: observed mixture

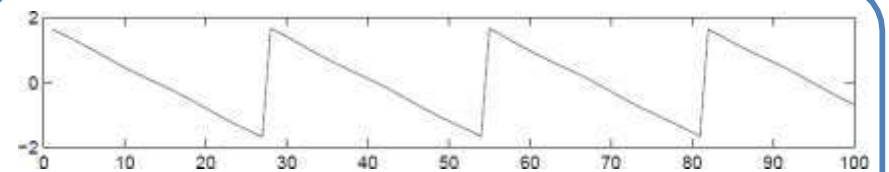


$x_2(t)$: observed mixture

ICA



$u_1(t)$: recovered estimate



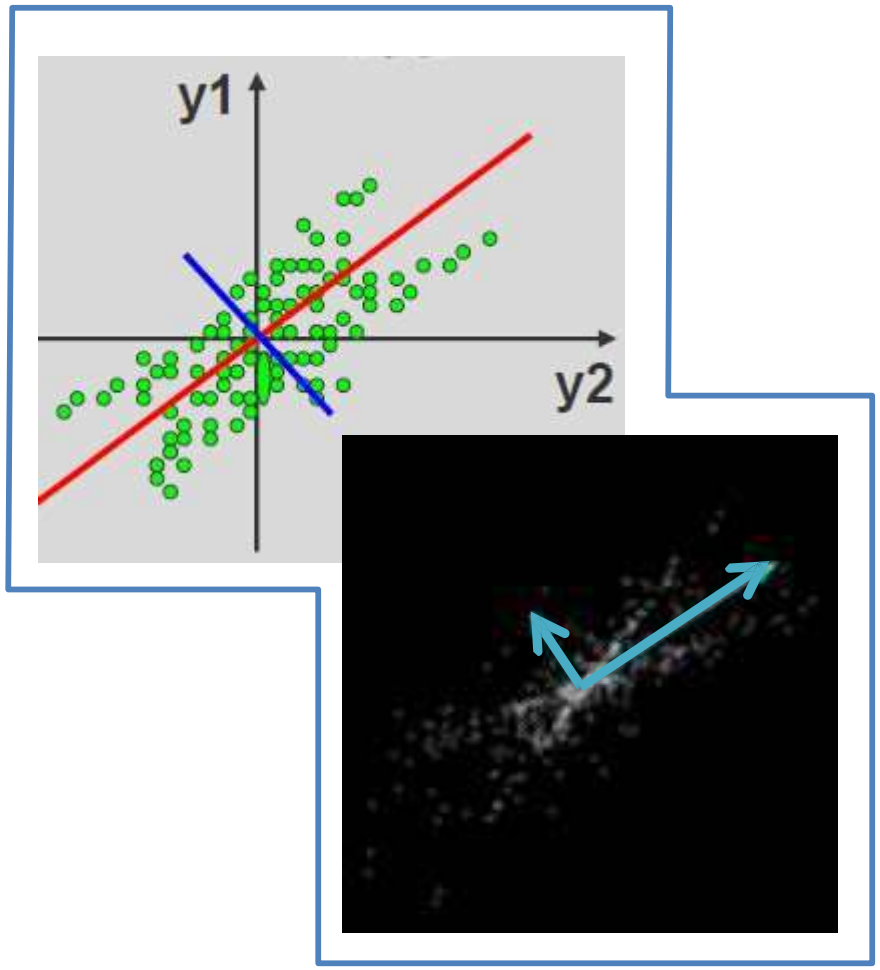
$u_2(t)$: recovered estimate

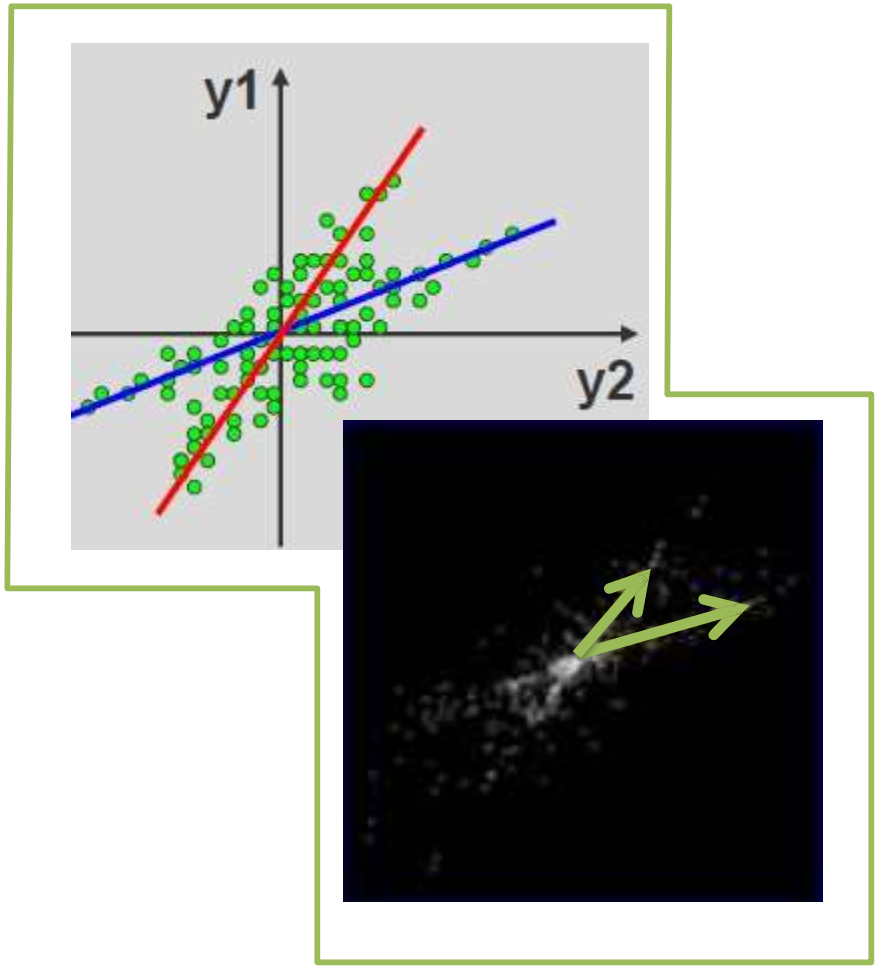
The original signals were very accurately estimated, up to multiplicative signs

ICA versus PCA

- Similarity
  - Feature extraction
  - Dimension reduction
- Difference
  - PCA uses up to *second order moments* of the data to produce uncorrelated components.
  - ICA strives to generate components as independent as possible through minimizing both the second-order and higher-order dependencies in the given data.

ICA versus PCA





PCA finds directions of maximal variance (using second order statistics)

ICA finds directions which maximize independence (using higher order statistics)

Definition of ICA

- Assume that we have *n* mixtures $\mathbf{x_1, \ldots, x_n}$ of *n* independent components:

$$\mathbf{x_j = a_{j1}s_1 + a_{j2}s_2 + \ldots + a_{jn}s_n} \quad \text{for all j}$$

The time index t has dropped in ICA model, since we assume that each mixture and individual components are random variables instead of a proper time signal. Thus the observed values $x_j(t)$, e.g. the microphone signals in the cocktail party problem, are then a sample/realization of this random variable.

- Without loss of generality, we can assume that both the mixture variables and the independent components have zero mean.

If this is not true, then the observable variables x_j can always be centered by subtracting the sample mean, which makes the model zero-mean.

Definition of ICA

- The equation can be expressed using vector-matrix notation,

$$\mathbf{x} = \mathbf{As}$$

where $x = \begin{bmatrix} x_1 \\ . \\ . \\ x_n \end{bmatrix}$, $s = \begin{bmatrix} s_1 \\ . \\ . \\ s_n \end{bmatrix}$ *and* $A = \begin{bmatrix} a_{11} & . & . & a_{1n} \\ . & . & . & . \\ . & . & . & . \\ a_{n1} & . & . & a_{nn} \end{bmatrix}$

x : random vector whose elements are the mixtures $\mathbf{x_1,\ldots,x_n}$

s : random vector whose elements are the sources $\mathbf{s_1,\ldots,s_n}$

A : mixing matrix with elements $\mathbf{a_{ij}}$

- Expression in *columns* of matrix **A**, $x = \sum_{i=1}^{n} a_i s_i$

Definition of ICA

- This statistical model is called *independent component analysis*, or **ICA** model.
- ICA model is a *generative* model, since it describes how the recorded data are generated by mixing the individual components.

ICA – Assumptions

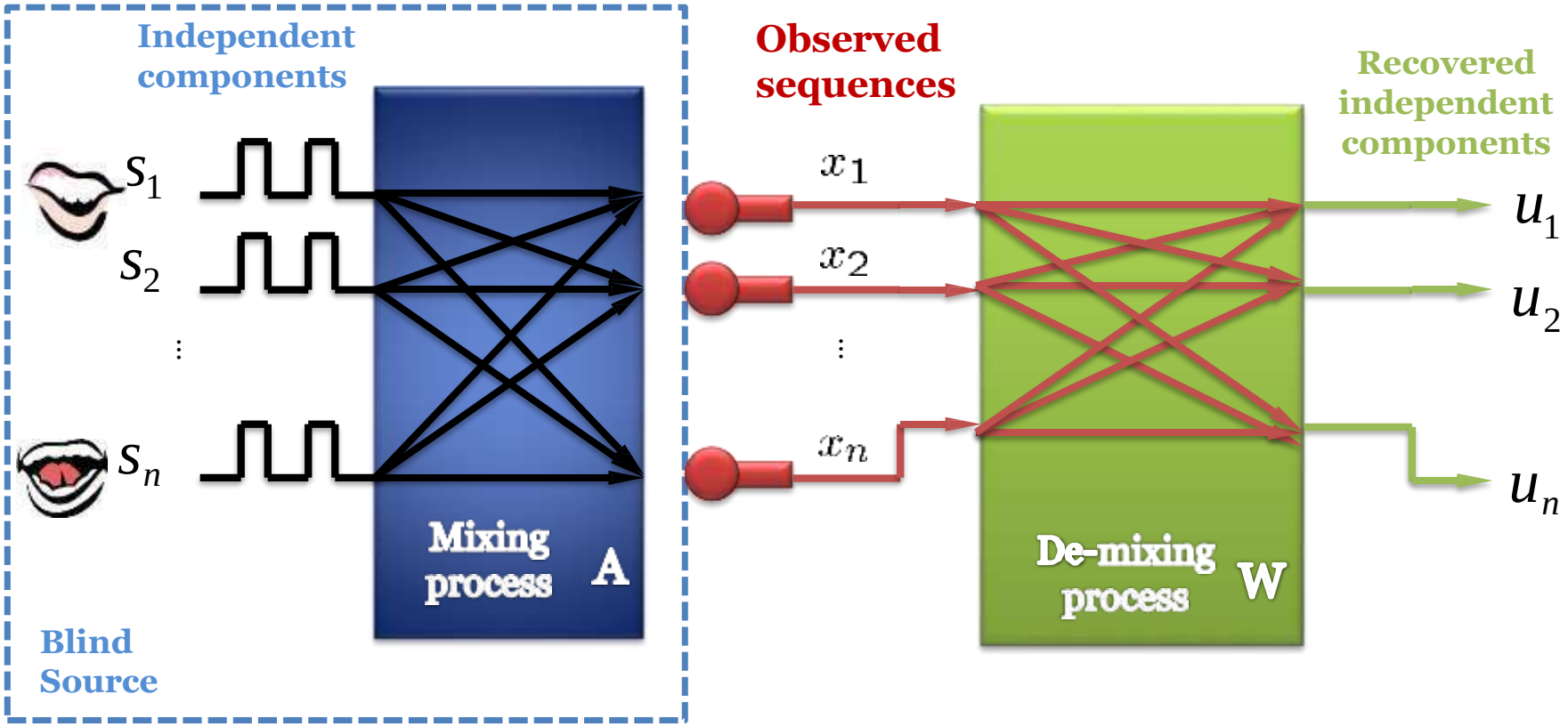
- The starting point for ICA is the very simple assumption that the components $\mathbf{s_i}$ *are statistically independent* – explained later.
- It will be shown that we must also assume that the independent component must have *nongaussian distributions*. However, in the basic model we do not assume these distributions known (if they are known, the problem is considerably simplified.)
- For simplicity, we are also assuming that the unknown mixing matrix is square, but this assumption can be sometimes relaxed.
- Then, after estimating the matrix **A,** we can compute its inverse, say **W,** and obtain the independent component simply by:

$$\mathbf{s = A^{-1}x = Wx}$$

BSS - Blind Source Separation

- ICA is very closely related to the method called *blind source separation (BSS) or blind signal separation.*
- *A* "**source**" means here an original signal, i.e. independent component, like the speaker in a cocktail party problem.
- "**Blind**" means that we know very little, if anything, on the mixing matrix **A**, and make little assumptions on the source signals.
- ICA is one method, perhaps the most widely used, for performing blind source separation.

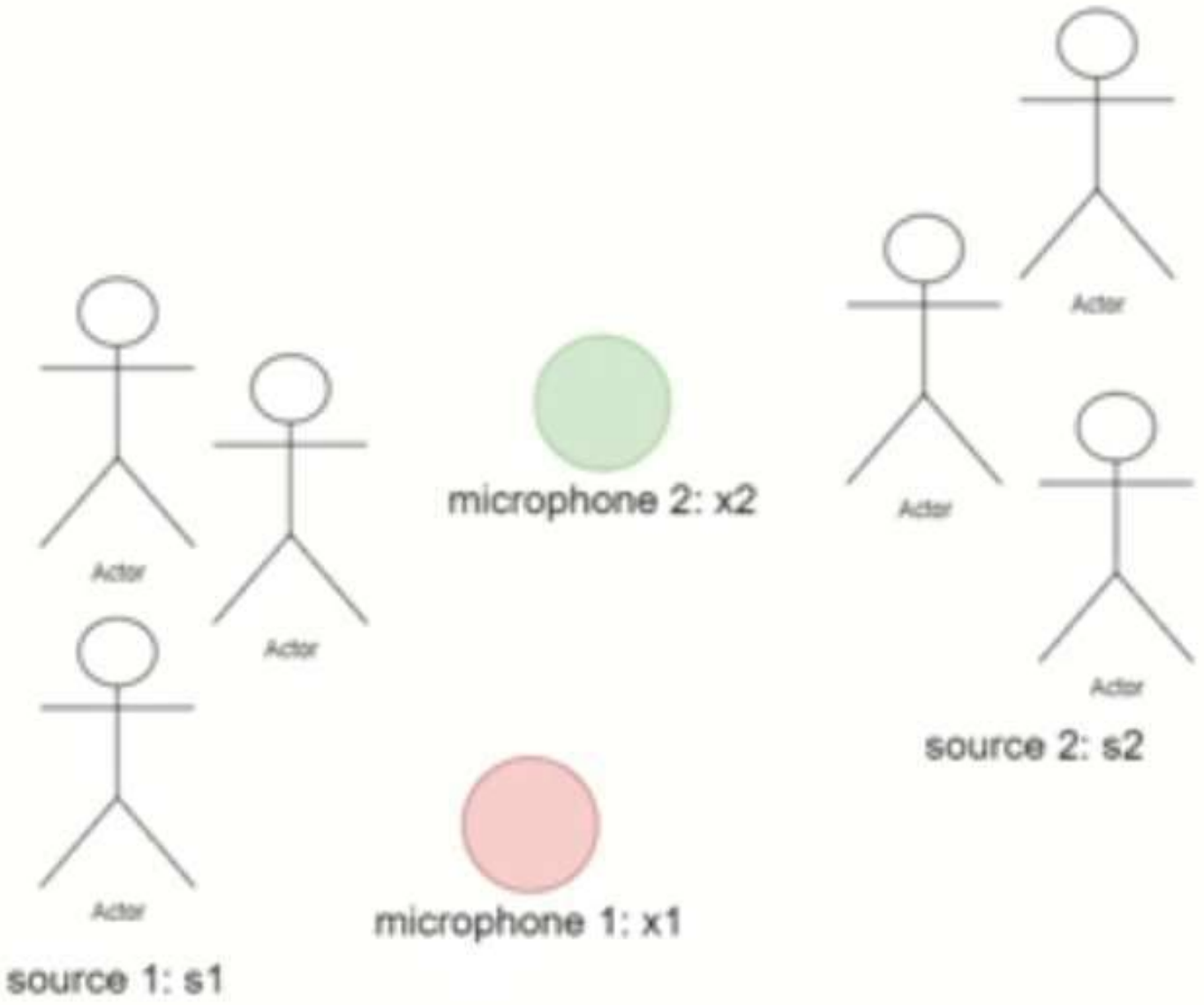
BSS - Blind source separation



Principles of ICA Estimation

- Two popular methods in estimating the ICA model are,
  1. Minimization of Mutual Information
  2. Maximum Likelihood Estimation
  3. ICA Gradient Ascent
  4. Fast ICA

Cocktail Party Effect & Blind Source Separation



Sources and Measured Signals

Assume we have two sources $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$. Assume we have two measurements (sensors) $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

Each measurement is a linear combination of the sources:

$$\boldsymbol{x}_1 = a_{11}\boldsymbol{s}_1 + a_{12}\boldsymbol{s}_2, \tag{1}$$

$$\boldsymbol{x}_1 = a_{21}\boldsymbol{s}_1 + a_{22}\boldsymbol{s}_2. \tag{2}$$

If $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are images, they are reshaped to column vectors, so as $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$. We define:

$$\boldsymbol{x} := \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \quad \boldsymbol{s} := \begin{bmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \end{bmatrix} \tag{3}$$

Thus, we have:

$$\boldsymbol{x} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \boldsymbol{s} = \boldsymbol{A}\,\boldsymbol{s} \tag{4}$$

$\boldsymbol{A}$ is called the **mixing matrix**.

Assumptions

We assume:

1. We center the measurements $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.
2. The sources $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are not Gaussian distributions. Usually natural images and data are not Gaussian.
3. The sources are statistically independent:
   $\boldsymbol{s}_1 \perp\!\!\!\perp \boldsymbol{s}_2 \implies \mathbb{P}(\boldsymbol{s}_1, \boldsymbol{s}_2) = \mathbb{P}(\boldsymbol{s}_1) \times \mathbb{P}(\boldsymbol{s}_2)$

Singular Value Decomposition

In $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{s}$, both $\boldsymbol{A}$ and $\boldsymbol{s}$ are unknown. Only $\boldsymbol{x}$ is known.

Let us apply Singular Value Decomposition (SVD) on the matrix $\boldsymbol{A}$:

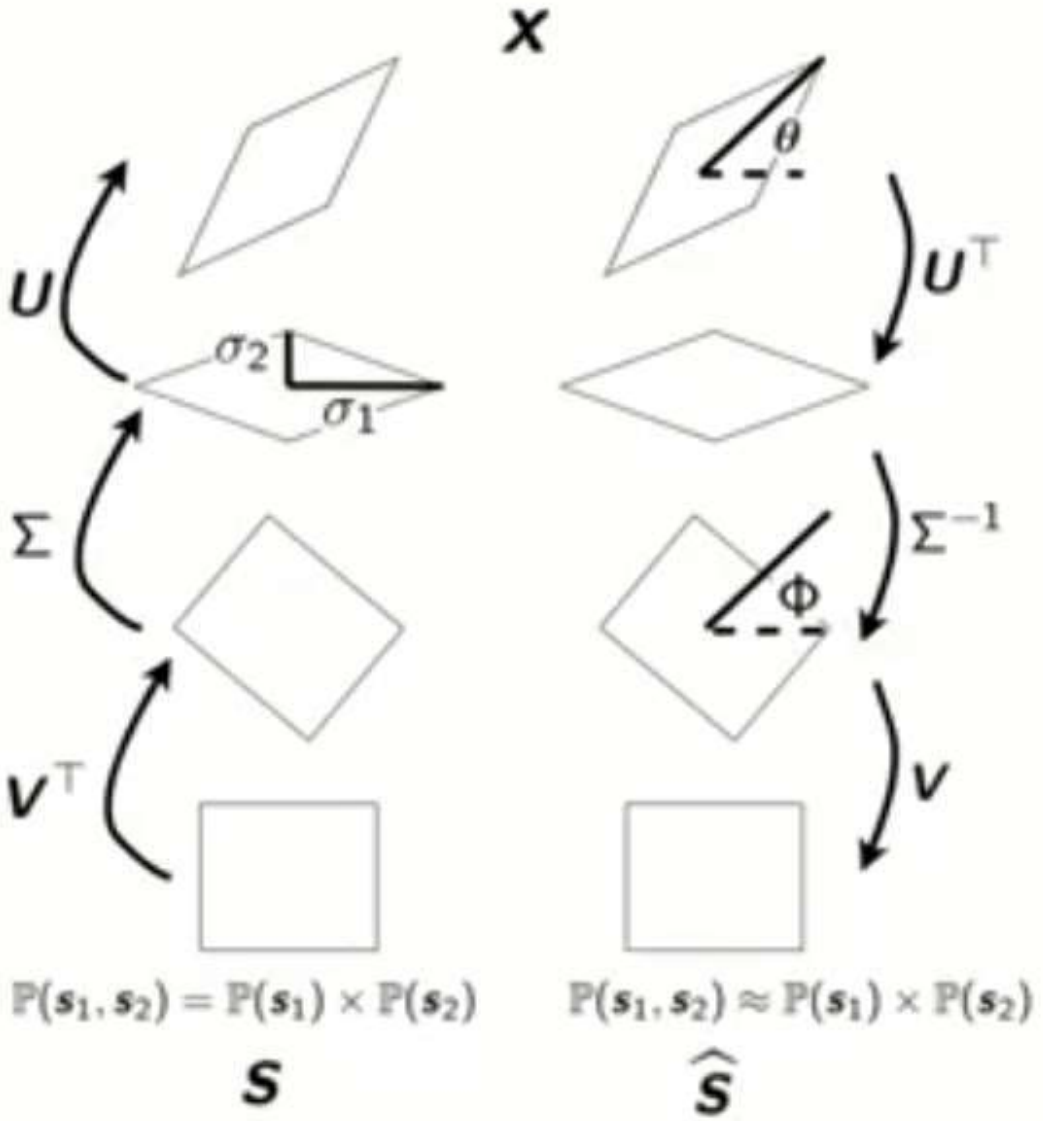
$$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^{\top} \tag{5}$$

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{s} \implies \boldsymbol{x} = \boldsymbol{U}\Sigma\boldsymbol{V}^{\top}\boldsymbol{s} \tag{6}$$

$$\widehat{\boldsymbol{s}} = \boldsymbol{A}^{-1}\boldsymbol{x} = (\boldsymbol{U}\Sigma\boldsymbol{V}^{\top})^{-1}\boldsymbol{x} = \boldsymbol{V}^{-\top}\Sigma^{-1}\boldsymbol{U}^{-1}\boldsymbol{x} = \boldsymbol{V}\Sigma^{-1}\boldsymbol{U}^{\top}\boldsymbol{x} \tag{7}$$

Singular Value Decomposition

$\boxed{A = U\Sigma V^\top}$ and $\boxed{\hat{s} = V\Sigma^{-1}U^\top x}$



Step 1: on $U^\top$

$\widehat{s} = V\Sigma^{-1}U^\top x$, Let us consider $U^\top x$. We assume the statistical distributions $\mathbb{P}(s_1)$ and $\mathbb{P}(s_2)$ have zero mean. We minimize the second moment or the variance:

$$\begin{aligned}
\mathbb{V}\mathrm{ar}(\theta) &= \sum_{j=1}^{n} \left([x_1(j), x_2(j)] \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \right)^2 \\
&= \sum_{j=1}^{n} \left(x_1(j)\cos(\theta) + x_2(j)\sin(\theta) \right)^2 \\
&= \sum_{j=1}^{n} \left(x_1^2(j)\cos^2(\theta) + 2x_1(j)x_2(j)\sin(\theta)\cos(\theta) + x_2^2(j)\sin^2(\theta) \right)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathbb{V}\mathrm{ar}(\theta)}{\partial \theta} = \sum_{j=1}^{n} \Big(& -2x_1(j)^2\cos(\theta)\sin(\theta) \\
& + 2x_1(j)x_2(j)\left(\cos^2(\theta) - \sin^2(\theta)\right) + 2x_2^2\sin(\theta)\cos(\theta) \Big)
\end{aligned}$$

Step 1: on $U^\top$

$$\frac{\partial \mathbb{V}\mathrm{ar}(\theta)}{\partial \theta} = \sum_{j=1}^{n} \Big(-2x_1(j)^2 \cos(\theta)\sin(\theta) + 2x_1(j)x_2(j)\big(\cos^2(\theta) - \sin^2(\theta)\big) + 2x_2^2 \sin(\theta)\cos(\theta) \Big)$$

$$= \sum_{j=1}^{n} \Big((x_2^2(j) - x_1^2(j)) \sin(2\theta) + 2x_1(j)x_2(j)\cos(2\theta) \Big) \overset{\text{set}}{=} 0$$

$$\implies \sin(2\theta) \sum_{j=1}^{n} (x_2^2(j) - x_1^2(j)) = -2\cos(2\theta) \sum_{j=1}^{n} x_1(j)x_2(j)$$

$$\implies \tan(2\theta) = \frac{-2\sum_{j=1}^{n} x_1(j)x_2(j)}{\sum_{j=1}^{n}(x_2^2(j) - x_1^2(j))}$$

$$\implies \boxed{\theta = \frac{1}{2}\tan^{-1}\left(\frac{-2\sum_{j=1}^{n} x_1(j)x_2(j)}{\sum_{j=1}^{n}(x_2^2(j) - x_1^2(j))}\right)} \tag{8}$$

Step 1: on $U^\top$

$$\boxed{\theta = \frac{1}{2}\tan^{-1}\left(\frac{-2\sum_{j=1}^{n} x_1(j)x_2(j)}{\sum_{j=1}^{n}(x_2^2(j) - x_1^2(j))}\right)} \tag{9}$$

In polar coordinate, we can represent measurements as $x_1 = r\cos(\psi)$ and $x_2 = r\sin(\psi)$, so:

$$\boxed{\theta = \frac{1}{2}\tan^{-1}\left(\frac{\sum_{j=1}^{n} r^2(j)\sin(2\psi_j)}{\sum_{j=1}^{n} r^2(j)\cos(2\psi_j)}\right)} \tag{10}$$

$$U = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{11}$$

$$\implies \boxed{U^\top = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}} \tag{12}$$

Step 2: on Σ^{-1}

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \tag{13}$$

$$\Sigma^{-1} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 \\ 0 & \frac{1}{\sigma_2} \end{bmatrix} \tag{14}$$

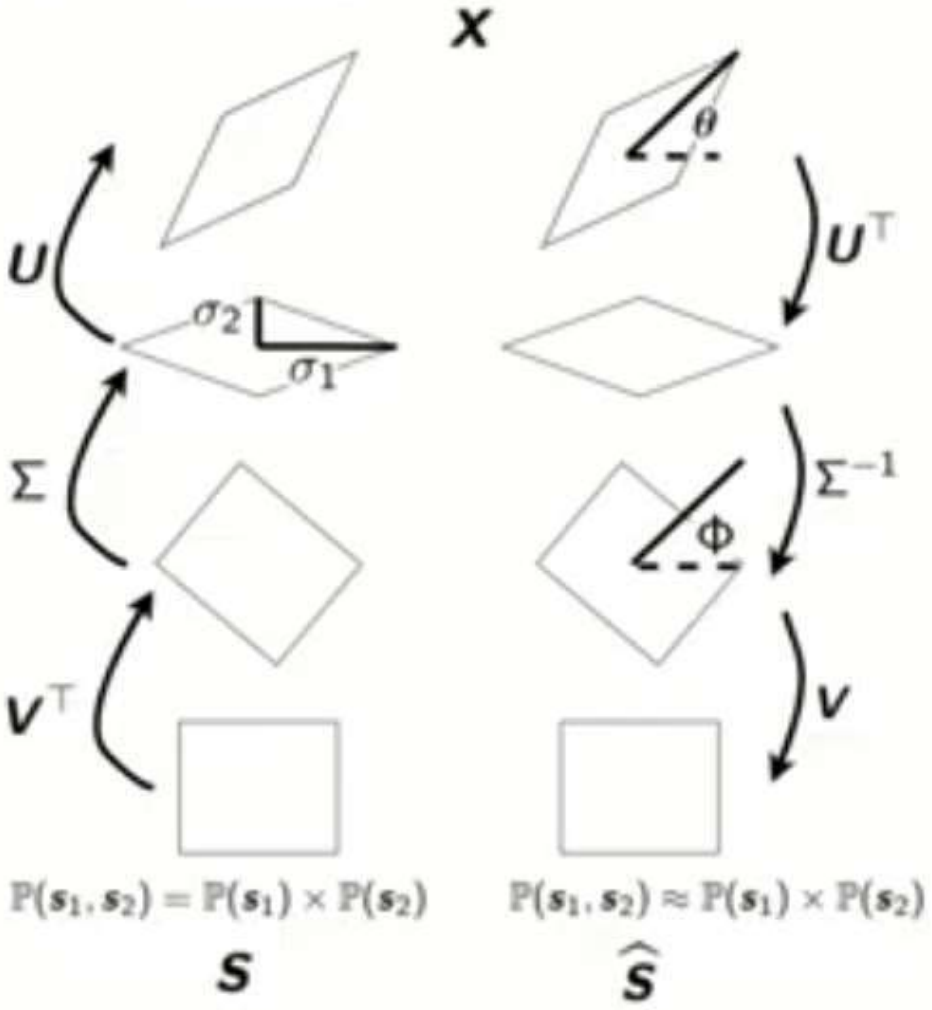
$$\boxed{\Sigma^{-1} \approx \begin{bmatrix} \frac{1}{\rho_1} & 0 \\ 0 & \frac{1}{\rho_2} \end{bmatrix}} \tag{15}$$

$$\boxed{\rho_1 = \sum_{j=1}^{n} \left([\boldsymbol{x}_1(j), \boldsymbol{x}_2(j)] \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \right)^2} \tag{16}$$

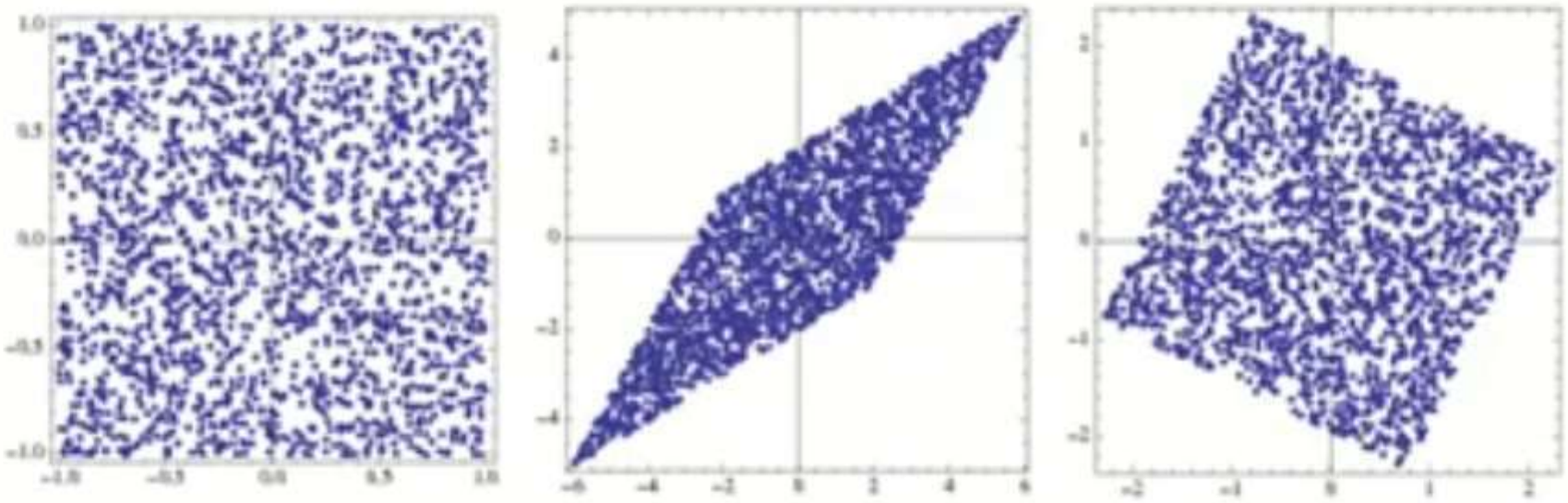
$$\boxed{\rho_2 = \sum_{j=1}^{n} \left([\boldsymbol{x}_1(j), \boldsymbol{x}_2(j)] \begin{bmatrix} \cos(\theta - \frac{\pi}{2}) \\ \sin(\theta - \frac{\pi}{2}) \end{bmatrix} \right)^2} \tag{17}$$

Whitening

Recall $\widehat{\boldsymbol{s}} = \boldsymbol{V}\Sigma^{-1}\boldsymbol{U}^\top \boldsymbol{x}$. The steps 1 and 2 result in whitening: $\Sigma^{-1}\boldsymbol{U}^\top \boldsymbol{x}$. After whitening, the covariance matrix of measurements becomes the identity matrix (see below figure).



Whitening Example



The credit of this image is for https://dsp.stackexchange.com/questions/80/what-are-the-proper-pre-processing-steps-to-perform-independent-component-analys

Step 3: on V

The first moment was assumed to be zero. The second moment was taken care of in step 1. Usually natural images or data are not much skewed so we ignore the third moment (skewness). The fourth moment is Kurtosis which is a statistical measure that defines how heavily the tails of a distribution differ from the tails of a Gaussian distribution. We have assumed $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ do not have Gaussian distributions so their Kurtosis is not zero.

$$K(\phi) = \sum_{j=1}^{n} \left([\boldsymbol{x}_1, \boldsymbol{x}_2] \begin{bmatrix} \cos(\phi) \\ \sin(\phi) \end{bmatrix} \right)^4 \tag{18}$$

$$\frac{\partial K(\phi)}{\partial \phi} \overset{\text{set}}{=} 0 \implies \boxed{\phi = \frac{1}{4} \tan^{-1} \left(\frac{\sum_{j=1}^{n} r^2(j) \sin(4\psi_j)}{\sum_{j=1}^{n} r^2(j) \cos(4\psi_j)} \right)} \tag{19}$$

$$\boxed{\boldsymbol{V} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}} \tag{20}$$

Ambiguities of ICA

Two major ambiguities:

1. The variances (energies) of the independent components $\mathbf{s_i}$ cannot be determined.
   - Since both **s** and **A** are unknown, any scalar multiplier of source $\mathbf{s_i}$ can be cancelled by dividing the corresponding column $\mathbf{a_i}$ of **A** with the same scalar value.
   - As a consequence, we may quite as well fix the magnitudes of the independent components; as they are random variables, the most natural way to do this is to assume that each has unit variance: $E\{s_i^2\} = 1$.
   - Note that this still leaves the ambiguity of the sign: we could multiply the an independent component by -1 without affecting the model. This ambiguity is, fortunately, insignificant in most applications.

Ambiguities of ICA

Two major ambiguities:

2. The order of the independent components cannot be determined.

   - Again, since **s** and **A** are unknown, order of the terms in the model can be changed freely, and we can call any of the independent components the first one.

Examples

Blind source separation for:

- **music** separation:
  `https://cnl.salk.edu/~tewon/Blind/blind_audio.html`
- **speech** separation:
  `https://cnl.salk.edu/~tewon/Blind/blind_audio.html`
- **EEG** separation: [1]
- **fMRI** separation [2]
- learning independent **filters** on natural **images**:
  `https://pydeep.readthedocs.io/en/latest/tutorials/ICA_natural_images.html`

Useful Resources To Read

- ICA using **maximum likelihood estimation**: Tutorial YouTube videos by Prof. Andrew Ng at the Stanford University: [Click here]
- Survey paper: "Survey on independent component analysis" [3]
- Tutorial paper: "A tutorial on independent component analysis" [4]
- Tutorial paper: "Independent component analysis: A tutorial" [5]
- Survey paper: "An overview of independent component analysis and its applications" [6]
- A book on ICA: [7]

Time Series and Dynamic Time Warping

Sequential Data

- **Sequential data**, as the name implies, are **sequences**.
- What is the difference between a **sequence** and a **set**?
- A sequence $\boldsymbol{X}$ is a set of elements, together with a **total order** imposed on those elements.
  - A **total order** describes, for any two elements $\boldsymbol{x_1}, \boldsymbol{x_2}$, which of them comes before and which comes after.
- Examples of sequential data:
  - Strings: sequences of characters.
  - Time series: sequences of vectors.

Time Series

- A **time series** is a **sequence** of observations made over time.
- Examples:
  - Stock market prices (for a single stock, or for multiple stocks).
  - Heart rate of a patient over time.
  - Position of one or multiple people/cars/airplanes over time.
  - Speech: represented as a sequence of audio measurements at discrete time steps.
  - A musical melody: represented as a sequence of pairs (note, duration).

Applications of Time Series Classification

- Predicting future prices (stock market, oil, currencies…).
- Heart rate of a patient over time:
  - Is it indicative of a healthy heart, or of some disease?
- Position of one or multiple people/cars/airplanes over time.
  - Predict congestion along a route suggested by the GPS.
- Speech recognition.
- Music recognition.
  - Sing a tune to your phone, have it recognize the song.
  - Recognize the genre of a song based on how it sounds.

Time Series Example: Signs

- 0.5 to 2 million users of American Sign Language (ASL) in the US.
- Different regions in the world use different sign languages.
  - For example, British Sign Language (BSL) is different than American Sign Language.
- These languages have vocabularies of thousands of signs.
- We will use sign recognition as our example application, as we introduce methods for time series classification.

Example: The ASL Sign for "again"

Example: The ASL Sign for "bicycle"

Representing Signs as Time Series

- A sign is a video (or part of a video).
- A video is a sequence of frames.
  - A sequence of images, which, when displayed rapidly in succession, create the illusion of motion.
- At every frame i, we extract a **feature vector** $\boldsymbol{x}_i$.
  - How should we define the feature vector? This is a (challenging) computer vision problem.
  - The methods we will discuss work with **any** choice we make for the feature vector.
- Then, the entire sign is represented as time series $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_D)$.
  - D is the length of the sign video, measured in frames.

Feature Vectors

- Finding good feature vectors for signs is an active research problem in computer vision.

Feature Vectors

- We choose a simple approach: the feature vector at each frame is the (x, y) pixel location of the right hand at that frame.

Frame 1:

Feature vector

$\boldsymbol{x}_1 = (192,205)$

Feature Vectors

- We choose a simple approach: the feature vector at each frame is the (x, y) pixel location of the right hand at that frame.

Frame 2:
Feature vector
$\boldsymbol{x}_2 = (190,201)$

Feature Vectors

- We choose a simple approach: the feature vector at each frame is the (x, y) pixel location of the right hand at that frame.

Frame 3:
Feature vector
$\boldsymbol{x}_3 = (189,197)$

Time Series for a Sign

- Using the previous representation, for sign "AGAIN" we end up with this time series:

((192, 205),(190, 201),(190, 194),(191, 188),(194, 182),(205, 183),
(211, 178),(217, 171),(225, 168),(232, 167),(233, 167),(238, 168),
(241, 169),(243, 176),(254, 177),(256, 179),(258, 186),(269, 194),
(271, 202),(274, 206),(276, 207),(277, 208)).

- It is a sequence of 22 2D vectors.

Time Series Classification

Training sign for "AGAIN".

Time Series Classification

Training sign for "BICYCLE".

Time Series Classification

Suppose our training set only contains those two signs: "AGAIN" and "BICYCLE".

We get this test sign.

Does it mean "AGAIN", or "BICYCLE"?

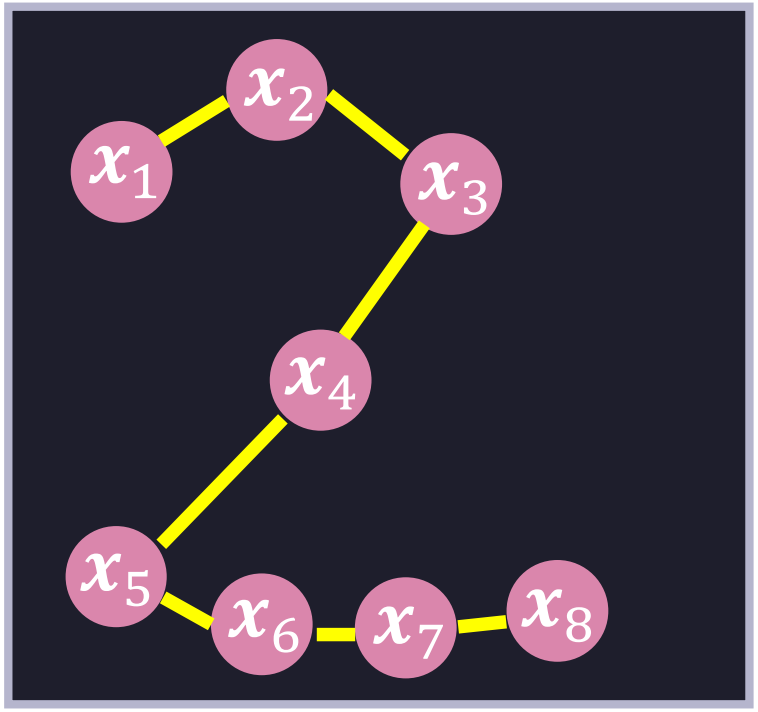
Time Series Classification

Does this test sign mean "AGAIN", or "BICYCLE"?

We can use nearest-neighbor classification.

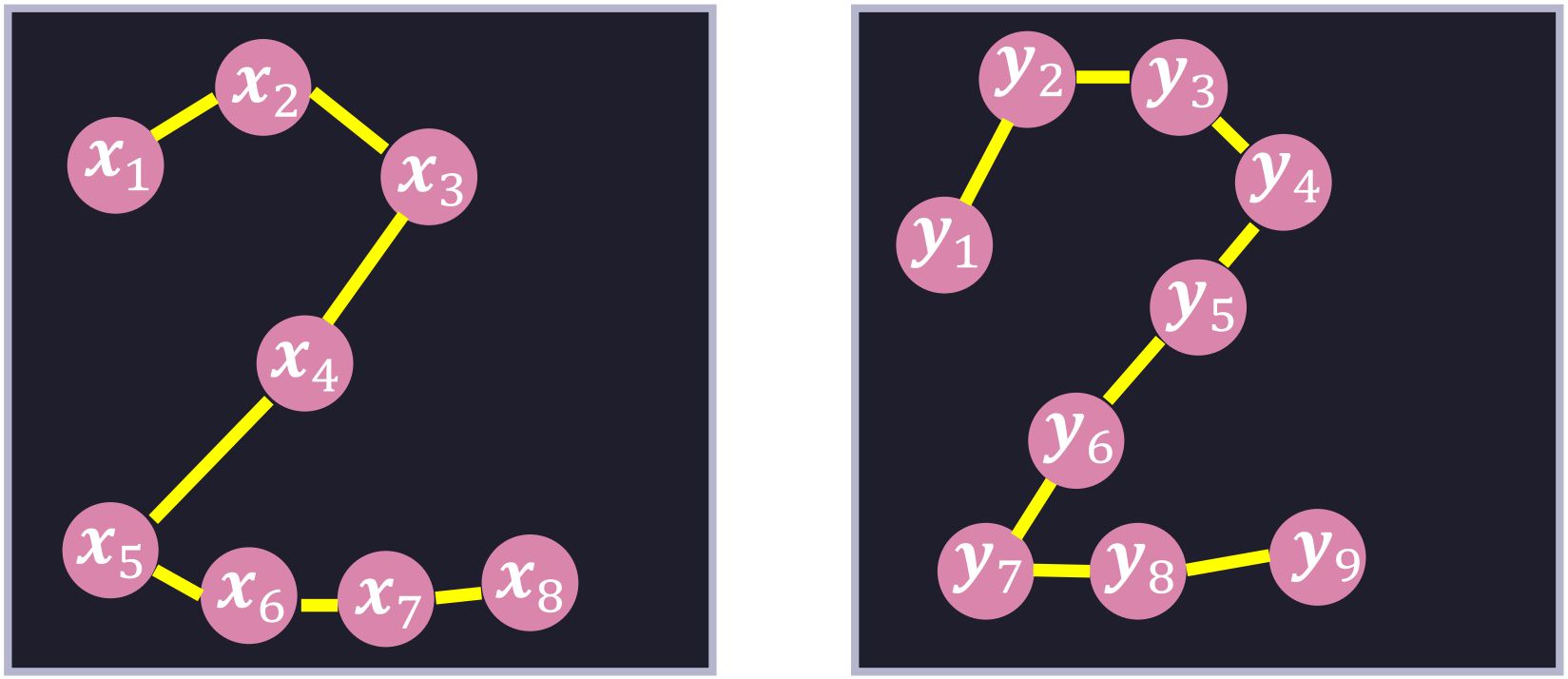
Big question: what distance measure should we choose?

Visualizing a Time Series



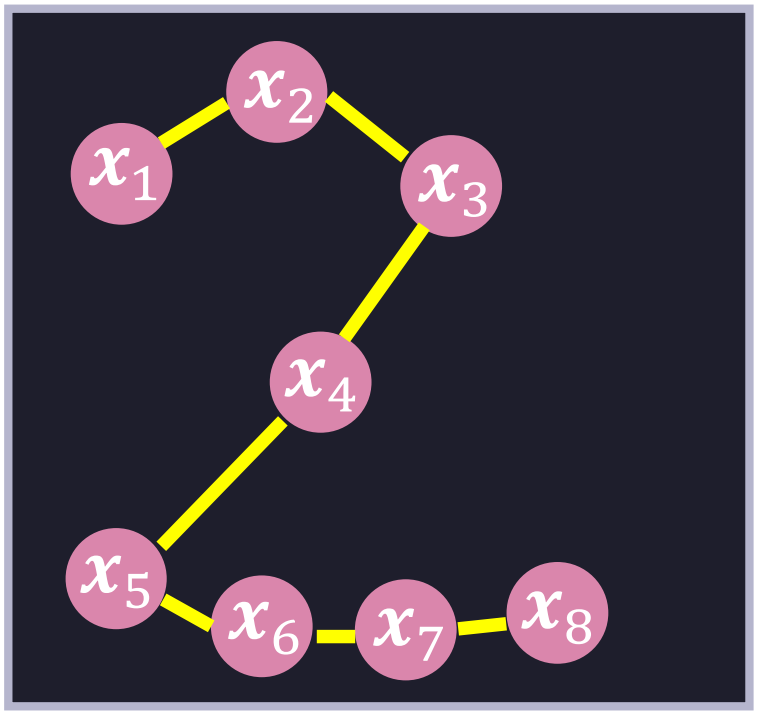
- Here is a visualization of one time series $\boldsymbol{X}$.
- Each feature vector (as before) is a point in 2D.
- The time series has eight elements. $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_D)$.
- For example, x_4 indicates the position of 2D point $\boldsymbol{x}_4$.

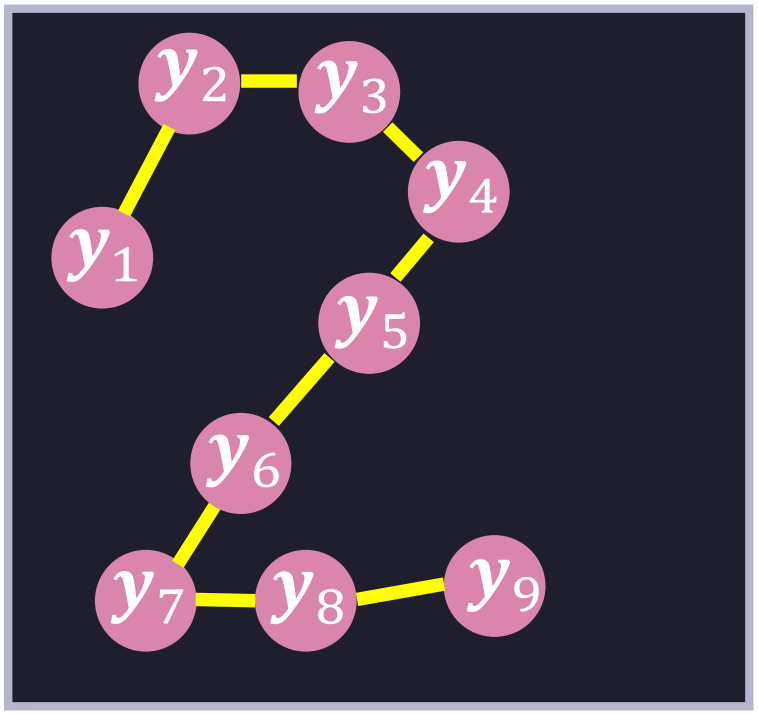
Comparing Time Series



- Here is a visualization of two time series, $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- How do we measure the distance between two time series?

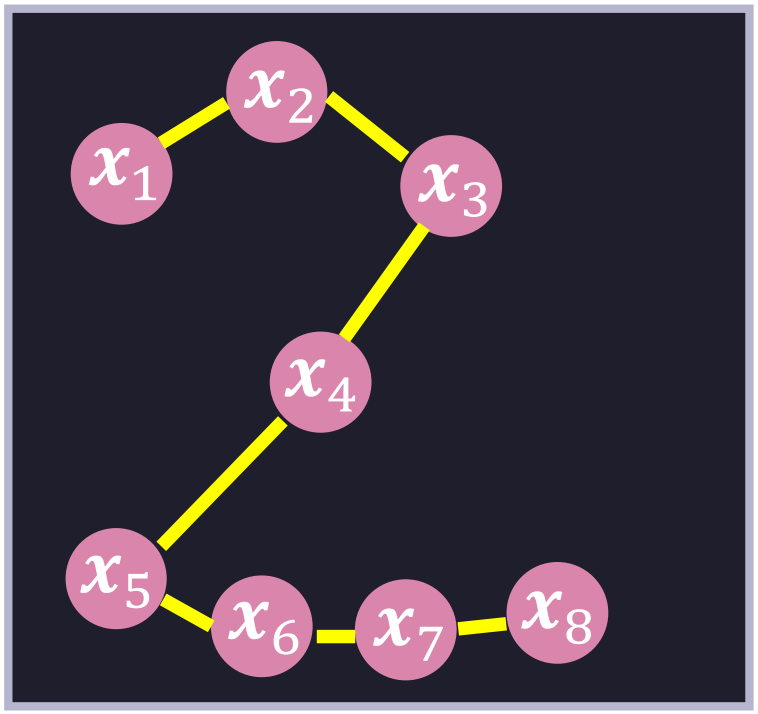
Comparing Time Series

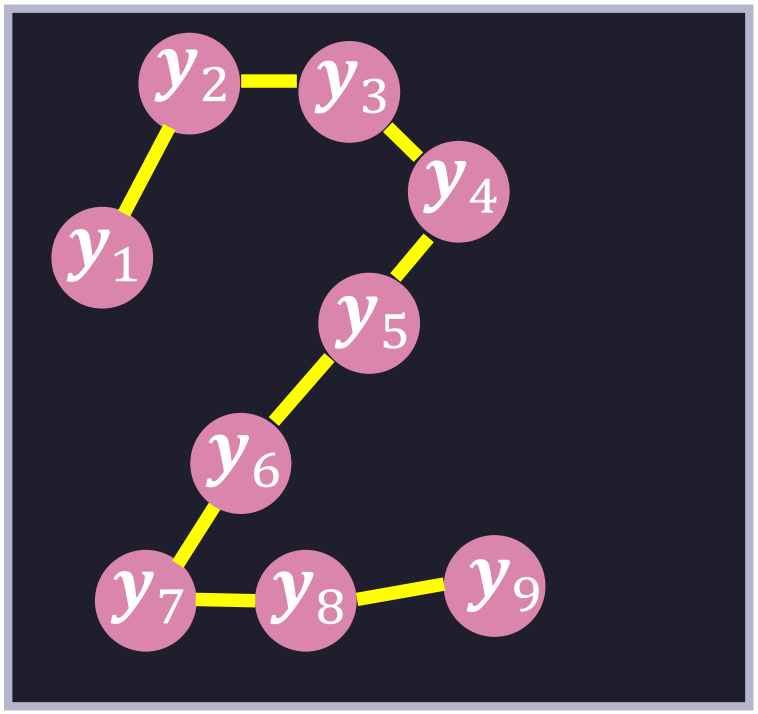




- We could possibly use the Euclidean distance.
  - The first time series is a 16-dimensional vector.
  - The second time series is an 18-dimensional vector.
- However, there are issues…

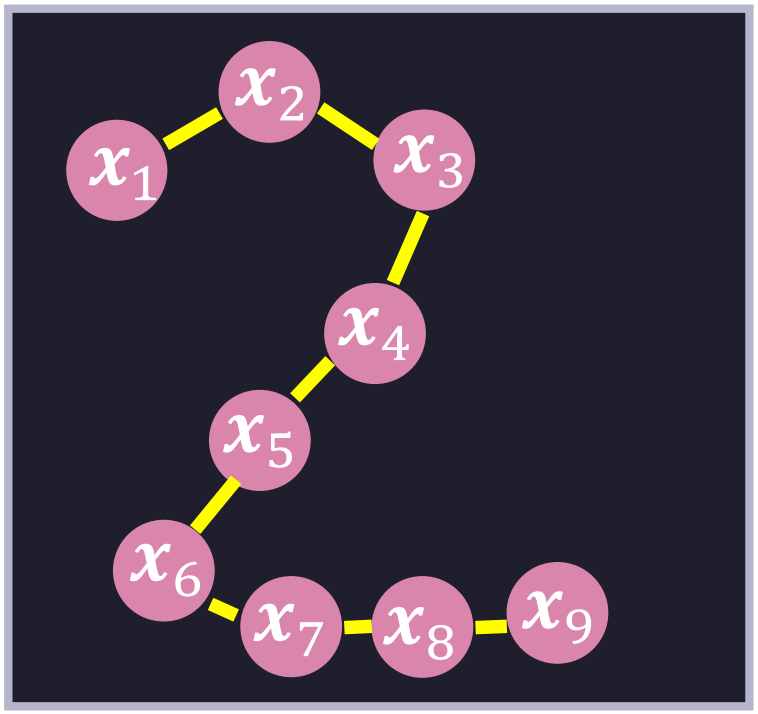
Problems with the Euclidean Distance

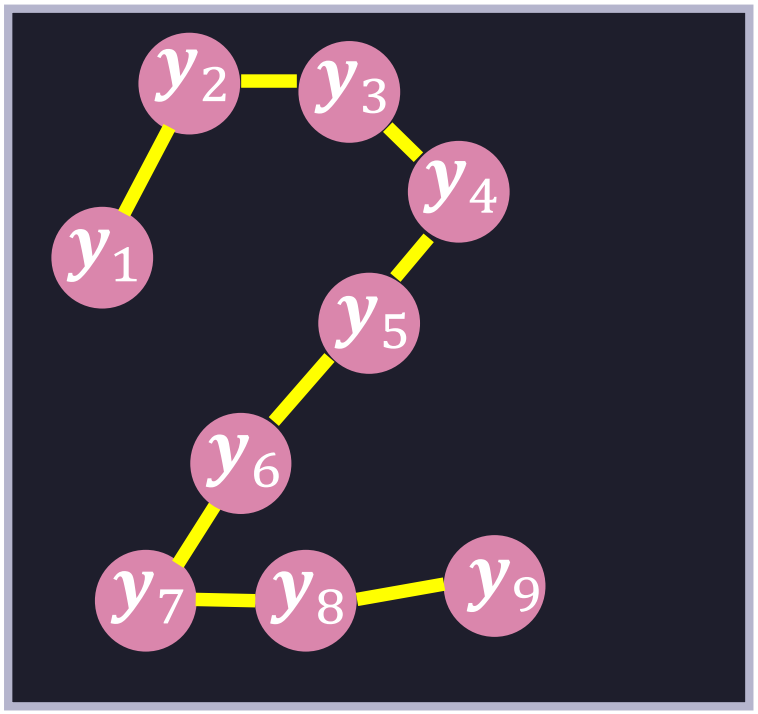




- The two vectors have different numbers of dimensions.
- We could fix that by "stretching" the first time series to also have 9 elements.

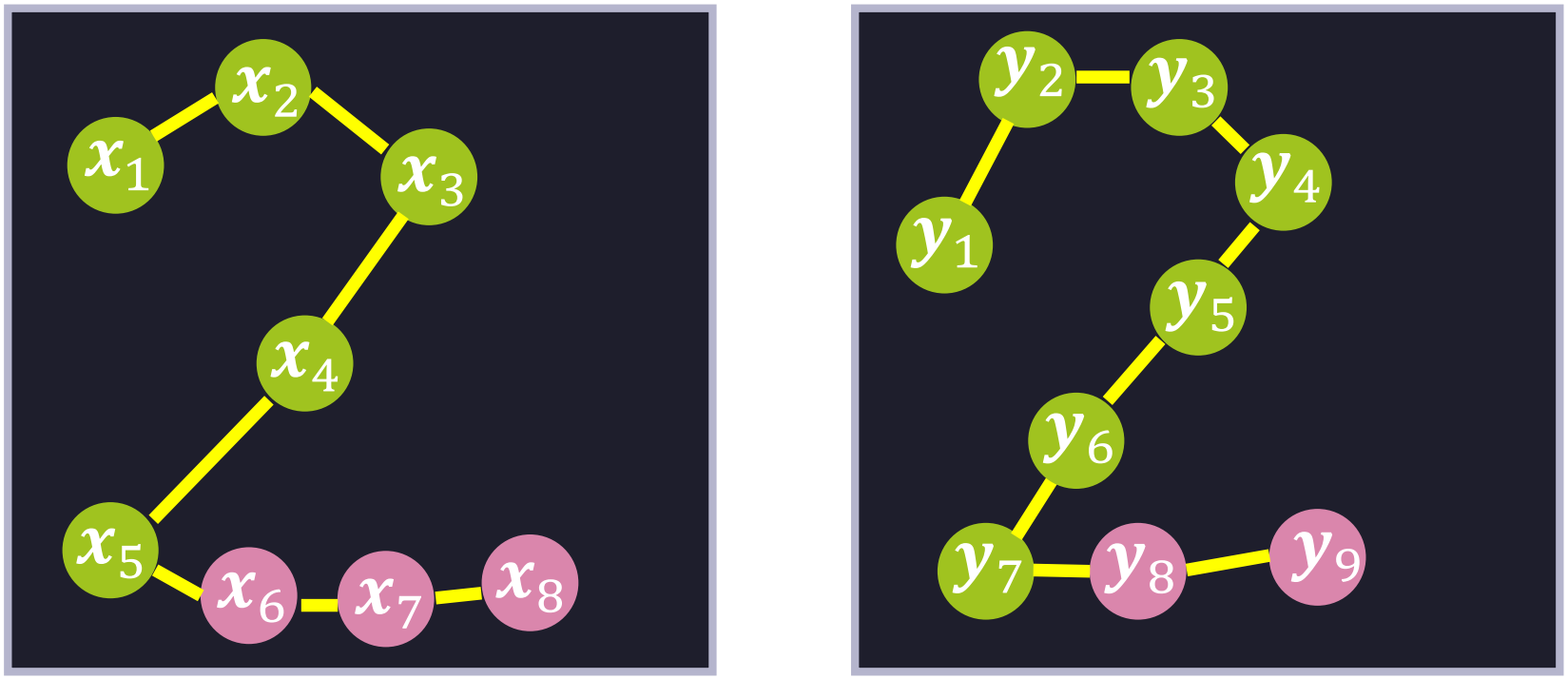
Problems with the Euclidean Distance





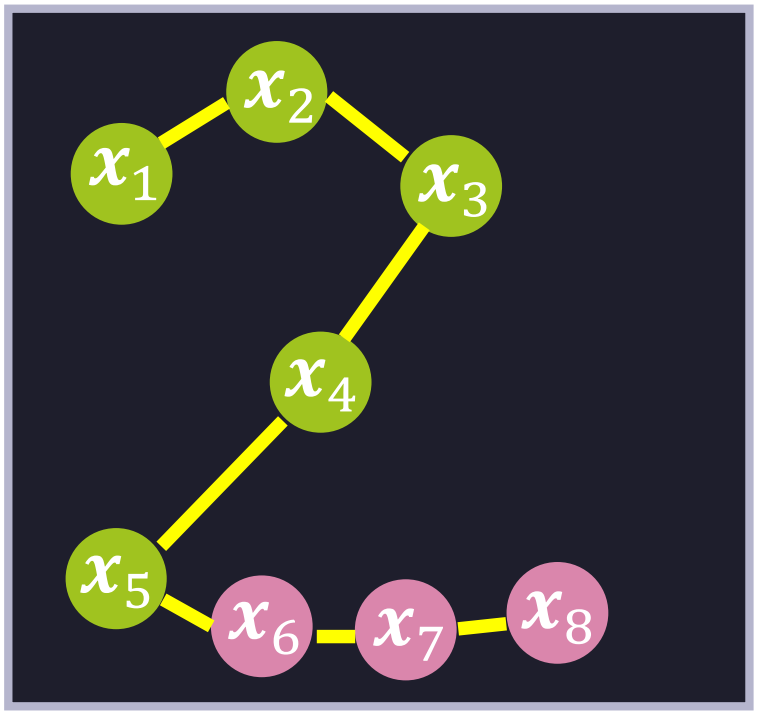
- Here, the time series on the left has been converted to have nine elements, using interpolation.
  - We will not explore that option any further, because…

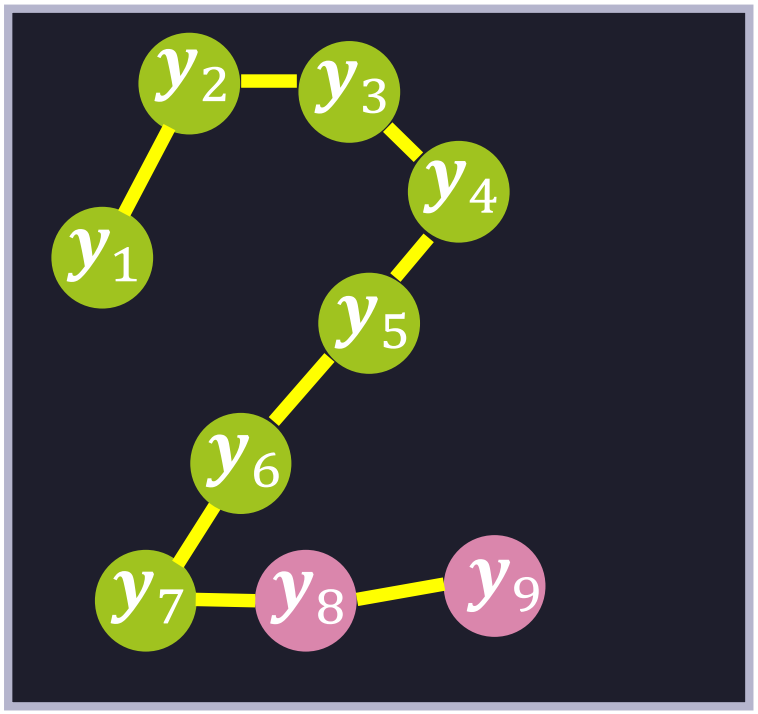
Problems with the Euclidean Distance



- Bigger problem: the two time series correspond to a similar pattern being performed with **variable speed**.
  - The first five elements of the first time series visually match the first seven elements of the second time series.

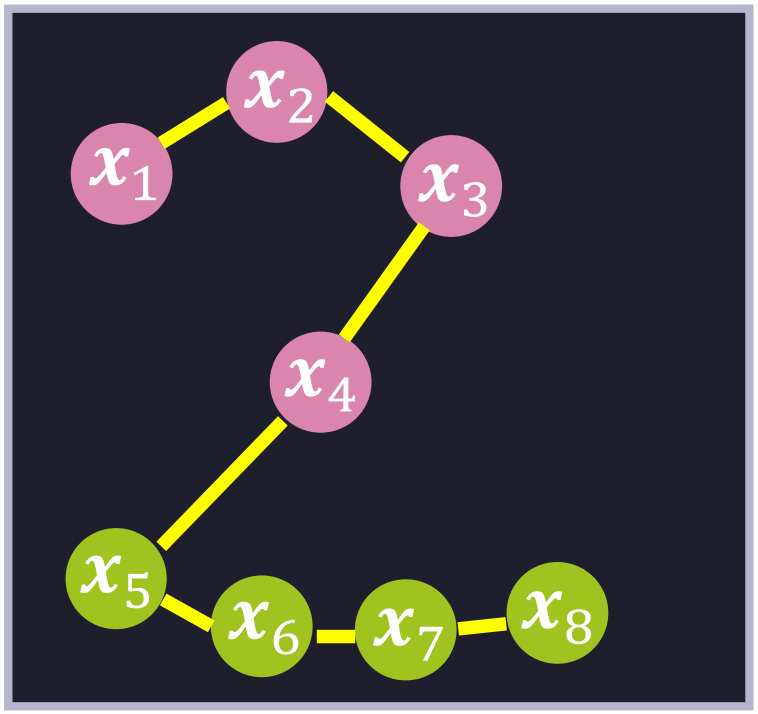
Problems with the Euclidean Distance

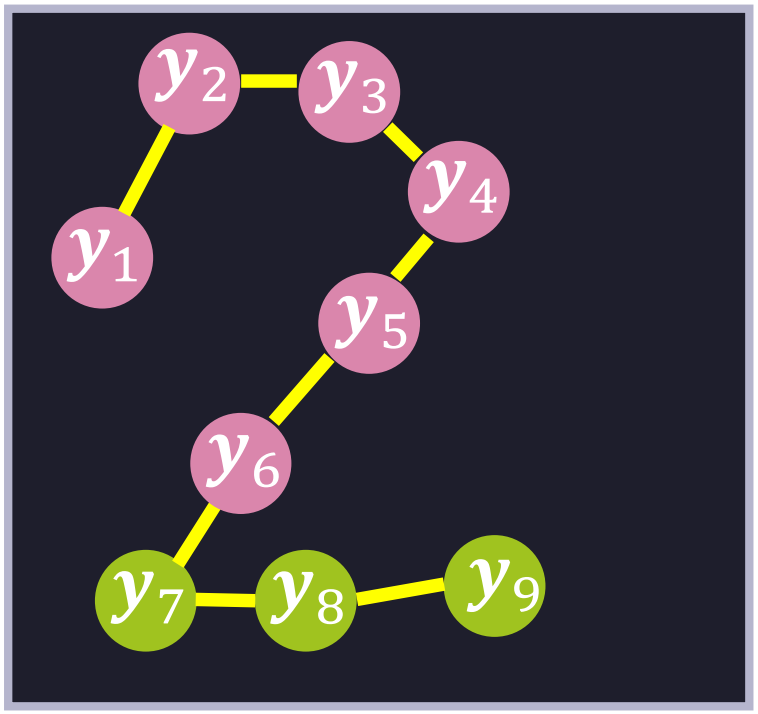




- So, the first time series had a faster speed than the second time series initially.
  - It took 5 time ticks for the first time series, and seven time ticks for the second time series, to move through approximately the same trajectory.

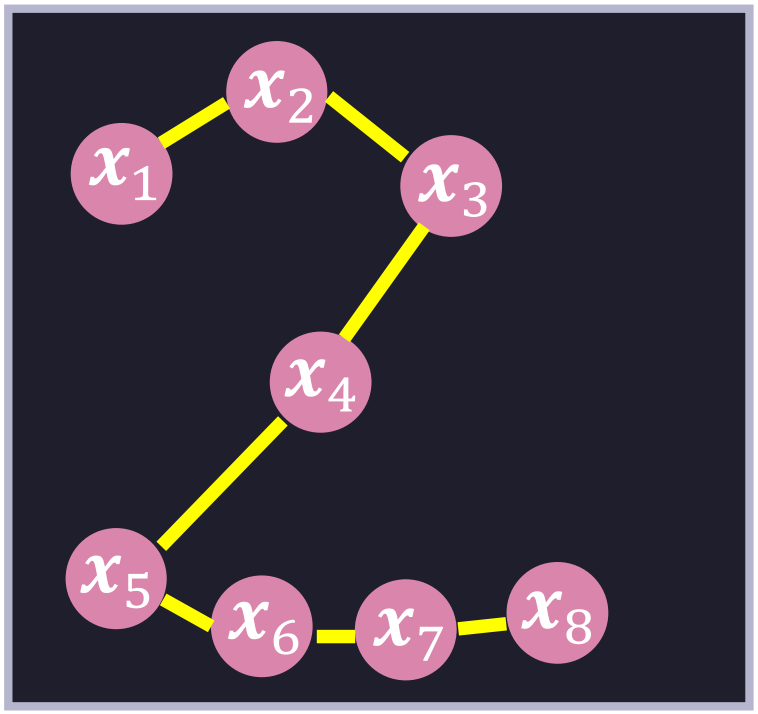
Problems with the Euclidean Distance

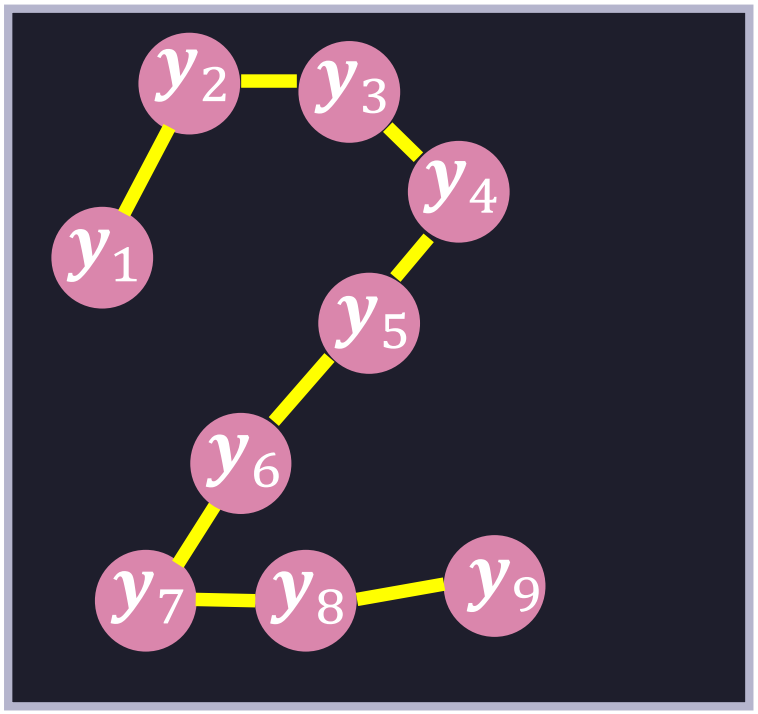




- Similarly, the last four elements of the first time series visually match the last three elements of the second time series.
- So, the first time series had a faster speed than the second time series in the last part.
  - 4 time ticks for the first time series, 3 time ticks for the second one.

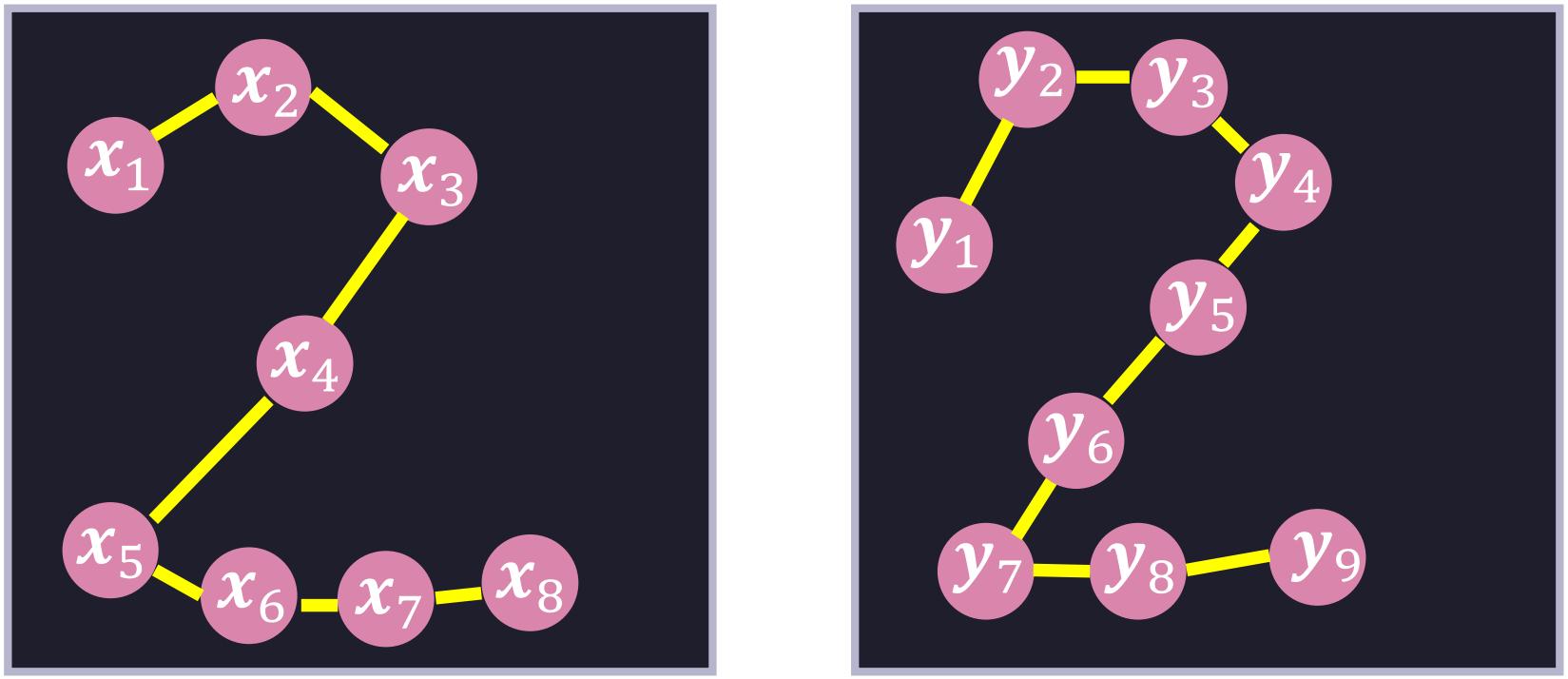
Time Series Alignment





- An alignment is a correspondence between elements of two time series.
- To reliably measure the similarity between two time series, we need to figure out, for each element of the first time series, which elements of the second time series it **corresponds** to.

Time Series Alignment

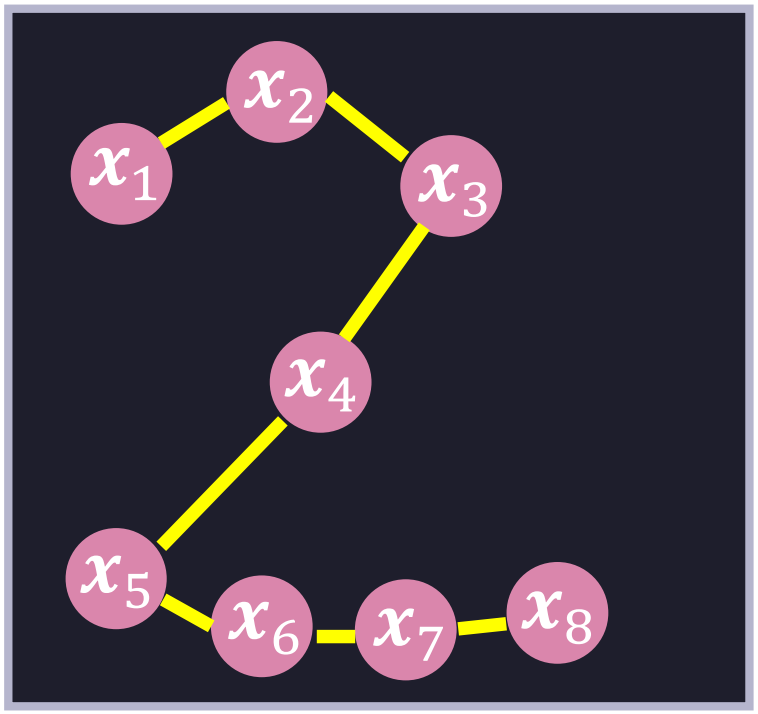


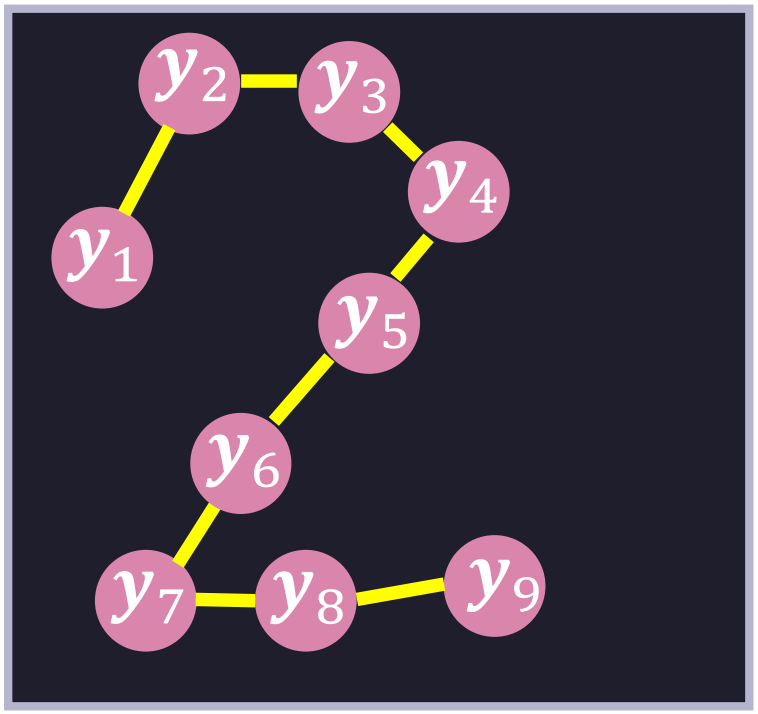
- An alignment $\boldsymbol{A}$ between $\boldsymbol{X}$ and $\boldsymbol{Y}$ is a sequence of pairs of indices:

$$\boldsymbol{A} = \Big(\big(a_{1,1}, a_{1,2}\big), \big(a_{2,1}, a_{2,2}\big), \dots, \big(a_{R,1}, a_{R,2}\big)\Big)$$

- Element $\big(a_{i,1}, a_{i,2}\big)$ of alignment $\boldsymbol{A}$ specifies that element $\boldsymbol{x}_{a_{i,1}}$ of $\boldsymbol{X}$ corresponds to element $\boldsymbol{y}_{a_{i,2}}$ of the other time series.

Time Series Alignment



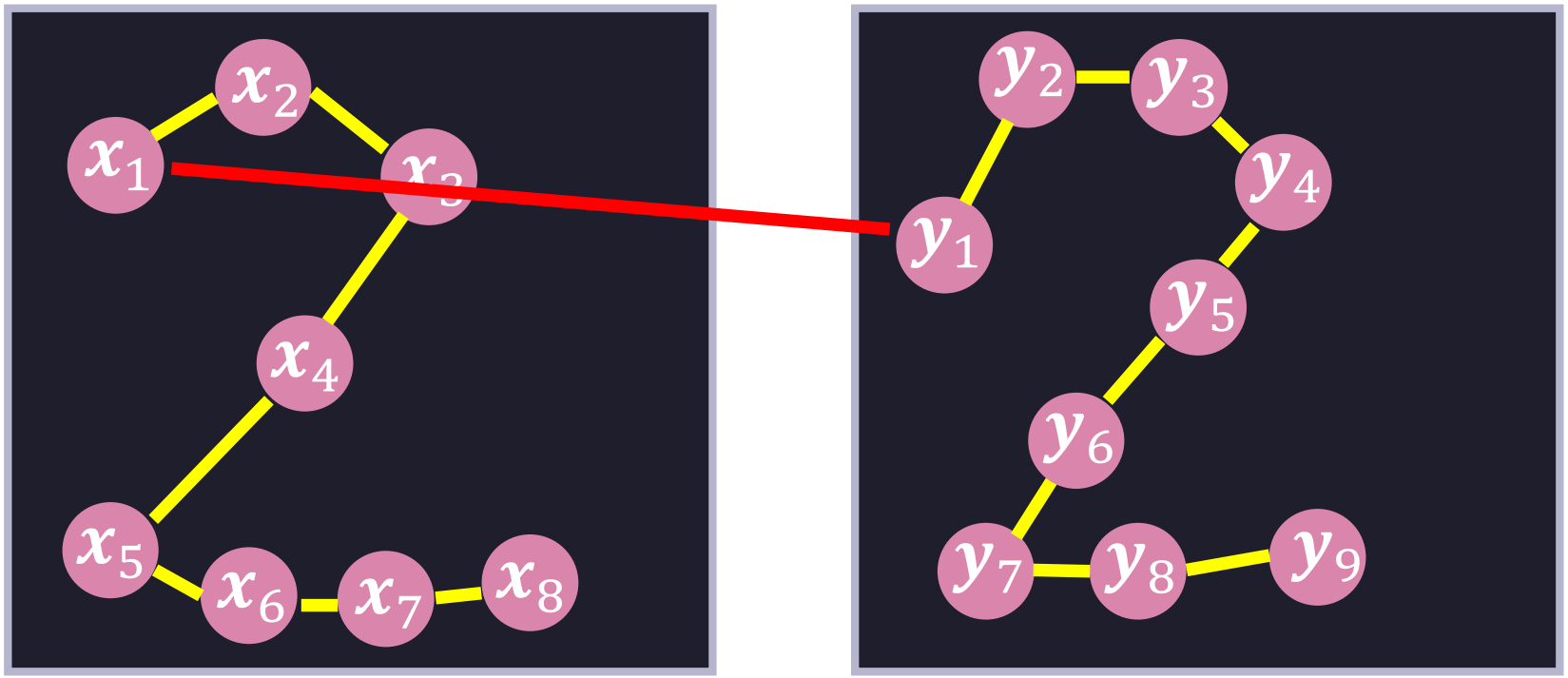


- For example, here is a "good" alignment between the two series:

$$\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$$

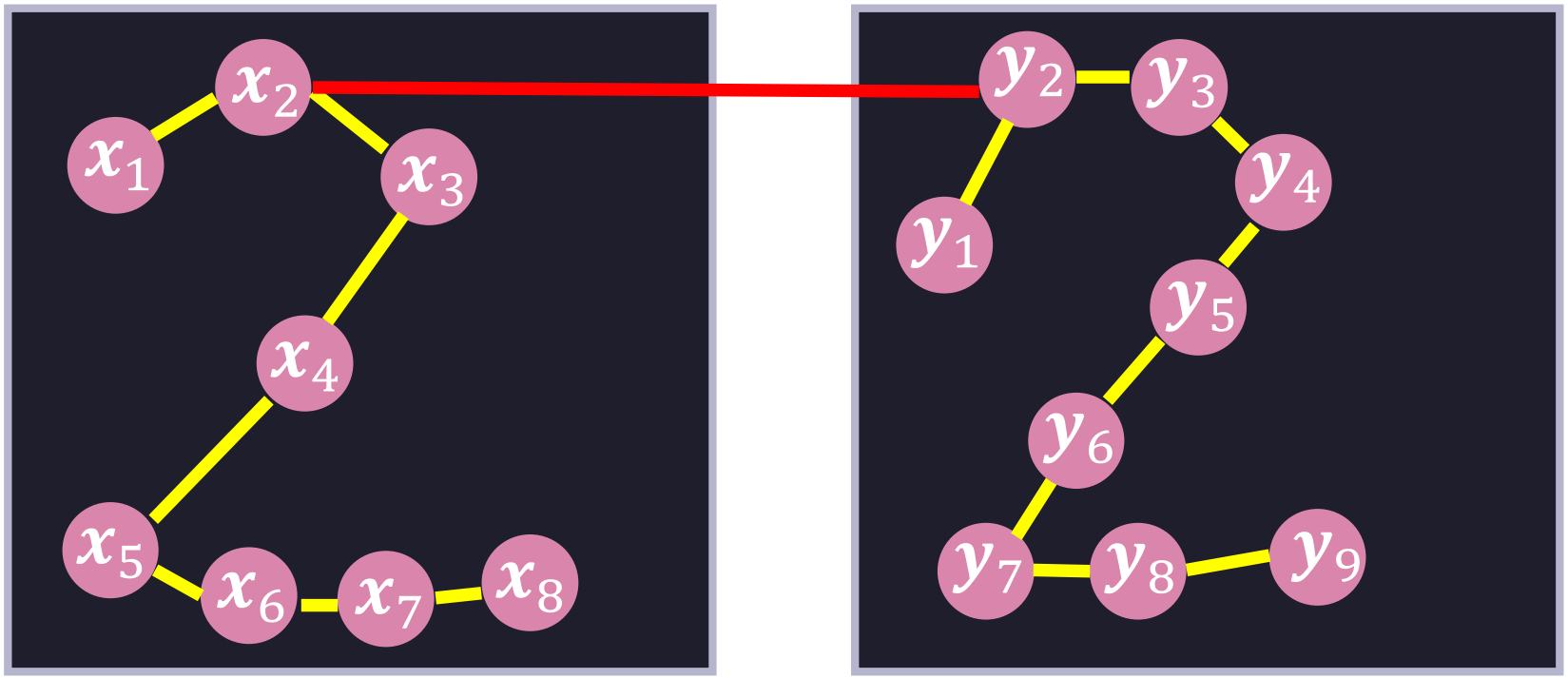
- We will discuss later how to find such an alignment automatically.

Time Series Alignment



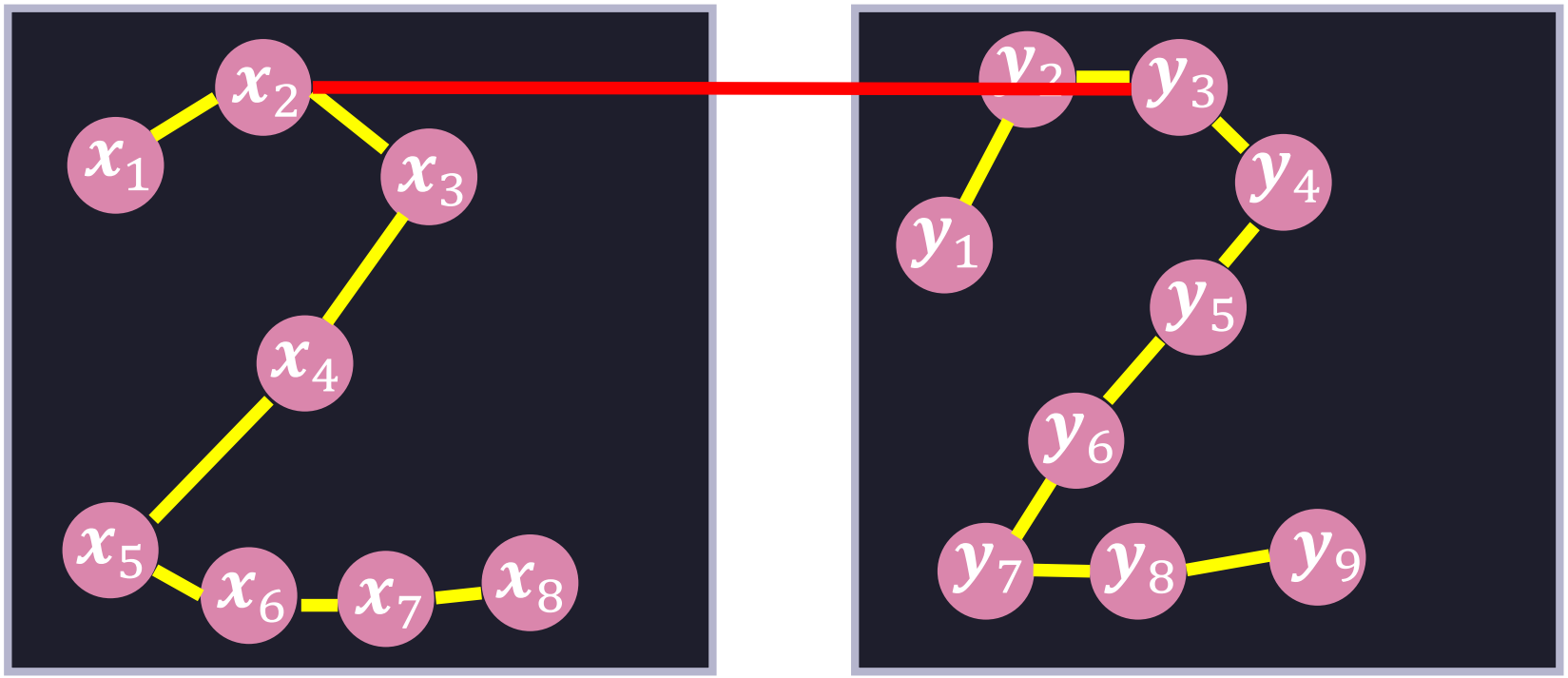
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - $\boldsymbol{x}_1$ corresponds to $\boldsymbol{y}_1$.

Time Series Alignment



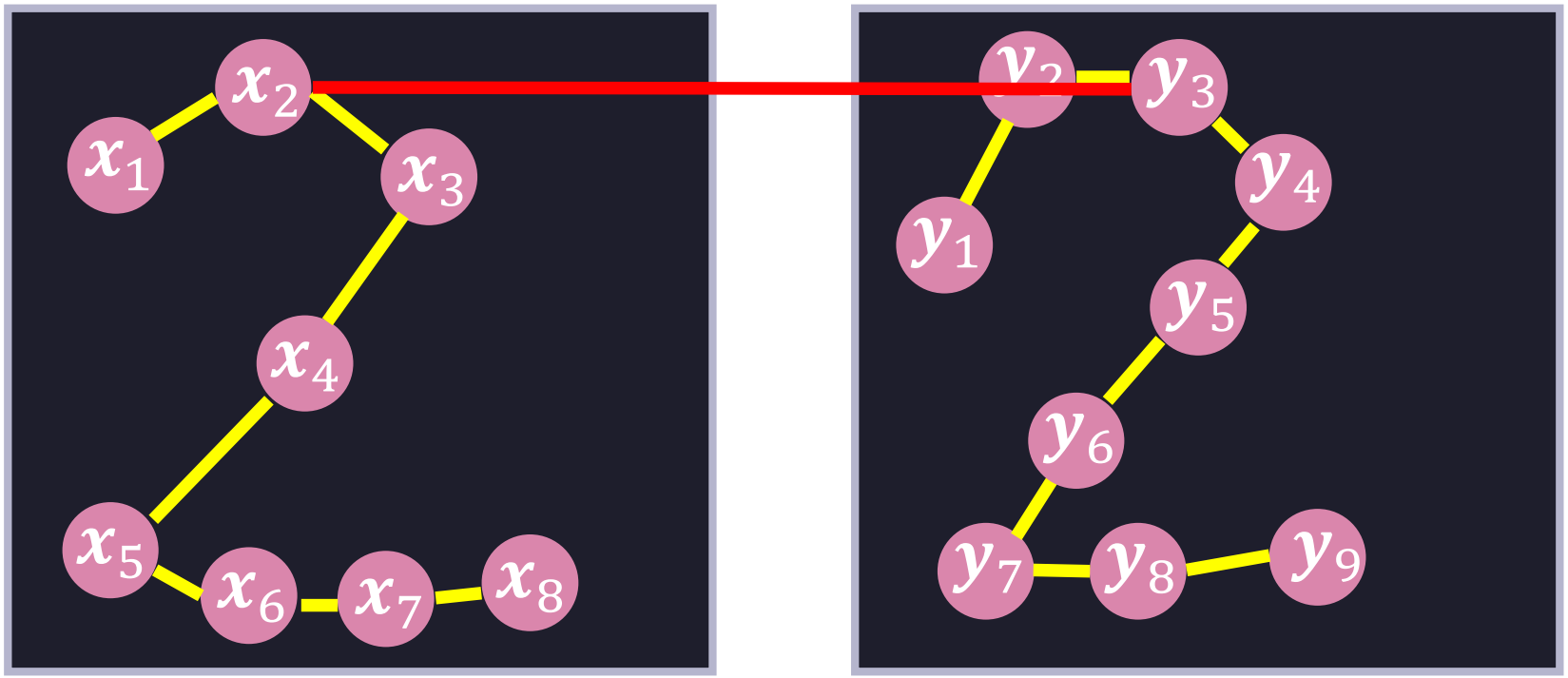
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - $\boldsymbol{x}_2$ corresponds to $\boldsymbol{y}_2$.

Time Series Alignment



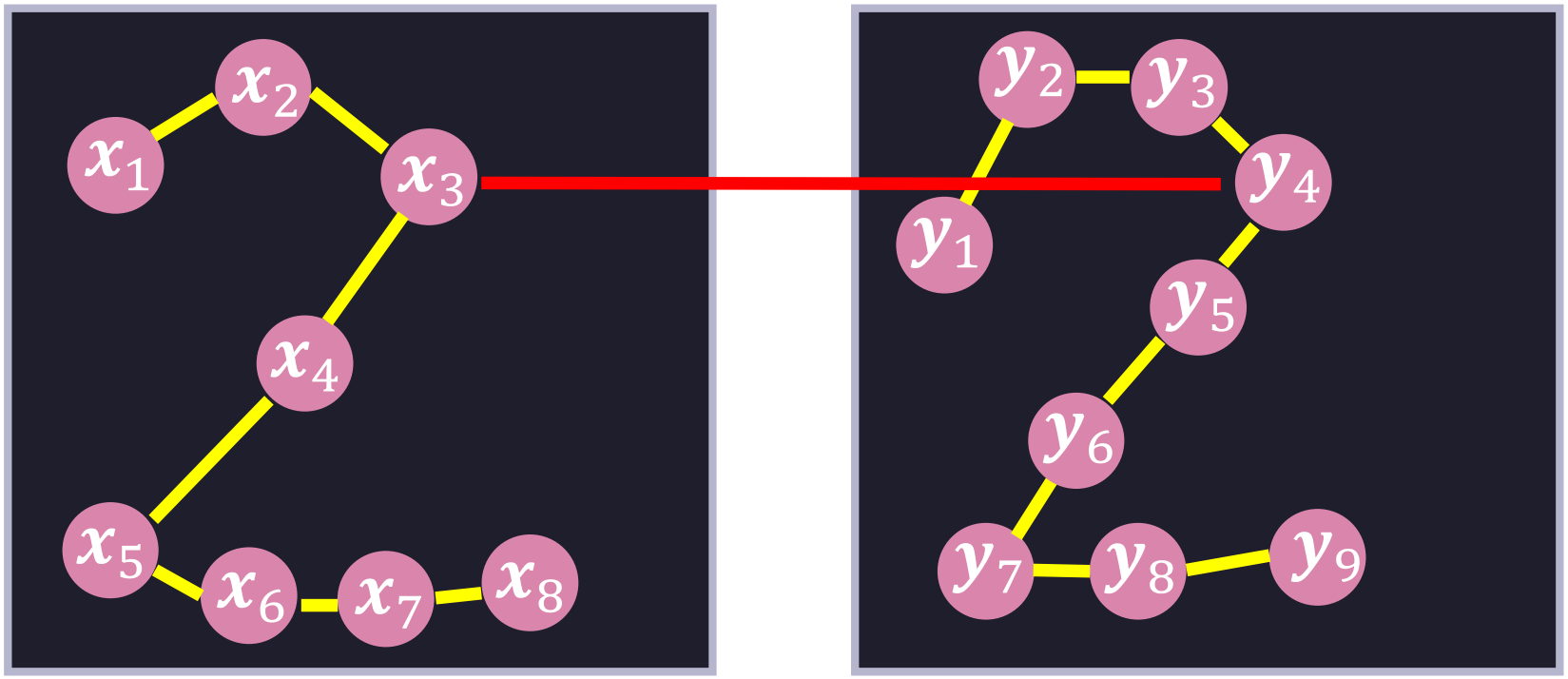
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - x_2 **also corresponds to** y_3.

Time Series Alignment



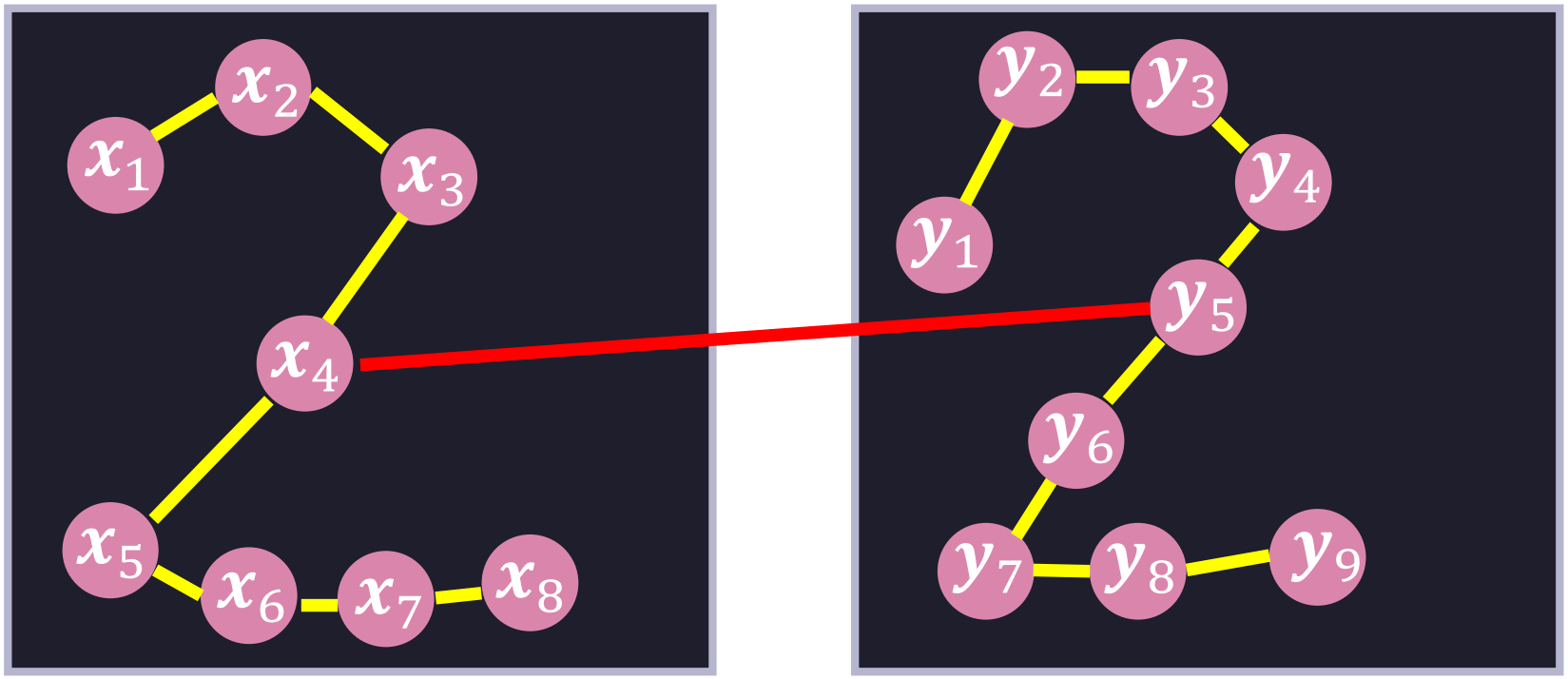
- x_2 **<u>also corresponds to</u>** y_3.
- One element from one time series can correspond to **<u>multiple consecutive elements</u>** from the other time series.
  - This captures cases where one series moves slower than the other.
  - For this segment, the first series moves faster than the second one.

Time Series Alignment



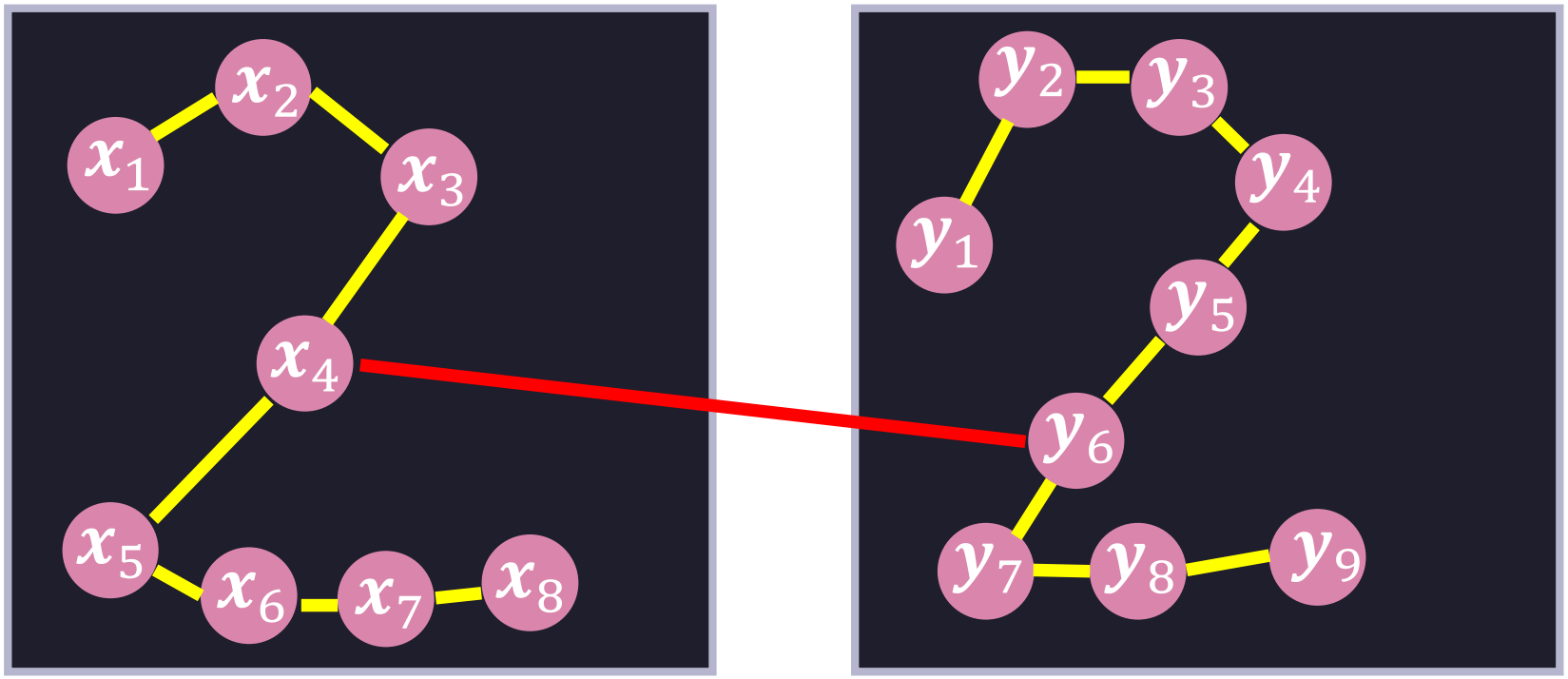
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - $\boldsymbol{x}_3$ corresponds to $\boldsymbol{y}_4$.

Time Series Alignment



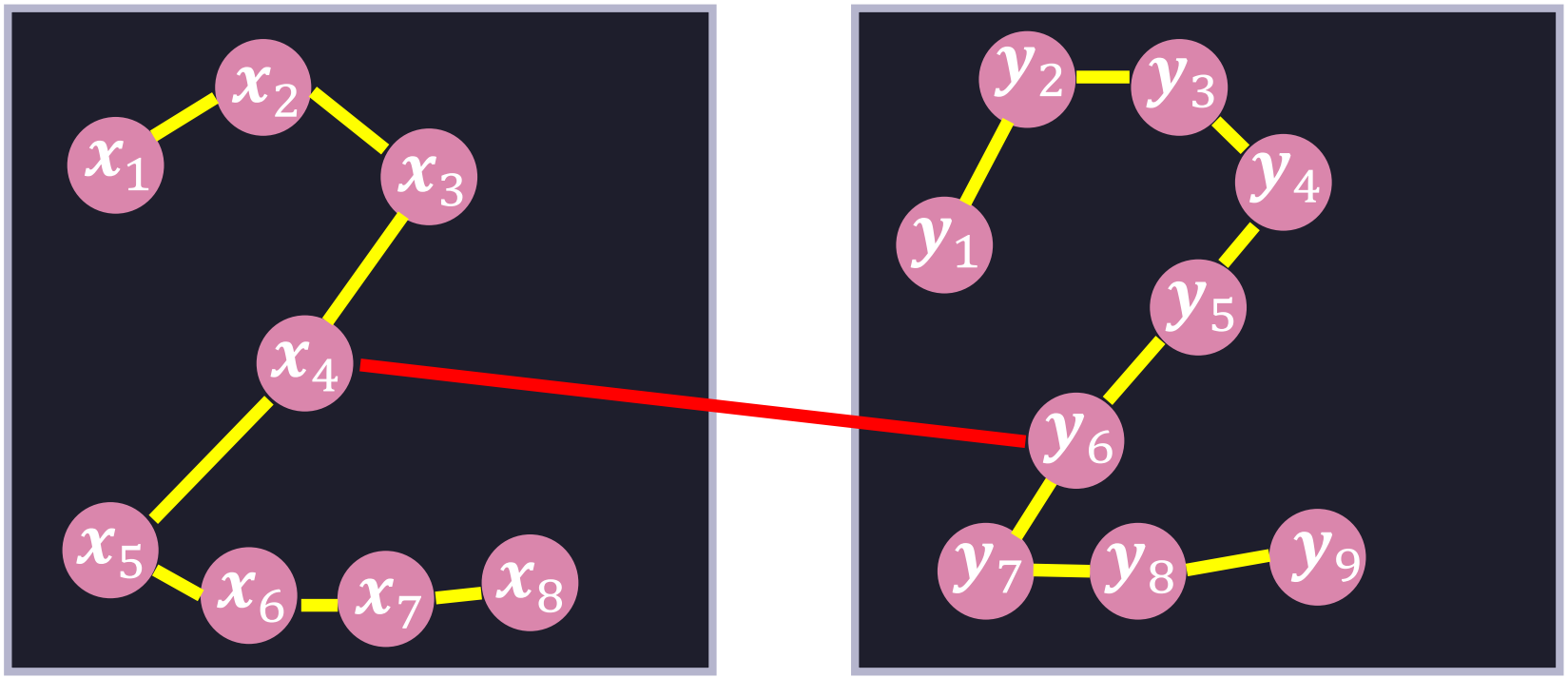
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - $\boldsymbol{x}_4$ corresponds to $\boldsymbol{y}_5$.

Time Series Alignment



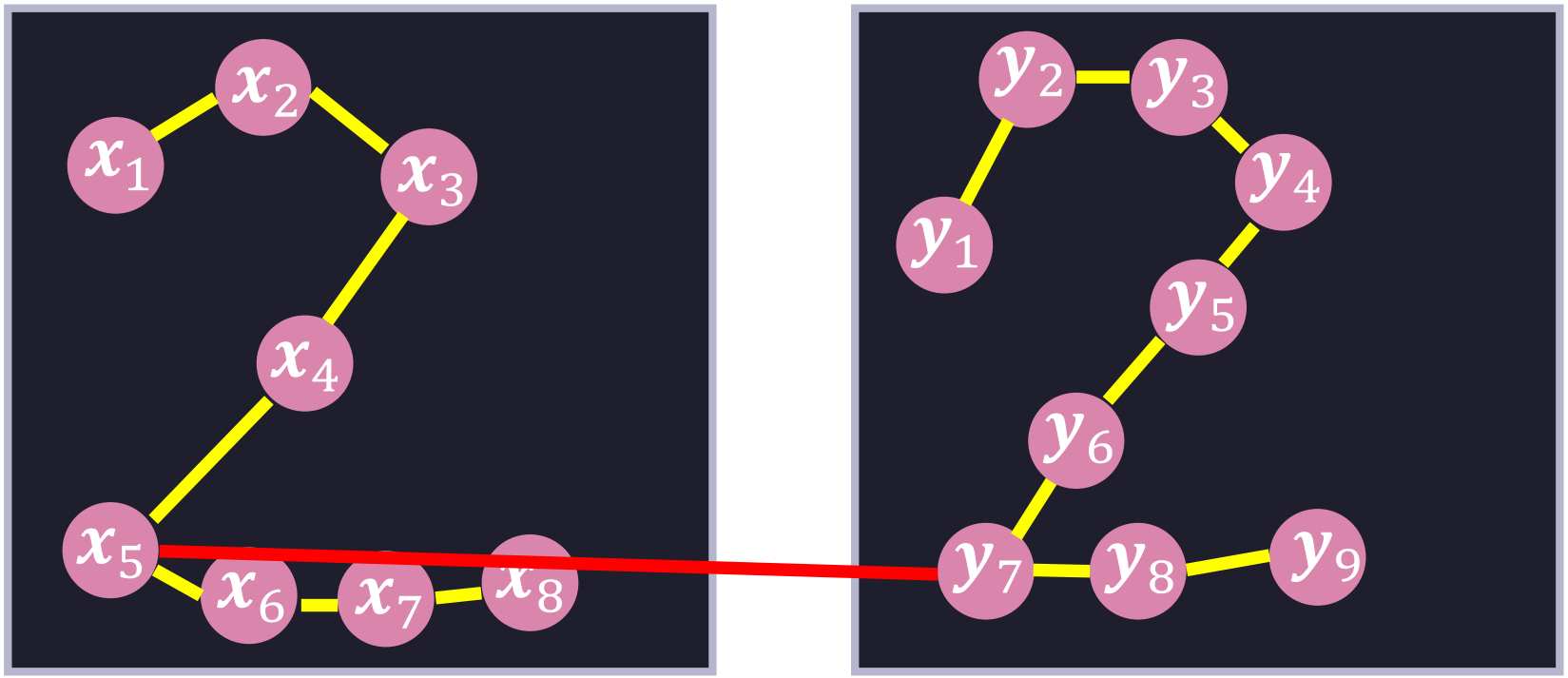
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - x_4 **also corresponds to** y_6.

Time Series Alignment



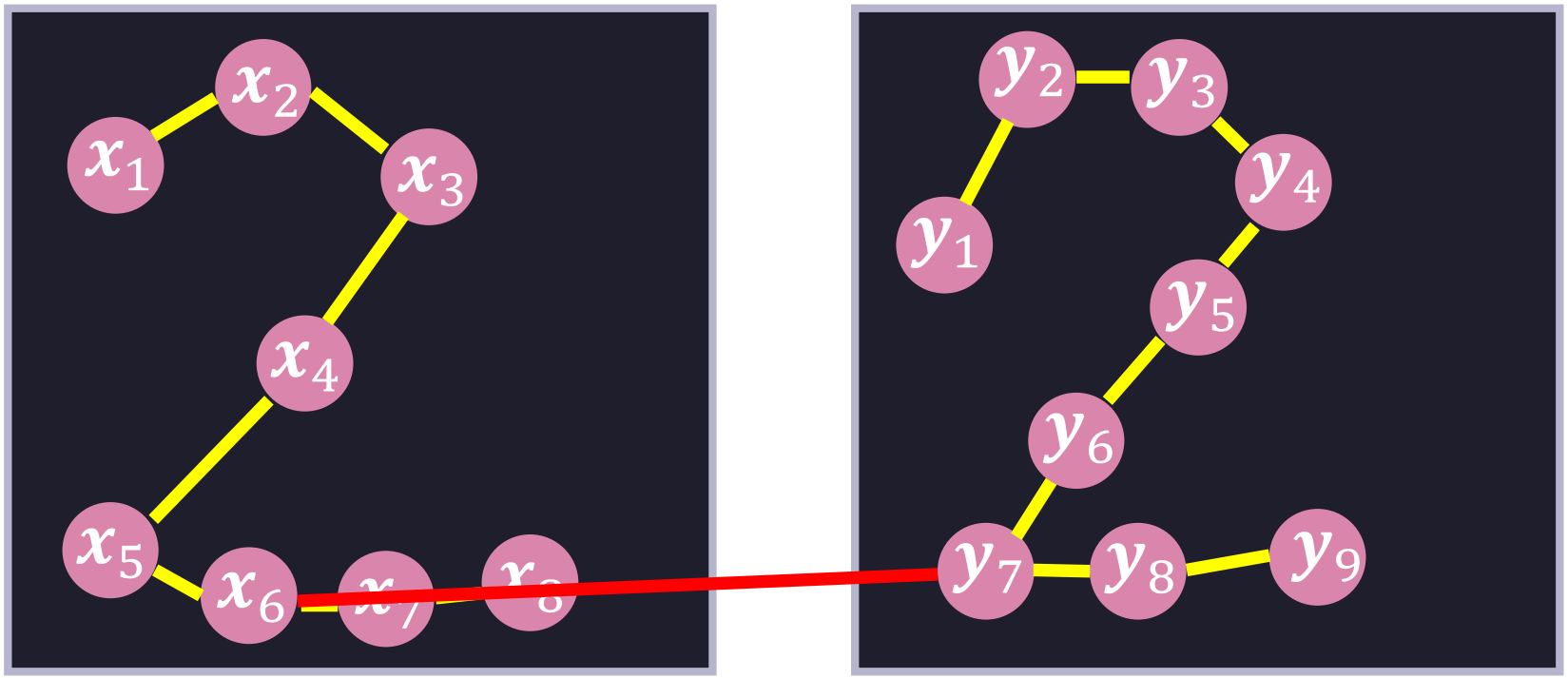
- x_4 **<u>also corresponds to</u>** y_6.
  - As before, for this segment, the first time series is moving faster than the second one.

Time Series Alignment



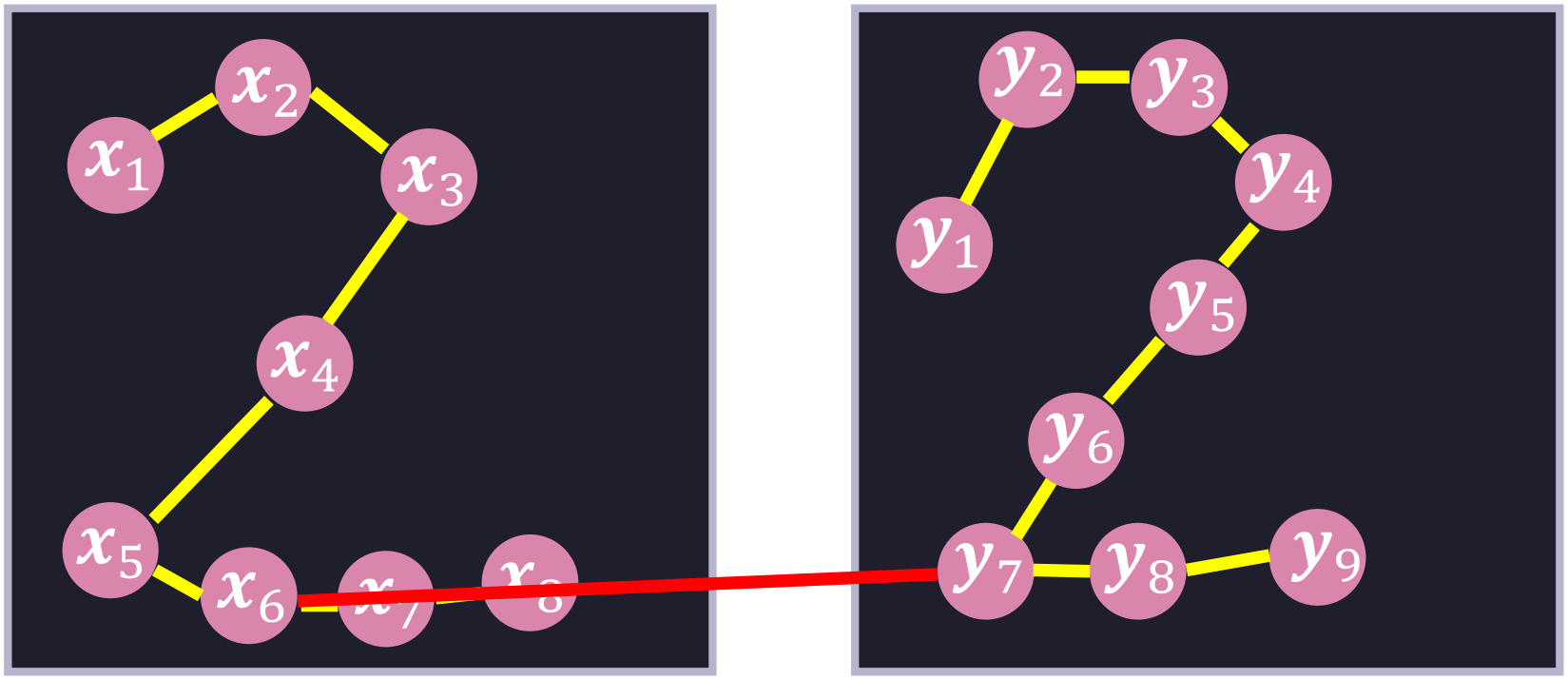
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - $\boldsymbol{x}_5$ corresponds to $\boldsymbol{y}_7$.

Time Series Alignment



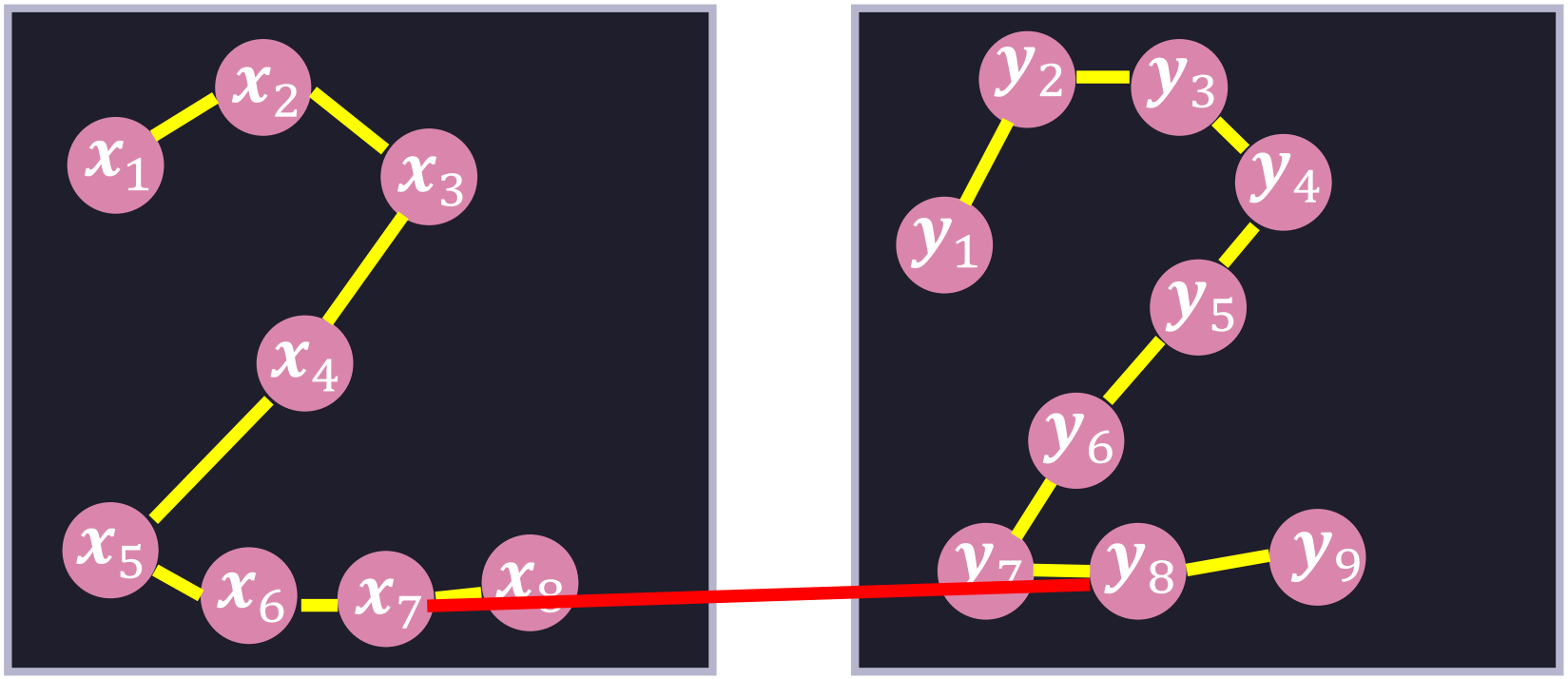
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - x_6 **also corresponds to** y_5.

Time Series Alignment



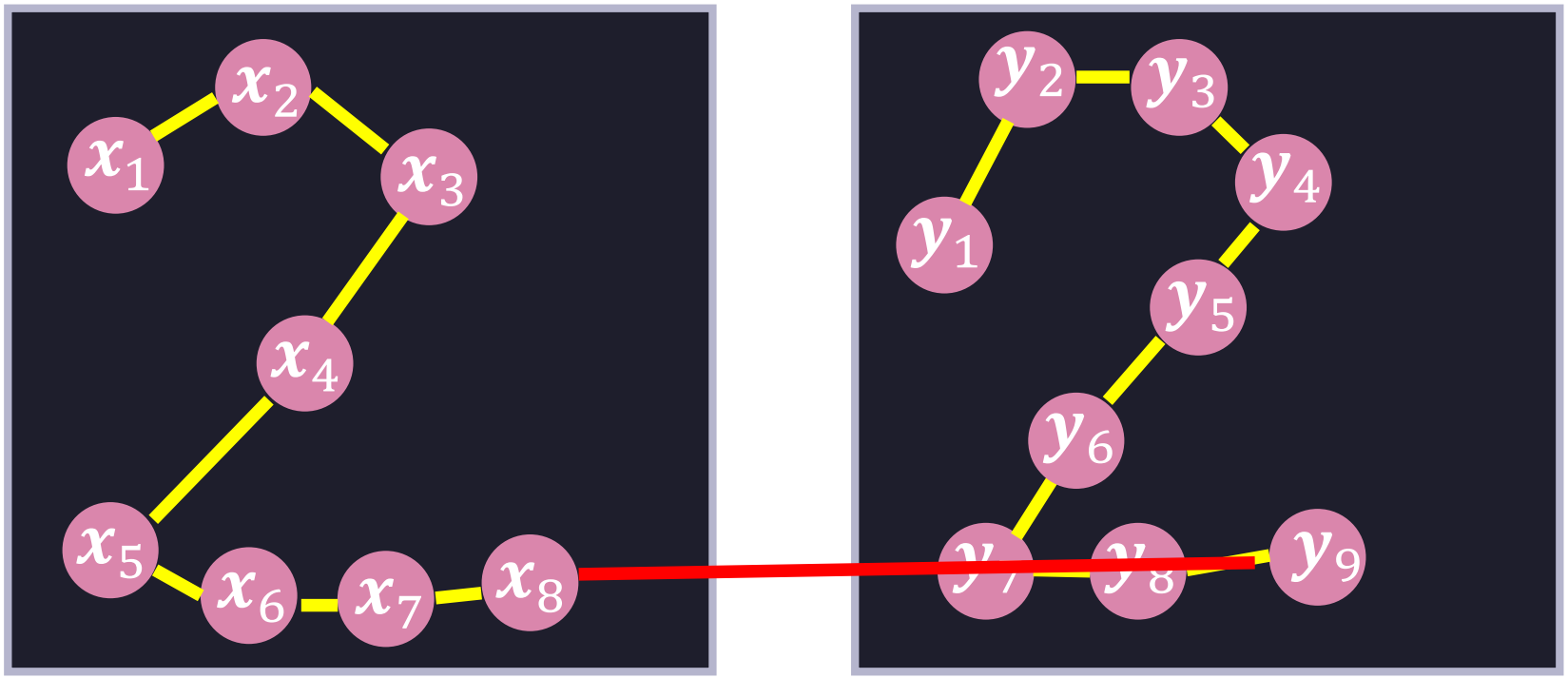
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- Here, the first time series is moving slower than the second one, and thus $\boldsymbol{x}_5$ and $\boldsymbol{x}_6$ are both matched to $\boldsymbol{y}_7$.

Time Series Alignment



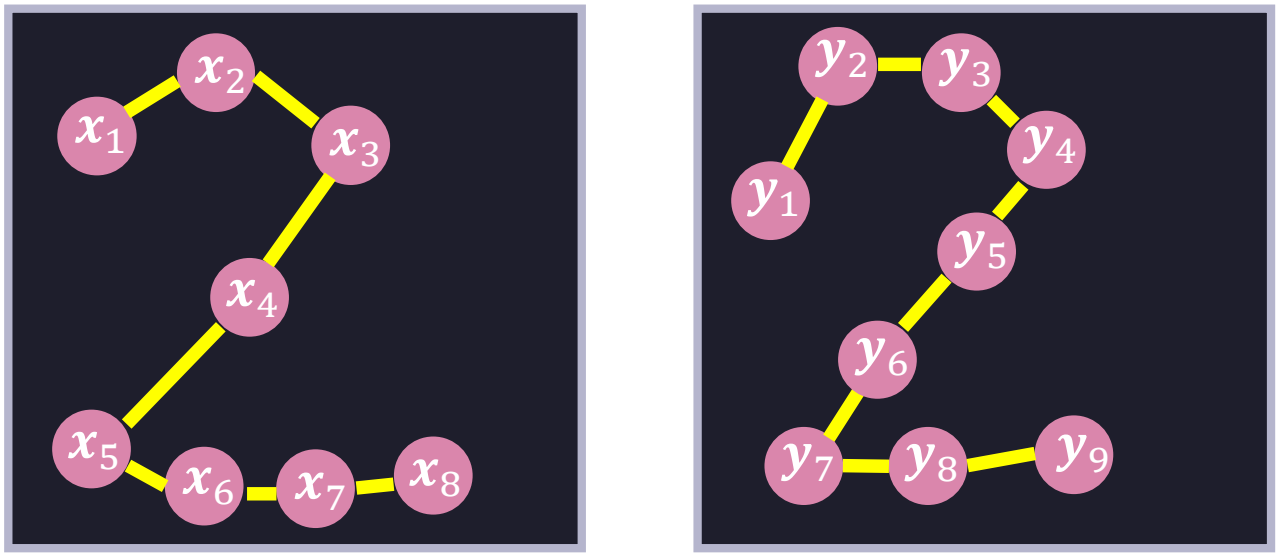
- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - $\boldsymbol{x}_7$ corresponds to $\boldsymbol{y}_8$.

Time Series Alignment



- Alignment:
  $\big((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9)\big)$
- According to this alignment:
  - $\boldsymbol{x}_8$ corresponds to $\boldsymbol{y}_9$.

The Cost of an Alignment



- An alignment $\boldsymbol{A}$ is defined as a sequence of pairs

$$\boldsymbol{A} = \Big((a_{1,1}, a_{1,2}), (a_{2,1}, a_{2,2}), \ldots, (a_{R,1}, a_{R,2}) \Big)$$

- If we are given an alignment $\boldsymbol{A}$ between two time series $\boldsymbol{X}$ and $\boldsymbol{Y}$, we can compute the cost $C_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{A})$ of that alignment as:

$$C_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{A}) = \sum_{i=1}^{R} \text{Cost}(\boldsymbol{x}_{a_{i,1}}, \boldsymbol{y}_{a_{i,2}})$$

The Cost of an Alignment

- An alignment is defined as a sequence of pairs:

$$\boldsymbol{A} = \Big(\big(a_{1,1}, a_{1,2}\big), \big(a_{2,1}, a_{2,2}\big), \dots, \big(a_{R,1}, a_{R,2}\big) \Big)$$

- If we are given an alignment $\boldsymbol{A}$ between two time series $\boldsymbol{X}$ and $\boldsymbol{Y}$, we can compute the cost $C_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{A})$ of that alignment as:

$$C_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{A}) = \sum_{i=1}^{R} \text{Cost}(\boldsymbol{x}_{a_{i,1}}, \boldsymbol{y}_{a_{i,2}})$$

- In this formula, $\text{Cost}\big(\boldsymbol{x}_{a_{i,1}}, \boldsymbol{y}_{a_{i,2}}\big)$ is a black box.
  - You can define it any way you like.
  - For example, if $\boldsymbol{x}_{a_{i,1}}$ and $\boldsymbol{y}_{a_{i,2}}$ are vectors, $\text{Cost}(\boldsymbol{x}_{a_{i,1}}, \boldsymbol{y}_{a_{i,2}})$ can be the Euclidean distance between $\boldsymbol{x}_{a_{i,1}}$ and $\boldsymbol{y}_{a_{i,2}}$.

The Cost of an Alignment

- An alignment is defined as a sequence of pairs:

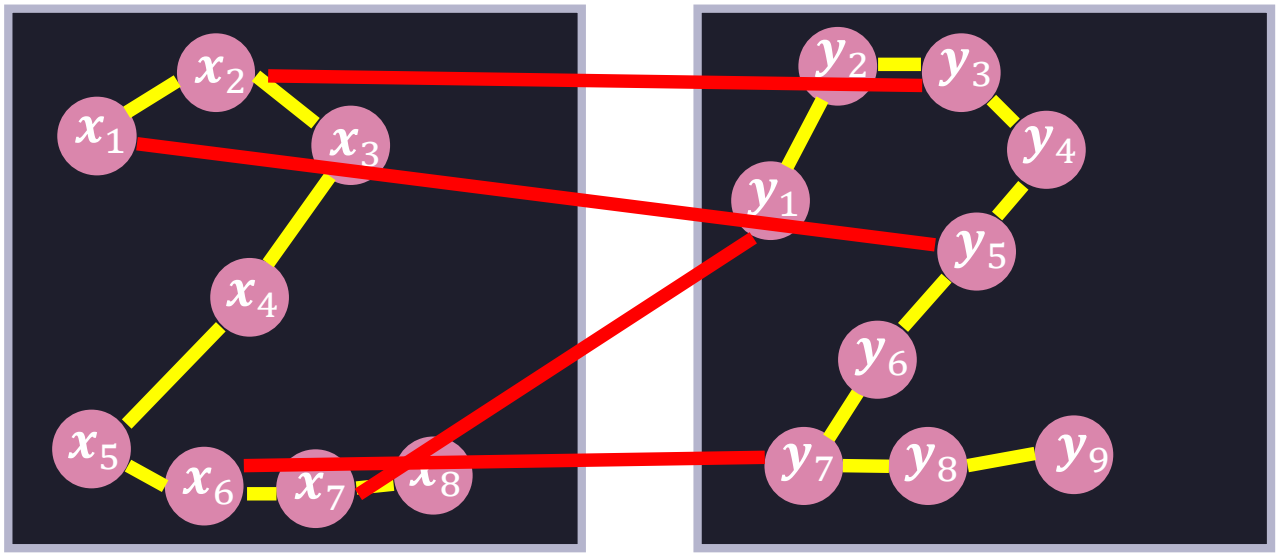
$$\boldsymbol{A} = \Big(\big(a_{1,1}, a_{1,2}\big), \big(a_{2,1}, a_{2,2}\big), \ldots, \big(a_{R,1}, a_{R,2}\big) \Big)$$

- If we are given an alignment $\boldsymbol{A}$ between two time series $\boldsymbol{X}$ and $\boldsymbol{Y}$, we can compute the cost $C_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{A})$ of that alignment as:

$$C_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{A}) = \sum_{i=1}^{R} \text{Cost}(\boldsymbol{x}_{a_{i,1}}, \boldsymbol{y}_{a_{i,2}})$$

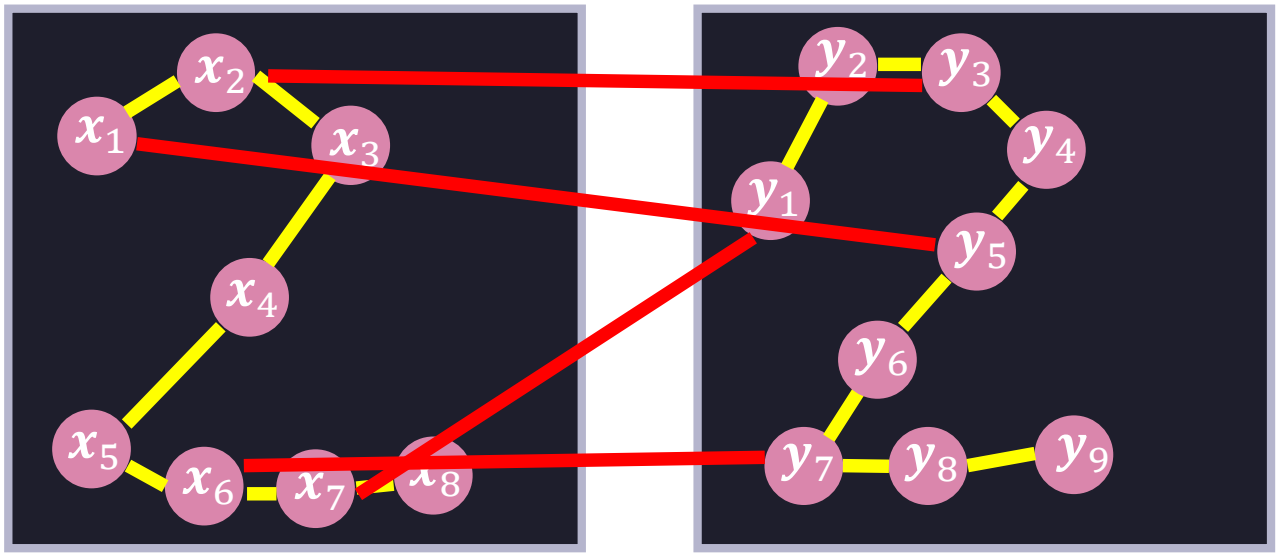
- Notation $C_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{A})$ indicates that the cost of alignment $\boldsymbol{A}$depends on the specific pair of time series that we are aligning.
  - The cost of an alignment $\boldsymbol{A}$ depends on the alignment itself, as well as the two time series $\boldsymbol{X}$ and $\boldsymbol{Y}$ that we are aligning.

Rules of Alignment



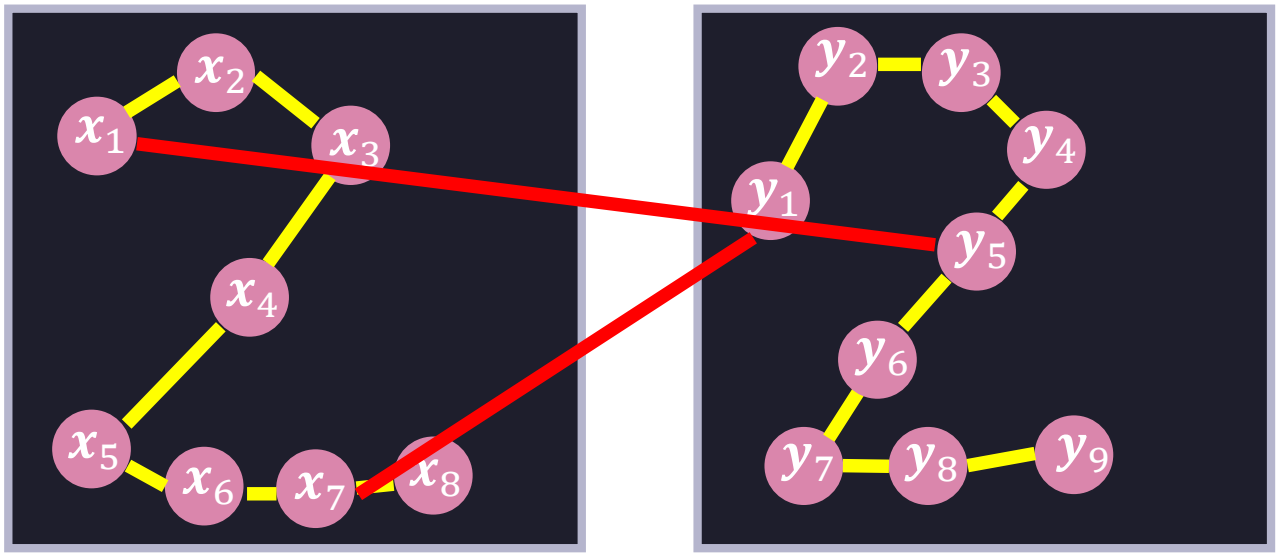
- Should alignment ((1, 5), (2, 3), (6, 7), (7, 1)) be legal?
- It always depends on what makes sense for your data.
- Typically, for time series, alignments have to obey certain rules, that this alignment violates.

Rules of Alignment



- We will require that legal alignments obey three rules:
  - Rule 1: Boundary Conditions.
  - Rule 2: Monotonicity.
  - Rule 3: Continuity
- The next slides define these rules.

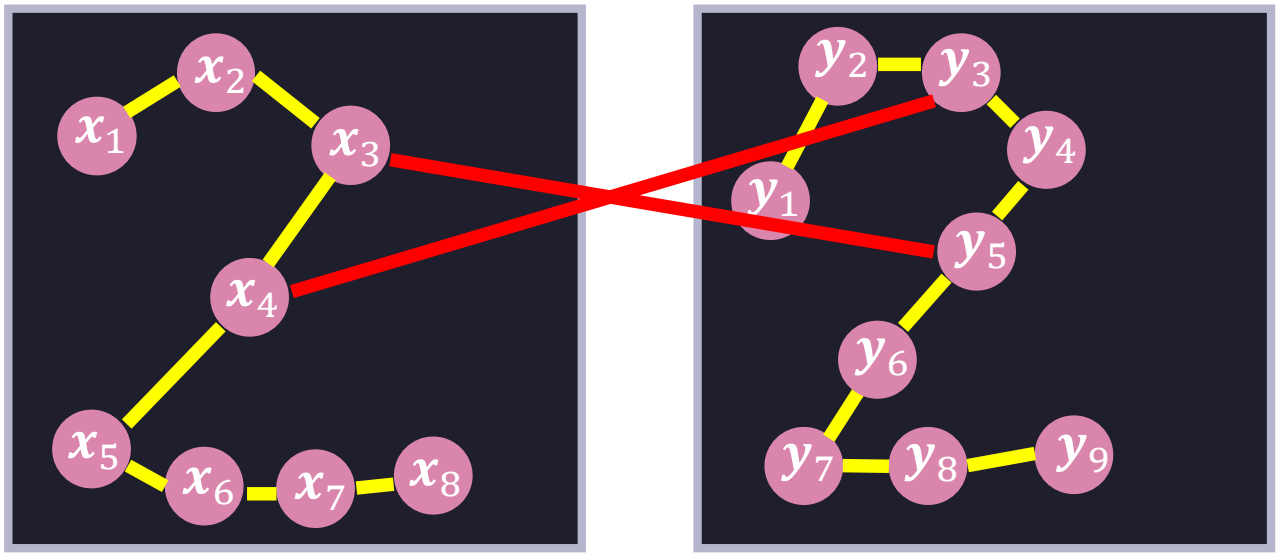
Rule 1: Boundary Conditions



- Illegal alignment (violating boundary conditions):
  - $(\,(1,5),\ldots,(7,1))$.
  - $\big((s_1,t_1),(s_2,t_2),\ldots,(s_R,t_R)\big)$
- Alignment rule #1 (boundary conditions):
  - $s_1 = 1,\ t_1 = 1.$
  - $s_R = M =$ length of first time series
  - $t_R = N =$ length of second time series

first elements match
last elements match

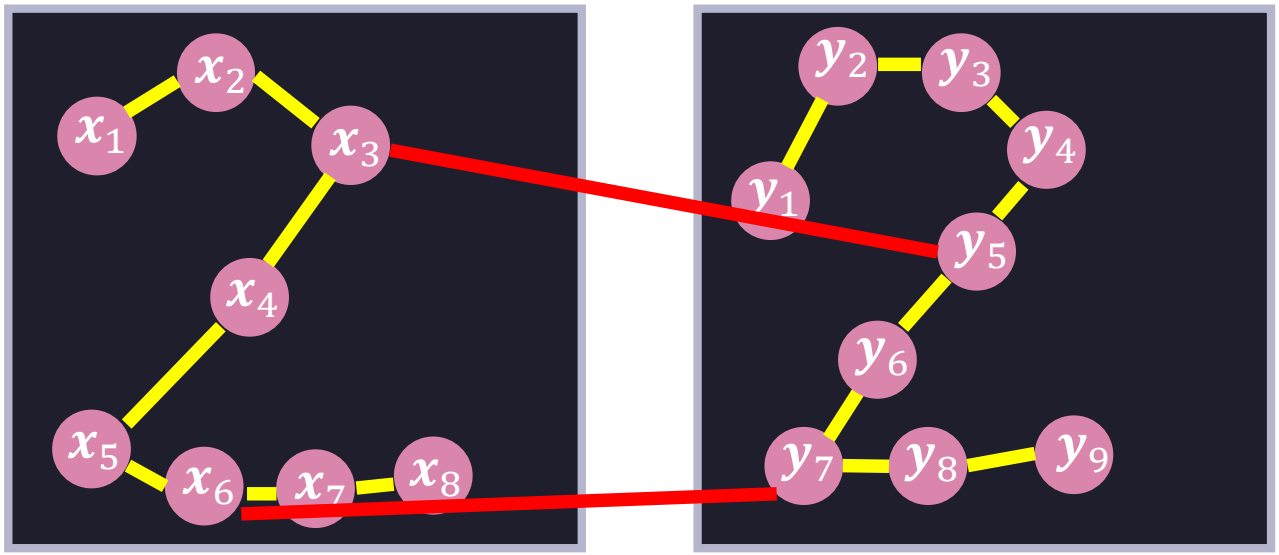
Rule 2: Monotonicity



- Illegal alignment (violating monotonicity):
  - $(\dots, (3,5), (4,3), \dots)$.
  - $\big((s_1, t_1), (s_2, t_2), \dots, (s_R, t_R)\big)$
- Alignment rule #2: sequences $s_1, \dots, s_R$ and $t_1, \dots, t_R$ are monotonically increasing.
  - $(s_{i+1} - s_i) \geq 0$
  - $(t_{i+1} - t_i) \geq 0$

The alignment cannot go backwards.

Rule 3: Continuity



- Illegal alignment (violating continuity):
  - $(\dots, (3,5), (6,7), \dots)$.
  - $\big((s_1, t_1), (s_2, t_2), \dots, (s_R, t_R)\big)$
- Alignment rule #3: sequences $s_1, \dots, s_R$ and $t_1, \dots, t_R$ cannot increase by more than one at each step.
  - $(s_{i+1} - s_i) \leq 1$
  - $(t_{i+1} - t_i) \leq 1$

The alignment cannot skip elements.

Visualizing a Warping Path

- **Warping path** is an alternative term for an alignment

Visualizing a Warping Path

$$\boldsymbol{A} = ((1,1),(2,2),(2,3),(3,4),(4,5),(4,6),(5,7),(6,7),(7,8),(8,9))$$

- We can visualize a warping path $\boldsymbol{A}$ in 2D as follows:
- An element of $\boldsymbol{A}$ corresponds to a colored cell in the 2D table.

	$\boldsymbol{y}_1$	$\boldsymbol{y}_2$	$\boldsymbol{y}_3$	$\boldsymbol{y}_4$	$\boldsymbol{y}_5$	$\boldsymbol{y}_6$	$\boldsymbol{y}_7$	$\boldsymbol{y}_8$	$\boldsymbol{y}_9$
$\boldsymbol{x}_1$	■								
$\boldsymbol{x}_2$		■	■						
$\boldsymbol{x}_3$				■					
$\boldsymbol{x}_4$					■	■			
$\boldsymbol{x}_5$							■		
$\boldsymbol{x}_6$							■		
$\boldsymbol{x}_7$								■	
$\boldsymbol{x}_8$									■

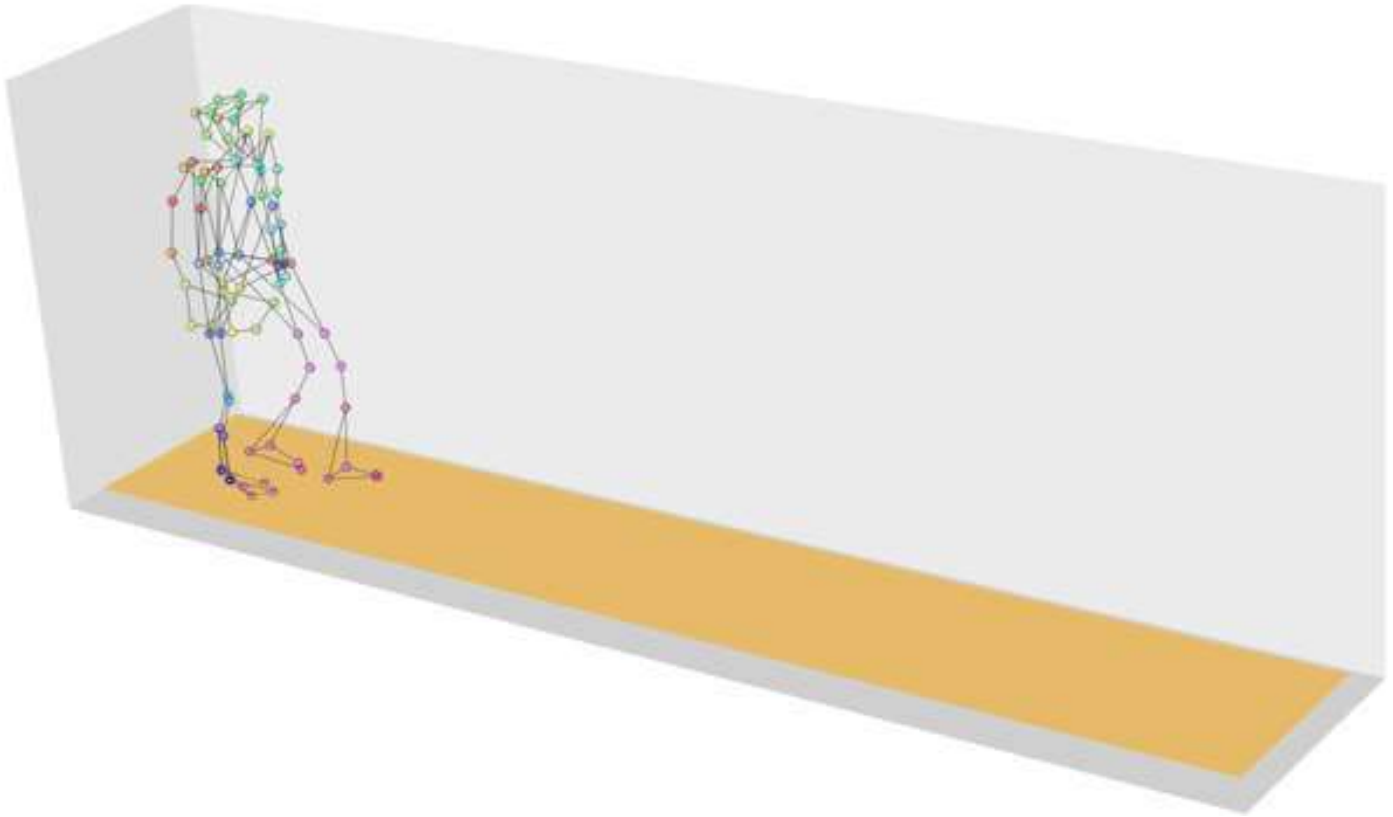
Dynamic Time Warping

- **Dynamic Time Warping** (DTW) is a distance measure between time series.
- The DTW distance is the cost of the **optimal legal alignment** between the two time series.
- The alignment must be legal: it must obey the three rules of alignments.
  - Boundary conditions.
  - Monotonicity.
  - Continuity.
- The alignment must minimize $C_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{A}) = \sum_{i=1}^{R} \text{Cost}(\boldsymbol{x}_{a_{i,1}}, \boldsymbol{y}_{a_{i,2}})$
- So:

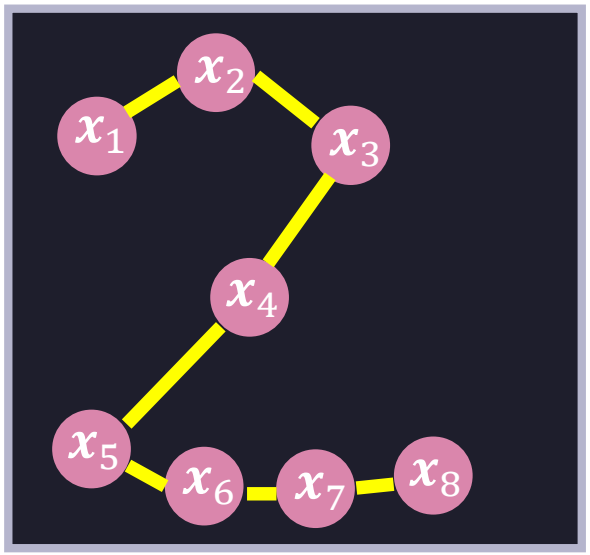
$$\text{DTW}(\boldsymbol{X}, \boldsymbol{Y}) = \min_{\boldsymbol{A}} C_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{A})$$

where $\boldsymbol{A}$ ranges over all legal alignments between $\boldsymbol{X}$ and $\boldsymbol{Y}$.

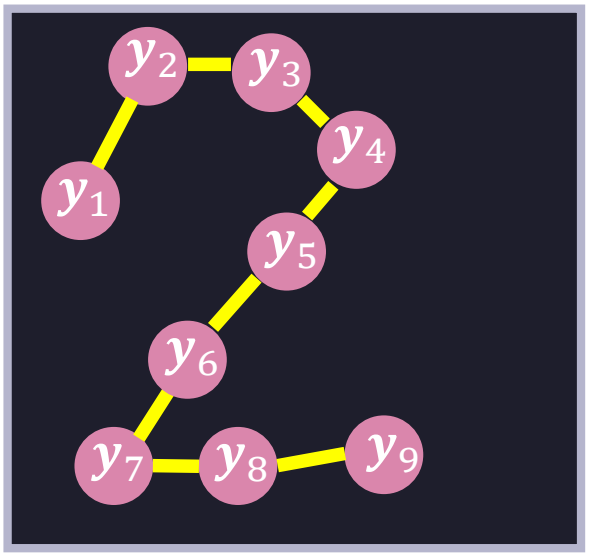
Dynamic Time Warping



Finding the Optimal Alignment



$\boldsymbol{X}$



$\boldsymbol{Y}$

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
  - $\boldsymbol{X}$ is a time series of length M.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
  - $\boldsymbol{Y}$ is a time series of length N.
- Each $\boldsymbol{x}_i$ and each $\boldsymbol{y}_i$ are feature vectors.

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- Dynamic programming strategy:
  - Break problem up into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- We need to solve every problem(i, j) for every i and j.
- Then, the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$ is the solution to problem(M, N).

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- How do we start computing the optimal alignment?

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- Solve problem$(1, 1)$:
  - How is problem$(1, 1)$ defined?

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- Solve problem$(1, 1)$:
  - Find optimal alignment between $(\boldsymbol{x}_1)$ and $(\boldsymbol{y}_1)$.
  - What is the optimal alignment between $(\boldsymbol{x}_1)$ and $(\boldsymbol{y}_1)$?

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- Solve problem$(1, 1)$:
  - Find optimal alignment between $(\boldsymbol{x}_1)$ and $(\boldsymbol{y}_1)$.
  - The optimal alignment is the **only legal alignment**: $((1, 1))$.

Alignment for Problem $(1, 1)$

$$\boldsymbol{A} = ((1, 1)\ )$$

	$\boldsymbol{y}_1$	$\boldsymbol{y}_2$	$\boldsymbol{y}_3$	$\boldsymbol{y}_4$	$\boldsymbol{y}_5$	$\boldsymbol{y}_6$	$\boldsymbol{y}_7$	$\boldsymbol{y}_8$	$\boldsymbol{y}_9$
$\boldsymbol{x}_1$									
$\boldsymbol{x}_2$									
$\boldsymbol{x}_3$									
$\boldsymbol{x}_4$									
$\boldsymbol{x}_5$									
$\boldsymbol{x}_6$									
$\boldsymbol{x}_7$									
$\boldsymbol{x}_8$									

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- Solve problem$(1, j)$:
  - How is problem$(1, j)$ defined?

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- Solve problem$(1, j)$:
  - Find optimal alignment between $(\boldsymbol{x}_1)$ and$(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
  - What is the optimal alignment between $(\boldsymbol{x}_1)$ and $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_j)$?

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- Solve problem$(1, j)$:
  - Find optimal alignment between $(\boldsymbol{x}_1)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
  - Optimal alignment: $((1,1), (1,2), \ldots, (1,j))$.
  - Here, $\boldsymbol{x}_1$ is matched with each of the first j elements of $\boldsymbol{Y}$.

Alignment for Problem $(1, 5)$

$$\boldsymbol{A} = ((1,1),(1,2),(1,3),(1,4),(1,5))$$

	$\boldsymbol{y}_1$	$\boldsymbol{y}_2$	$\boldsymbol{y}_3$	$\boldsymbol{y}_4$	$\boldsymbol{y}_5$	$\boldsymbol{y}_6$	$\boldsymbol{y}_7$	$\boldsymbol{y}_8$	$\boldsymbol{y}_9$
$\boldsymbol{x}_1$									
$\boldsymbol{x}_2$									
$\boldsymbol{x}_3$									
$\boldsymbol{x}_4$									
$\boldsymbol{x}_5$									
$\boldsymbol{x}_6$									
$\boldsymbol{x}_7$									
$\boldsymbol{x}_8$									

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- Solve problem$(i, 1)$:
  - How is problem$(i, 1)$ defined?

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- Solve problem$(i, 1)$:
  - Find optimal alignment between $(\boldsymbol{x}_1, \ldots \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1)$.
  - What is the optimal alignment between $(\boldsymbol{x}_1, \ldots \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1)$?

Finding the Optimal Alignment

- $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_M)$.
- $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots \boldsymbol{y}_N)$.
- We want to find the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- We break up the problem into a 2D array of smaller, interrelated problems (i, j), where $1 \leq i \leq M$, $1 \leq j \leq N$.
- Problem(i, j):
  - find optimal alignment between $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1, \ldots \boldsymbol{y}_j)$.
- Solve problem$(i, 1)$:
  - Find optimal alignment between $(\boldsymbol{x}_1, \ldots \boldsymbol{x}_i)$ and $(\boldsymbol{y}_1)$.
  - Optimal alignment: $((1, 1), (2, 1), \ldots, (i, 1))$.
  - Here, $\boldsymbol{y}_1$ is matched with each of the first i elements of $\boldsymbol{X}$.

Cost vs. Alignment

- Note: there are two related but different concepts here: **optimal alignment** and **optimal cost**.
  - We may want to find the optimal **alignment** between two time series, to visualize how elements of those two time series correspond to each other.
  - We may want to compute the DTW distance between two time series $\boldsymbol{X}$ and $\boldsymbol{Y}$, which means finding the **cost** of the optimal alignment between $\boldsymbol{X}$ and $\boldsymbol{Y}$. We can use such DTW distances, for example, for nearest neighbor classification.
- The pseudocode in the previous slide computes the **cost** of the optimal alignment, **without** explicitly outputing the optimal alignment itself.

DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{N+1 \times M+1}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$
- Calculate cost matrix:

  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$

DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

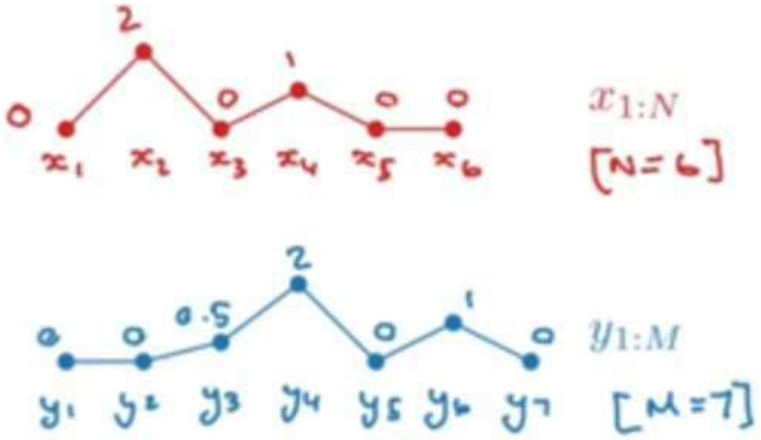
  for $j = 1$ to M: $D_{0,j} = \infty$
- Calculate cost matrix:

  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

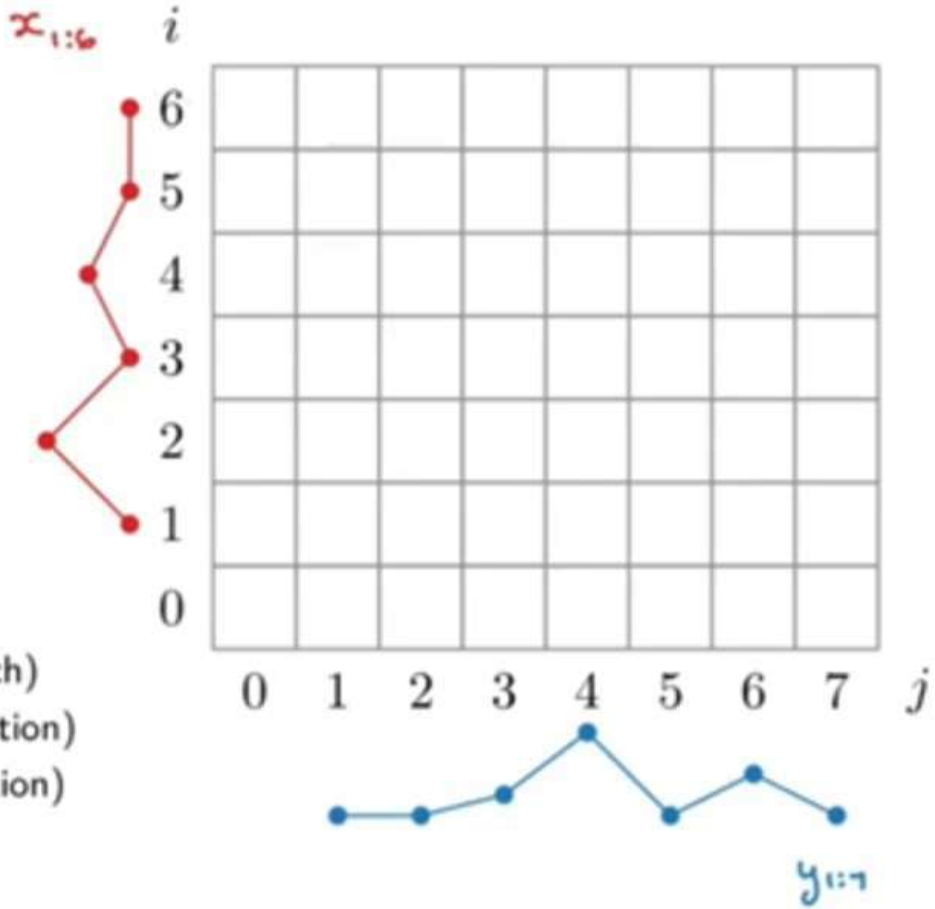
  for $j = 1$ to M: $D_{0,j} = \infty$
- Calculate cost matrix:

  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$

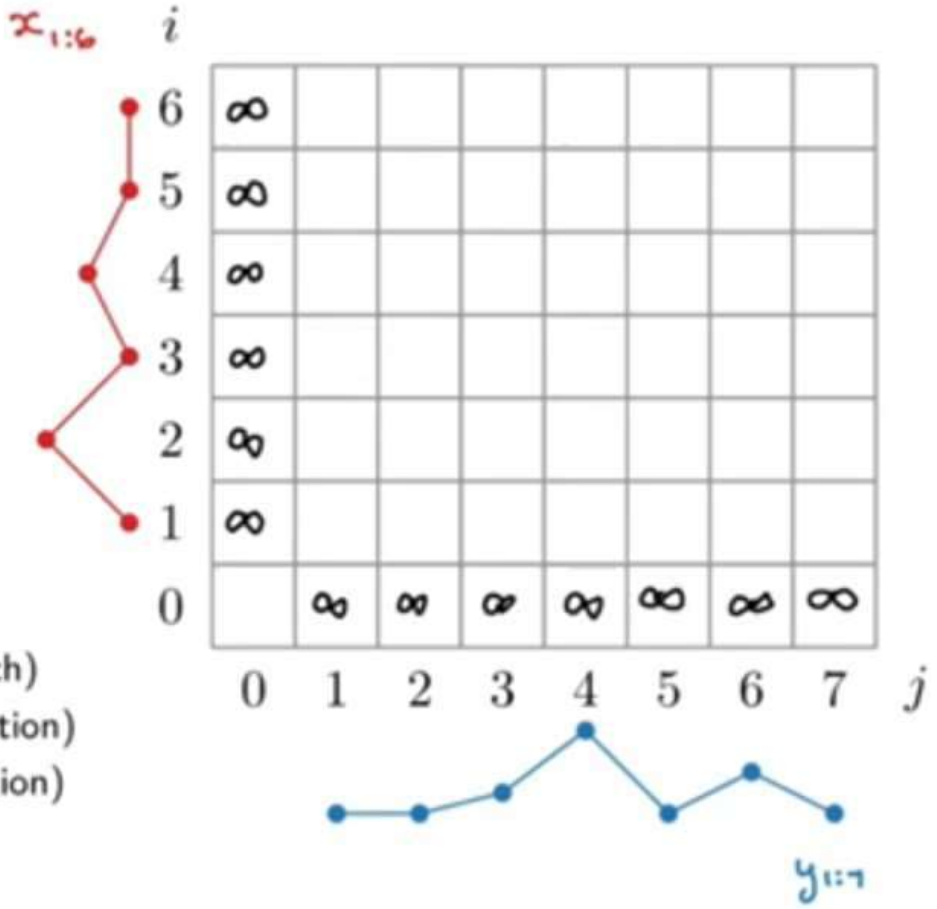


DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:
  for $i = 1$ to N: $D_{i,0} = \infty$
  for $j = 1$ to M: $D_{0,j} = \infty$
- Calculate cost matrix:
  for $i = 1$ to N:
    for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

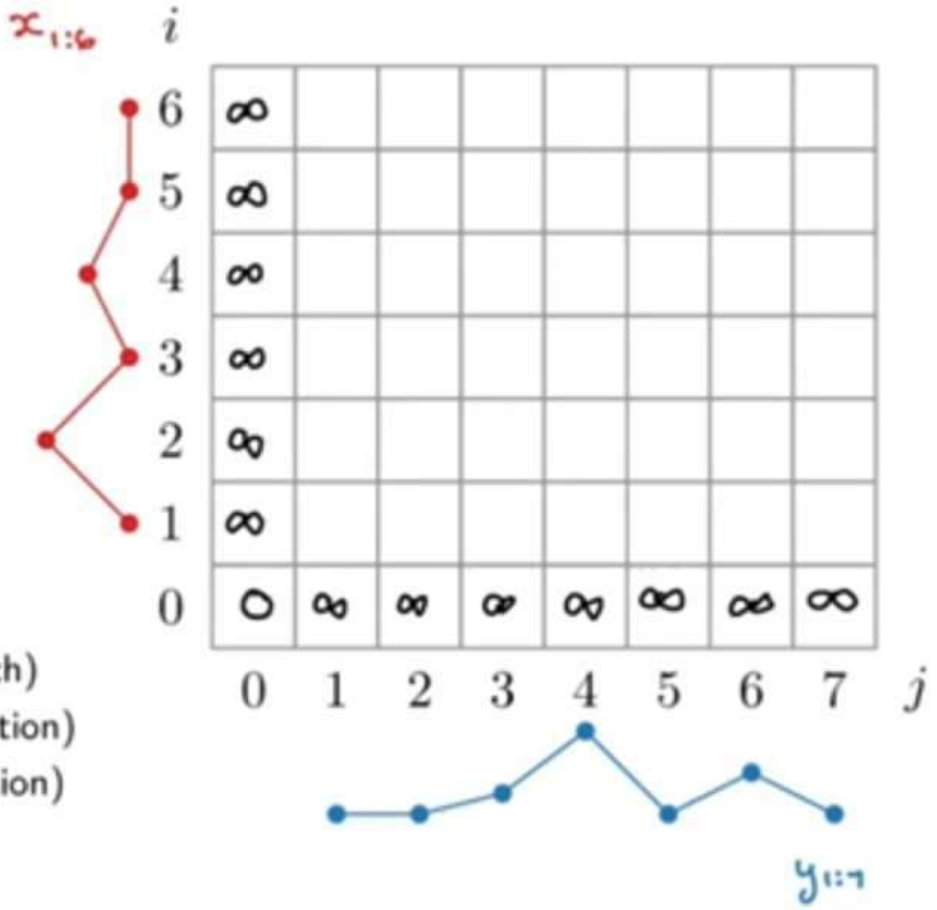
  $D_{0,0} = 0$
- Calculate cost matrix:

  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

  $D_{0,0} = 0$
- Calculate cost matrix:

  for $i = 1$ to N:

  $i = 1$

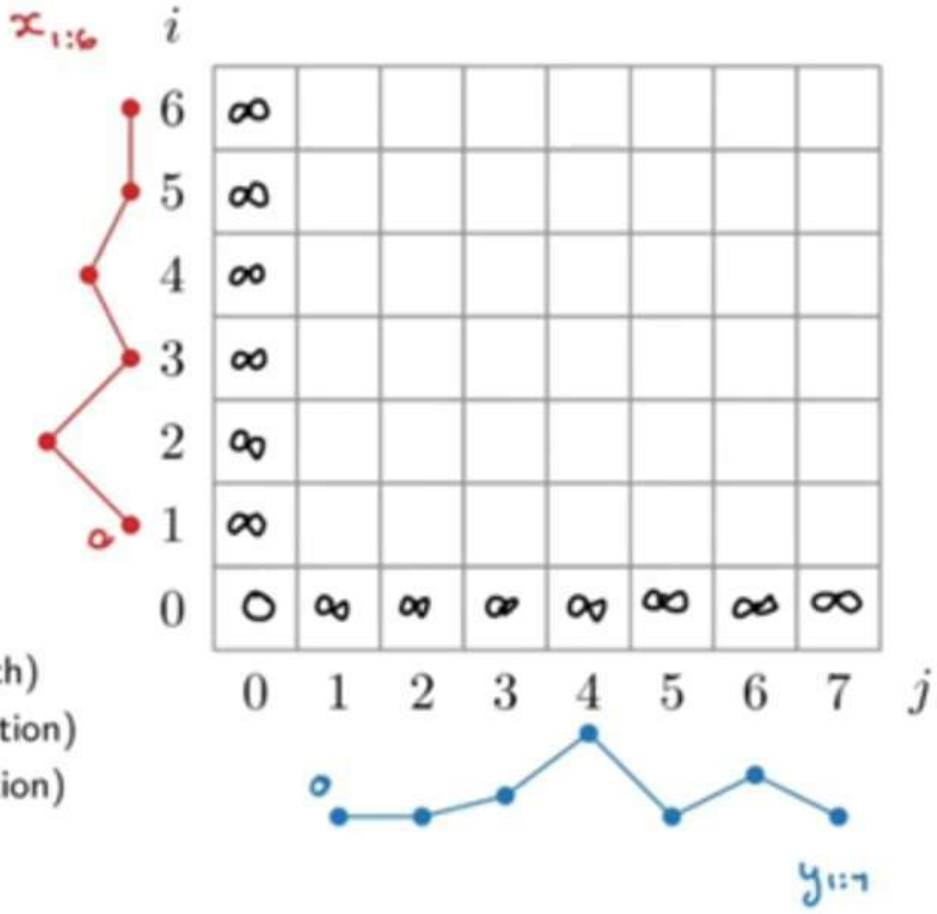
  $j = 1$

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$D_{1,1}$ x_1, y_1 $= 0$ $D_{0,0} = 0$ $D_{0,1} = \infty$ $D_{1,0} = \infty$

$d(x_i, y_j) = |x_i - y_j|$
- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

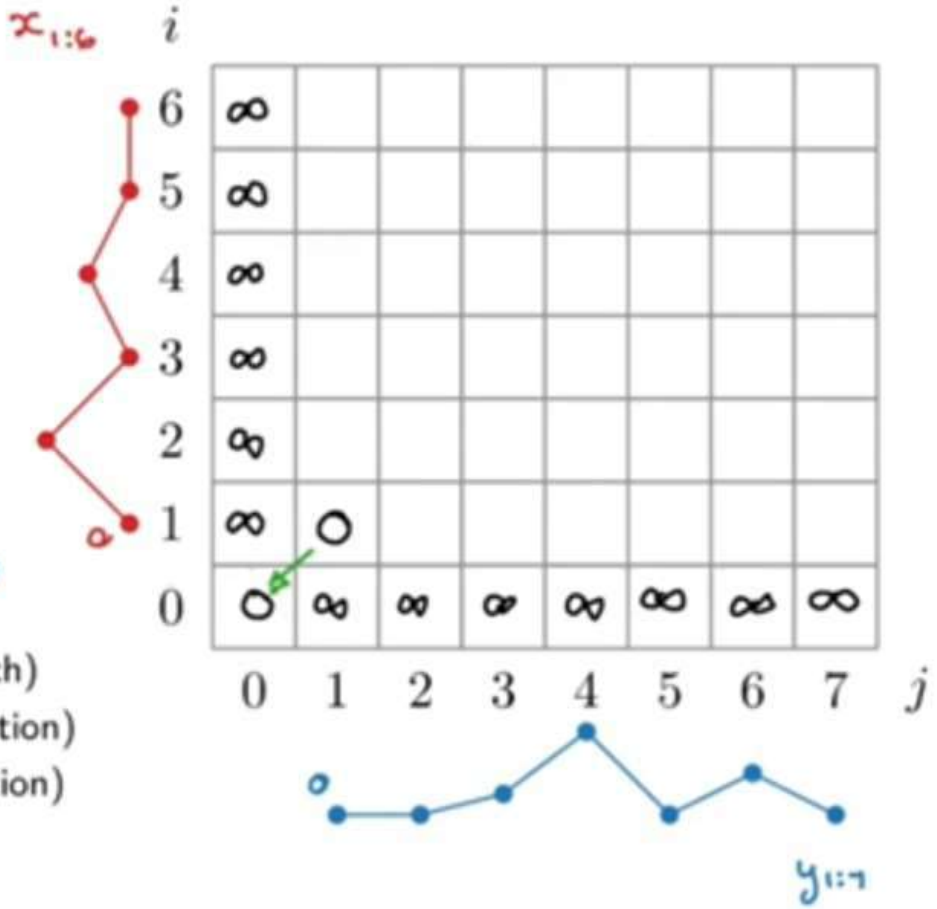
  $D_{0,0} = 0$
- Calculate cost matrix:

  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$d(x_i, y_j) = |x_i - y_j|$
- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

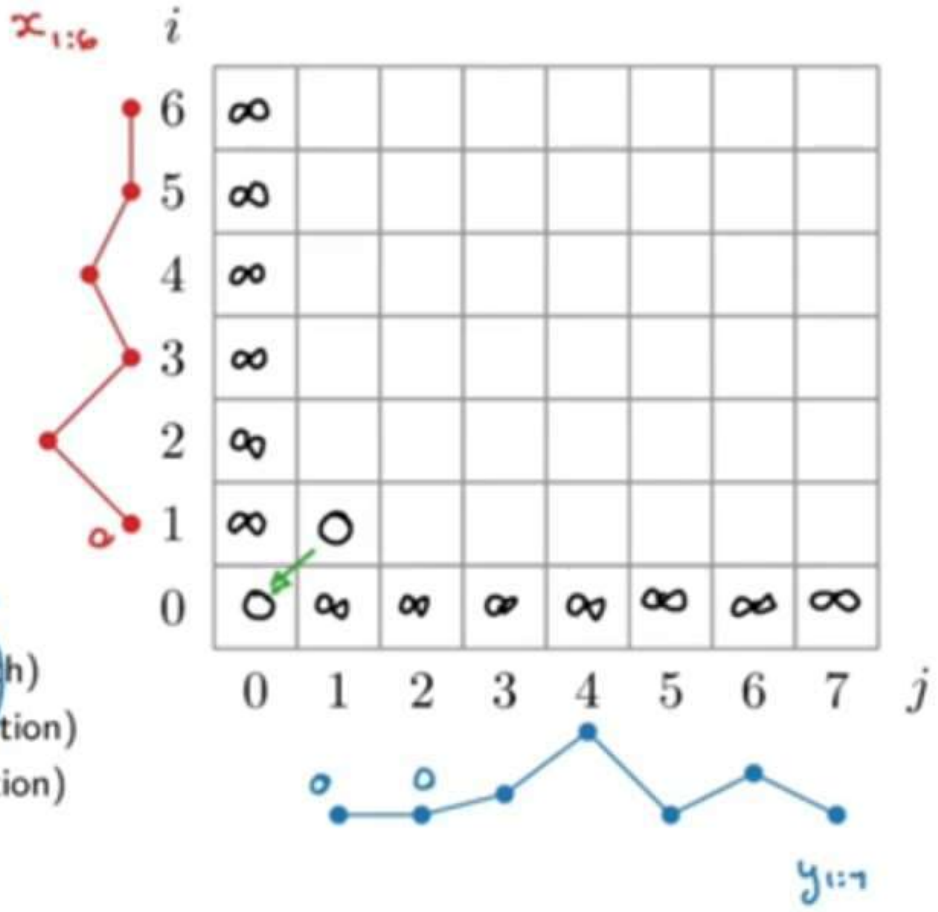
  $D_{0,0} = 0$
- Calculate cost matrix:

  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$d(x_i, y_j) = |x_i - y_j|$
- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

  $D_{0,0} = 0$
- Calculate cost matrix:

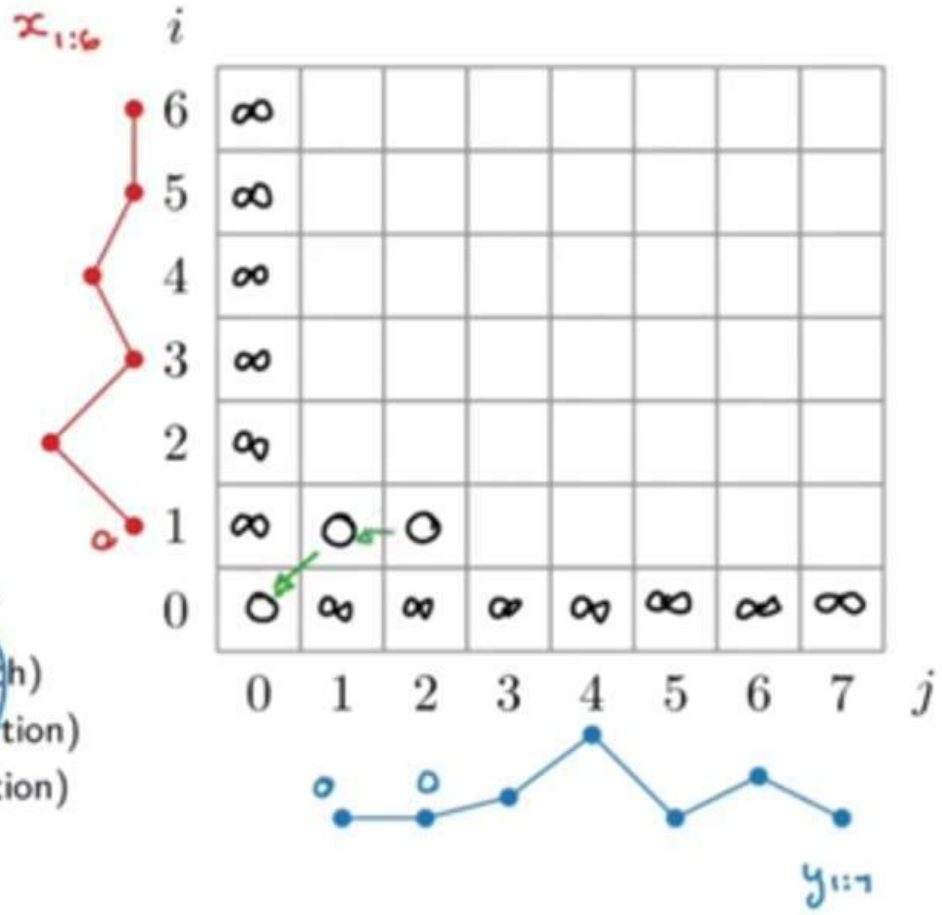
  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$d(x_i, y_j) = |x_i - y_j|$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

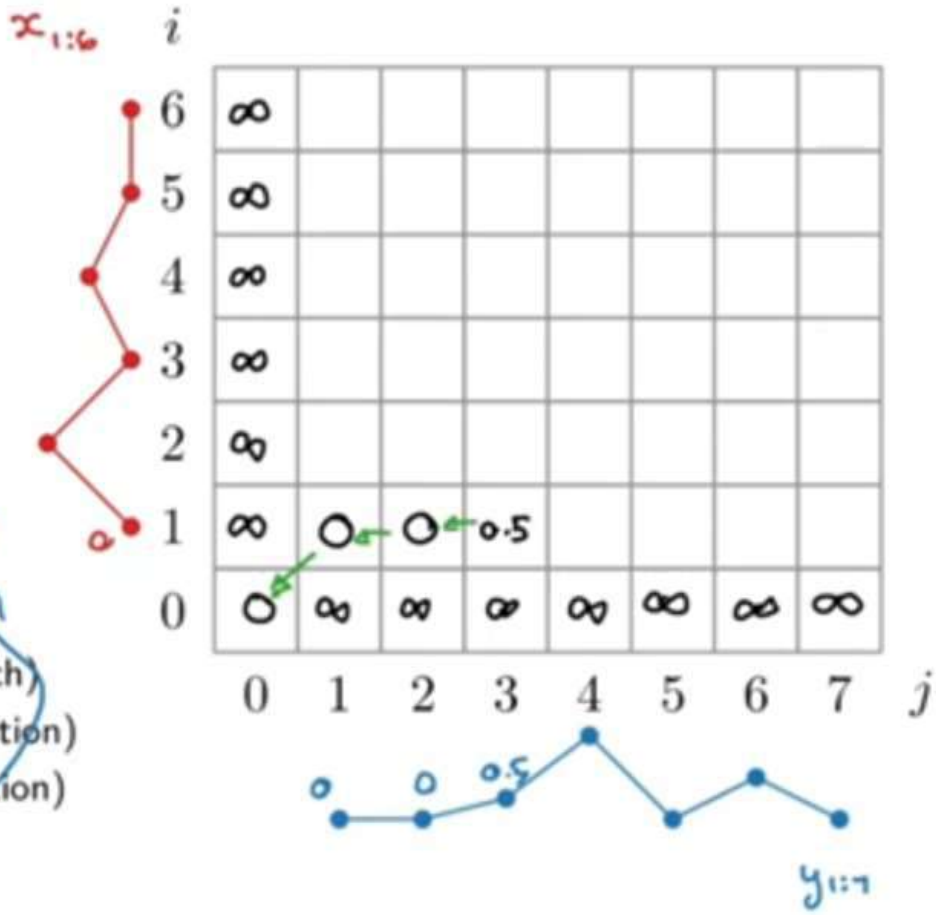
  $D_{0,0} = 0$
- Calculate cost matrix:

  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$d(x_i, y_j) = |x_i - y_j|$
- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

  $D_{0,0} = 0$
- Calculate cost matrix:

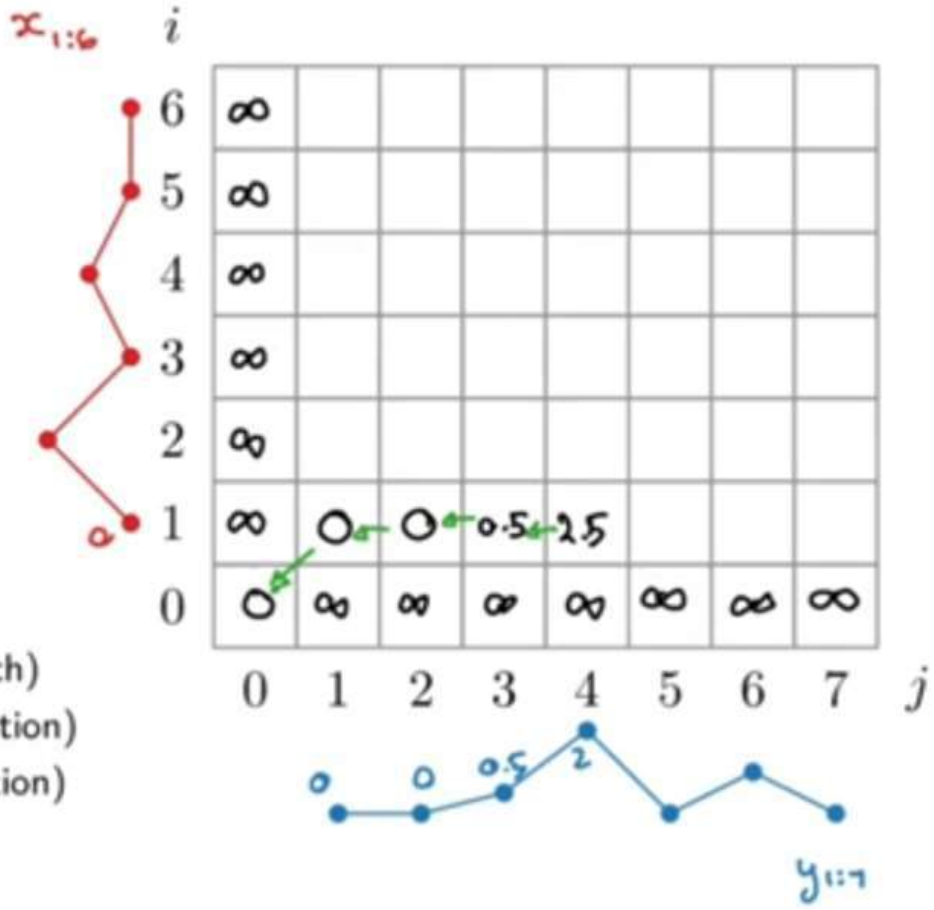
  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$d(x_i, y_j) = |x_i - y_j|$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

  $D_{0,0} = 0$
- Calculate cost matrix:

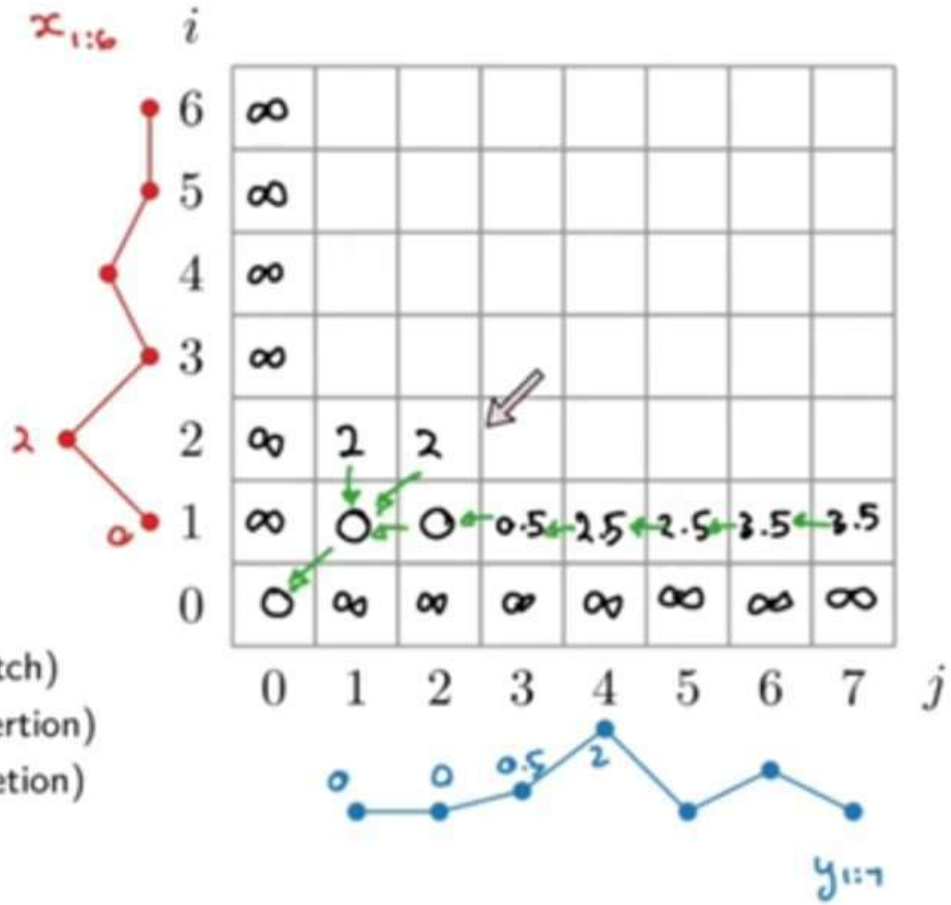
  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$d(x_i, y_j) = |x_i - y_j|$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

  $D_{0,0} = 0$
- Calculate cost matrix:

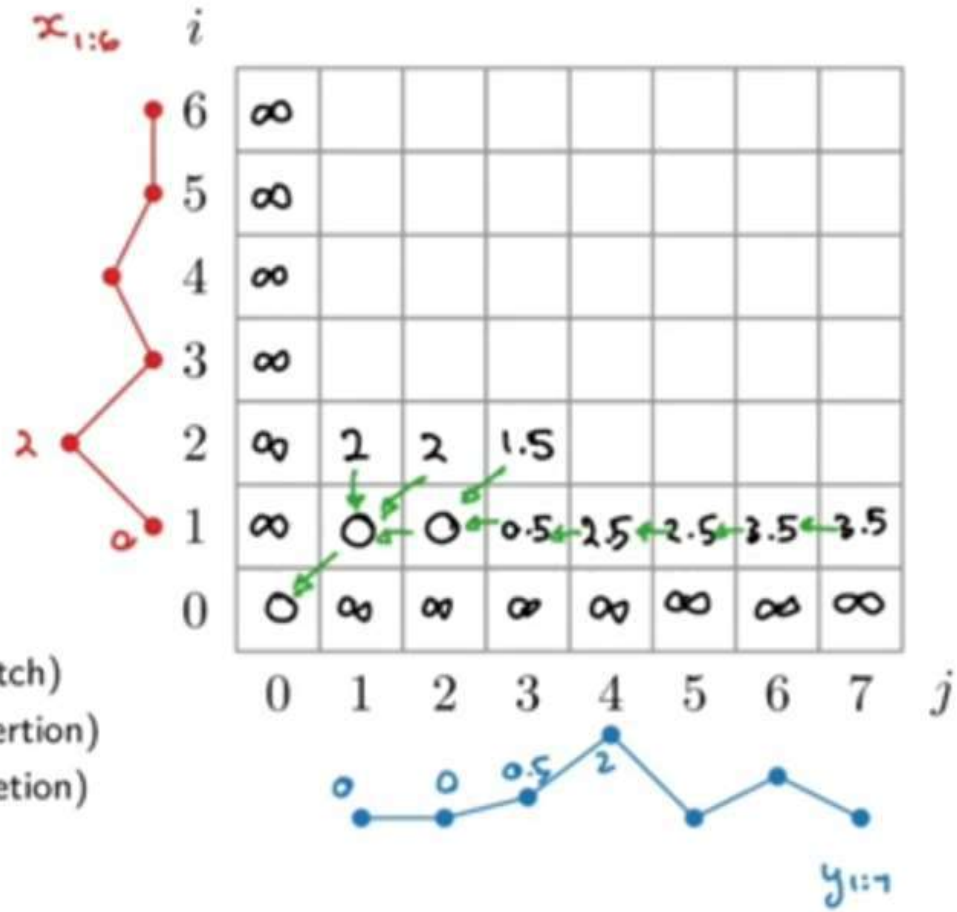
  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$d(x_i, y_j) = |x_i - y_j|$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

  $D_{0,0} = 0$
- Calculate cost matrix:

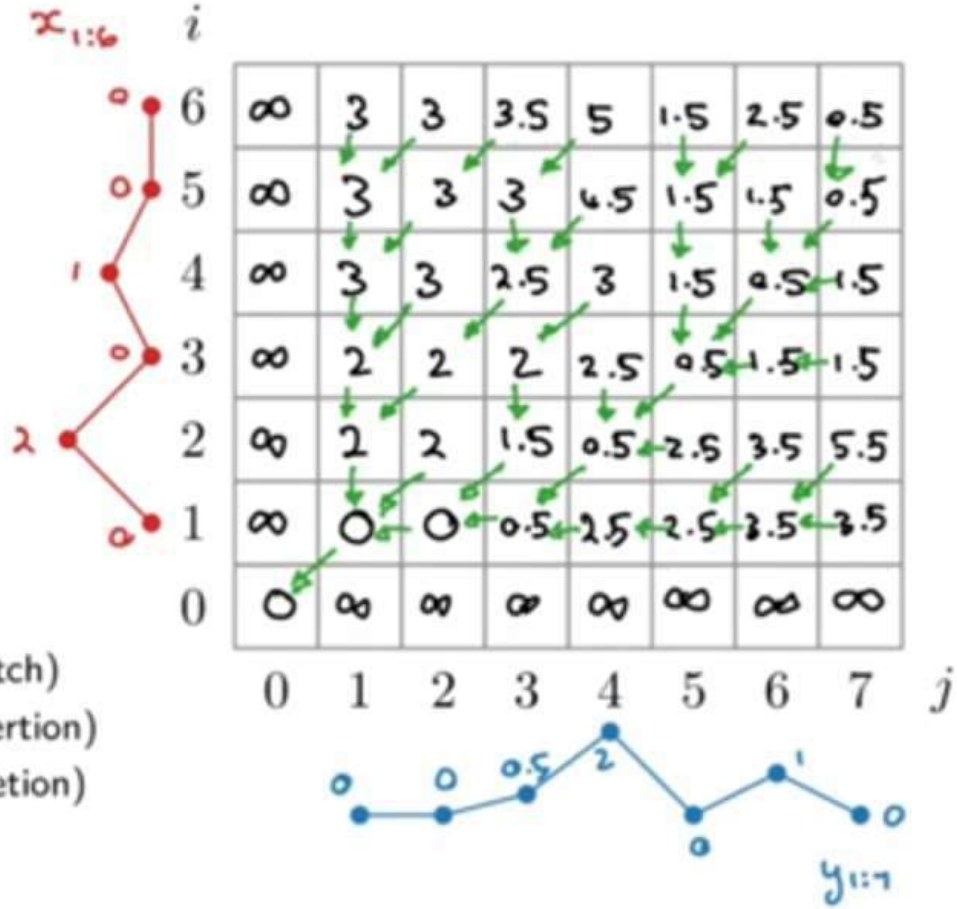
  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$d(x_i, y_j) = |x_i - y_j|$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

  $D_{0,0} = 0$
- Calculate cost matrix:

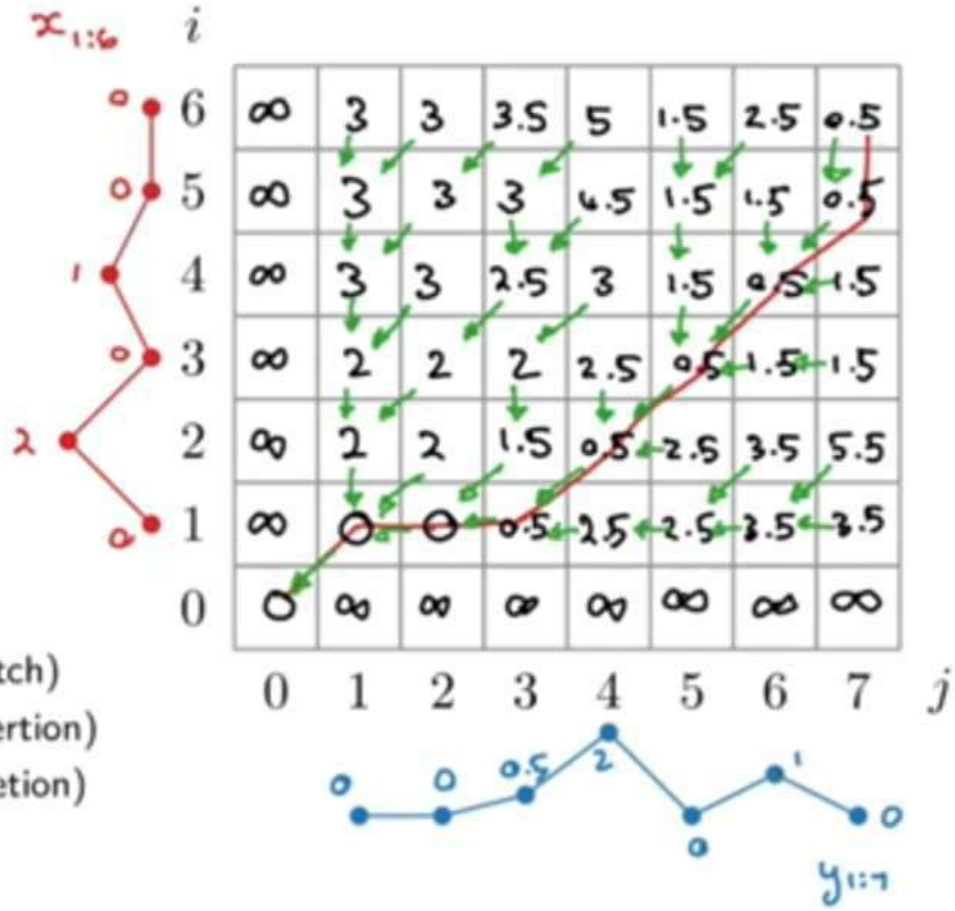
  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

$d(x_i, y_j) = |x_i - y_j|$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times(M+1)}$
- Initialization:

  for $i = 1$ to N: $D_{i,0} = \infty$

  for $j = 1$ to M: $D_{0,j} = \infty$

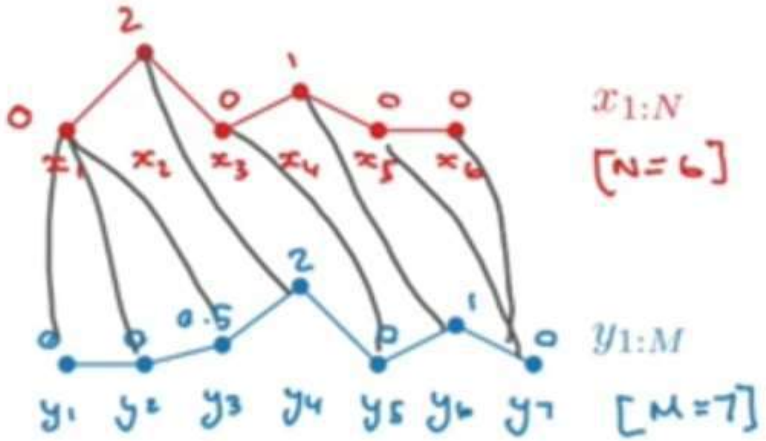
  $D_{0,0} = 0$
- Calculate cost matrix:

  for $i = 1$ to N:

  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$



Time Series and Dynamic Time Warping

Acknowledgement
CSE 6363 – Machine Learning
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

Thank you for your attention