CCE Dept. Colloquium

Support Vector Machines and Kernels on Time-Series

mcuturi@i.kyoto-u.ac.jp

Outline of the Talk

Very brief introduction to the Support Vector Machine

- Intuition and computation
- Geometric interpretation

Very brief introduction to kernel methods

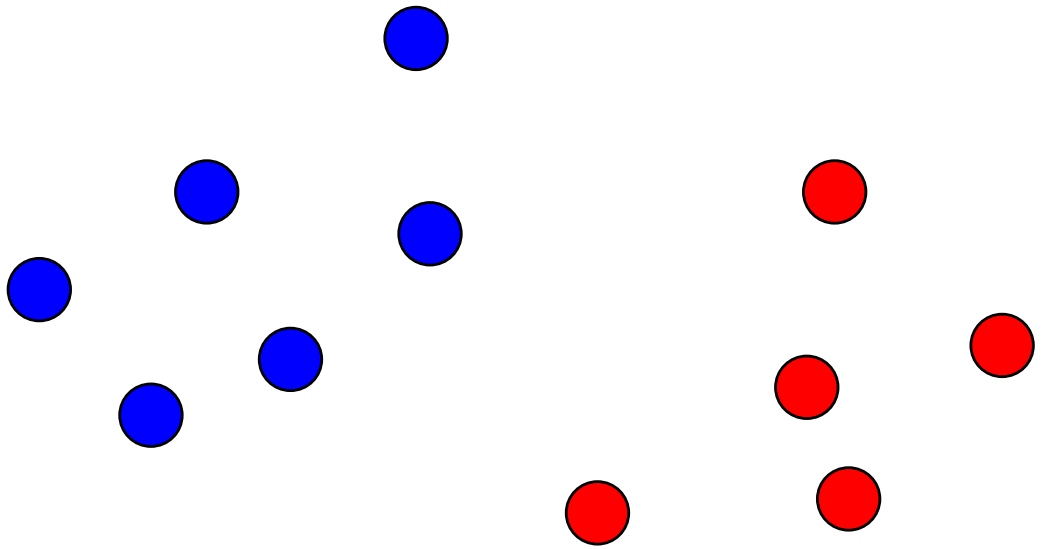
- What are kernels in a machine learning context?

Present new work on kernels for time-series

- Inspired by the Dynamic Time Warping Distance
  - Cuturi-Vert-Birkenes-Matsui,
    A kernel for Time-Series based on Global Alignments (ICASSP 2007)
  - Cuturi, *Fast Global Alignment Kernels* (ICML 2011)

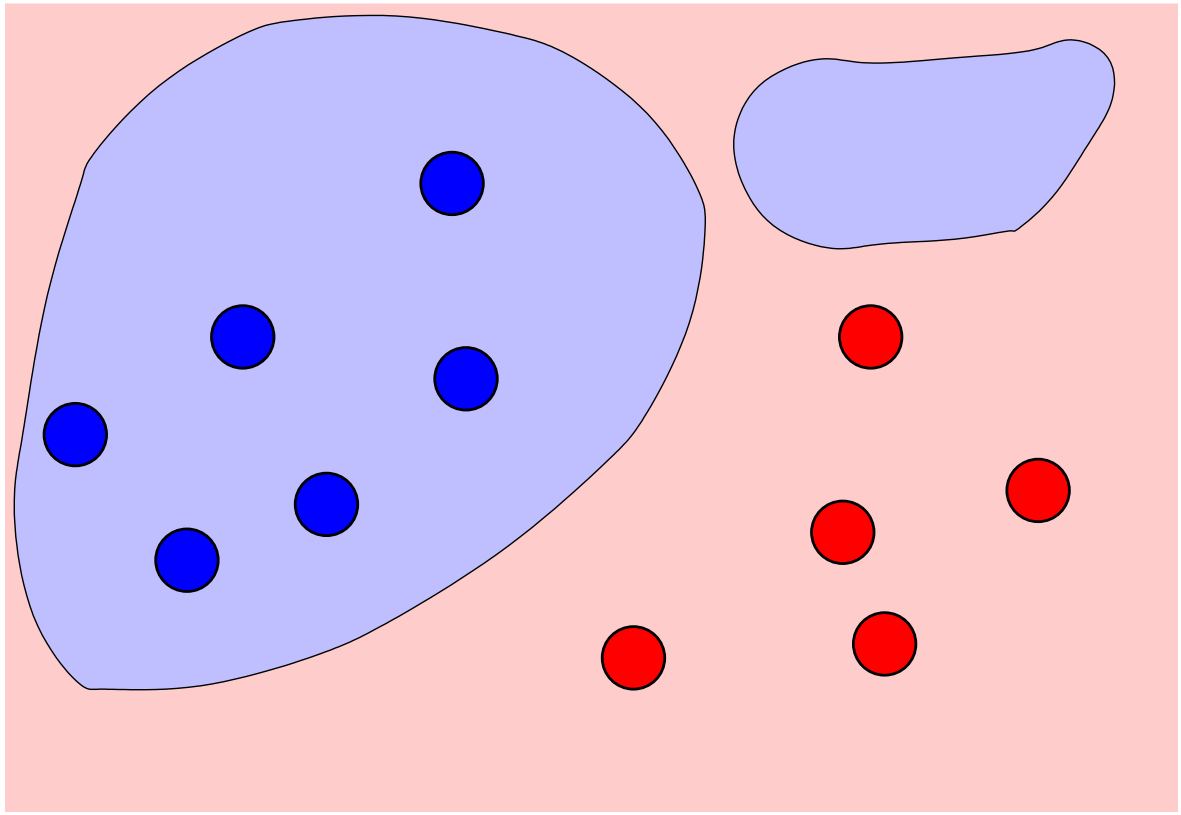
Support Vector Machines

Binary Classification Separation Surfaces for Vectors



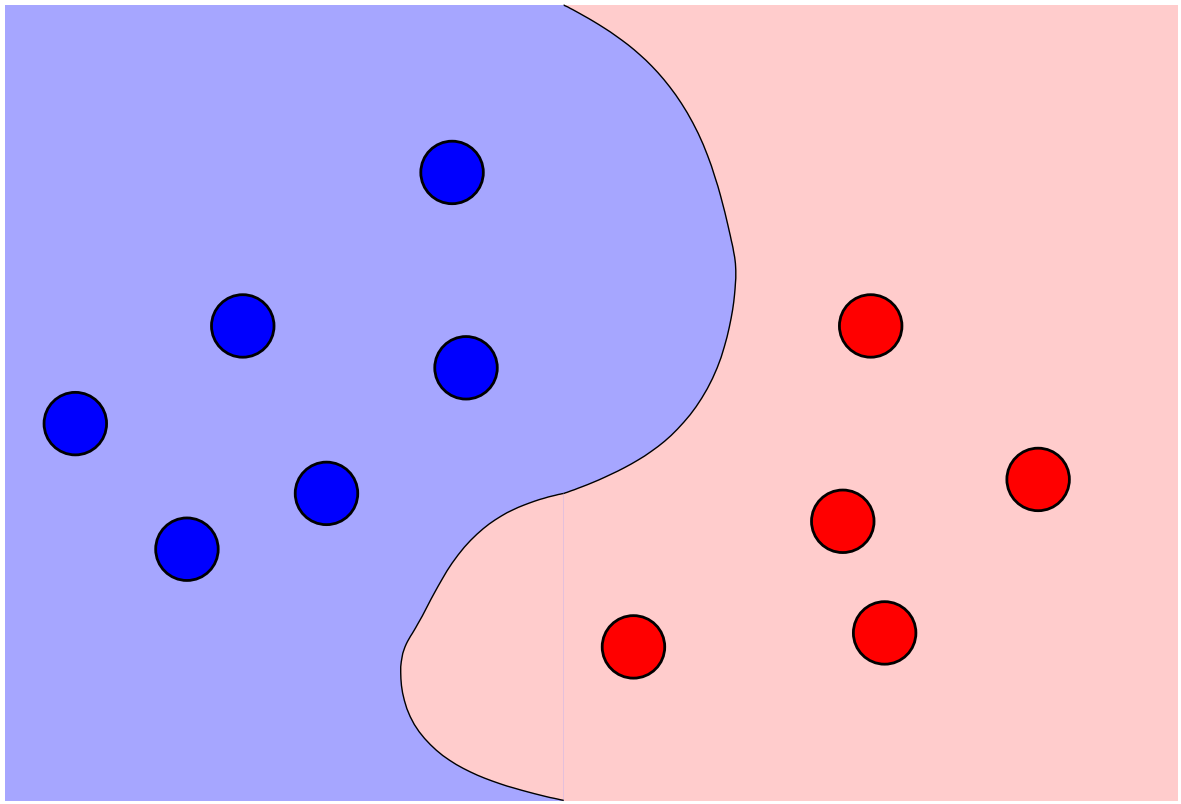
What is a classification rule?

Binary Classification Separation Surfaces for Vectors



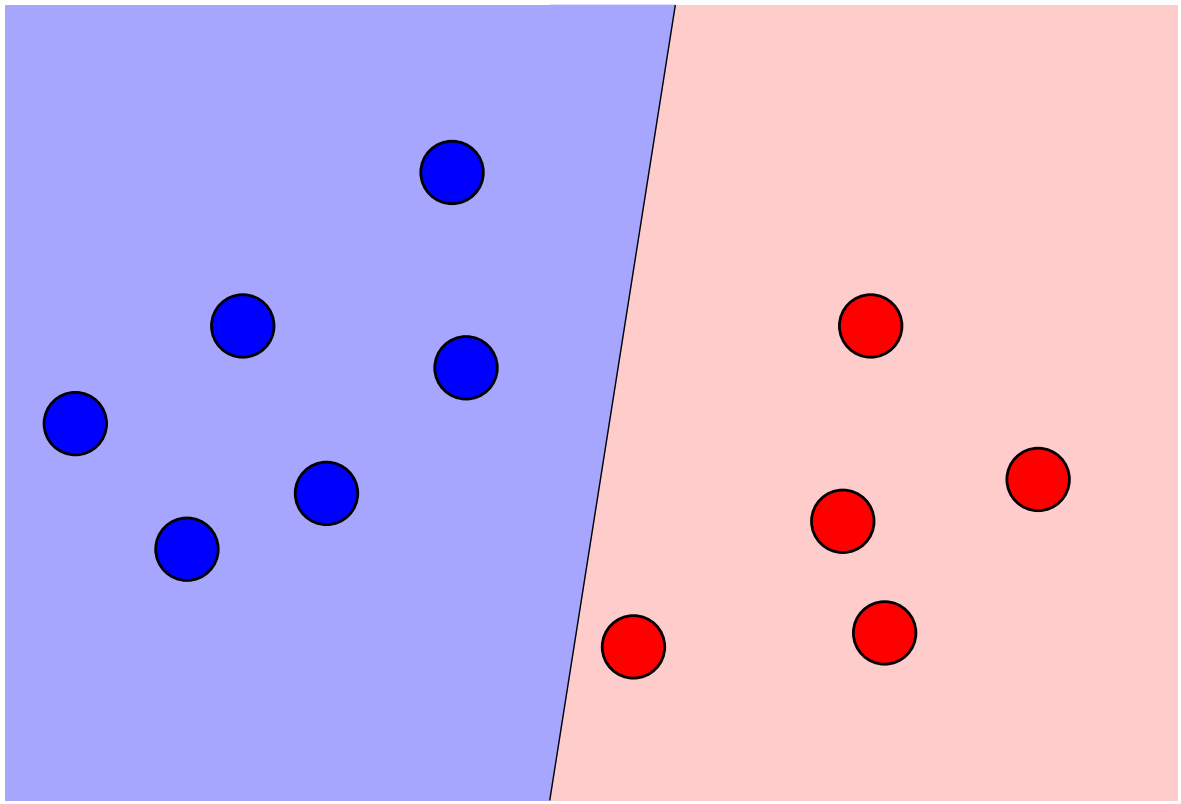
Classification rule = a partition of $\mathbb{R}^d$ into two sets

Classification Separation Surfaces for Vectors



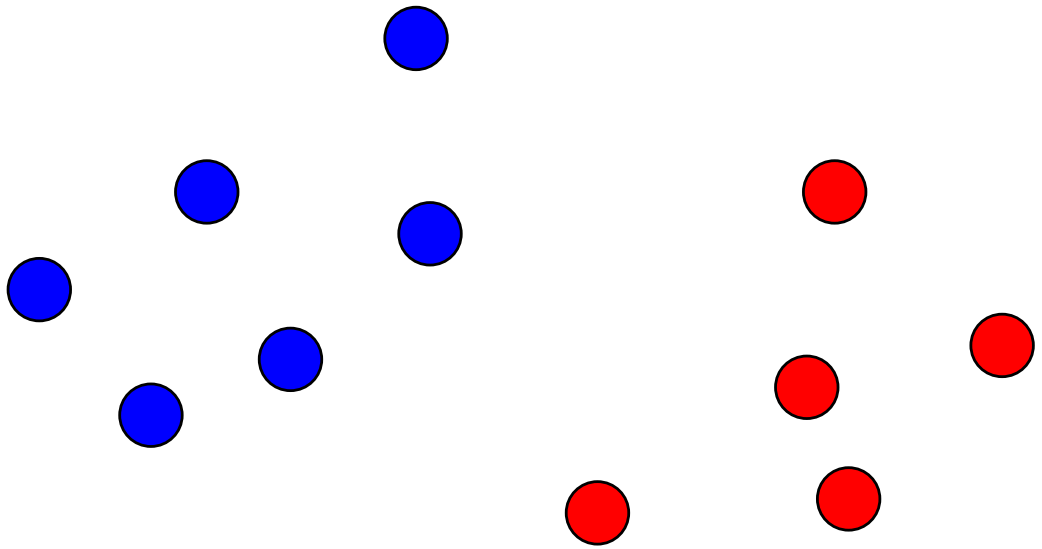
Can be defined by a single surface, $e.g.$ a curved line

Classification Separation Surfaces for Vectors



Even more **simple**: using **straight lines** and halfspaces.

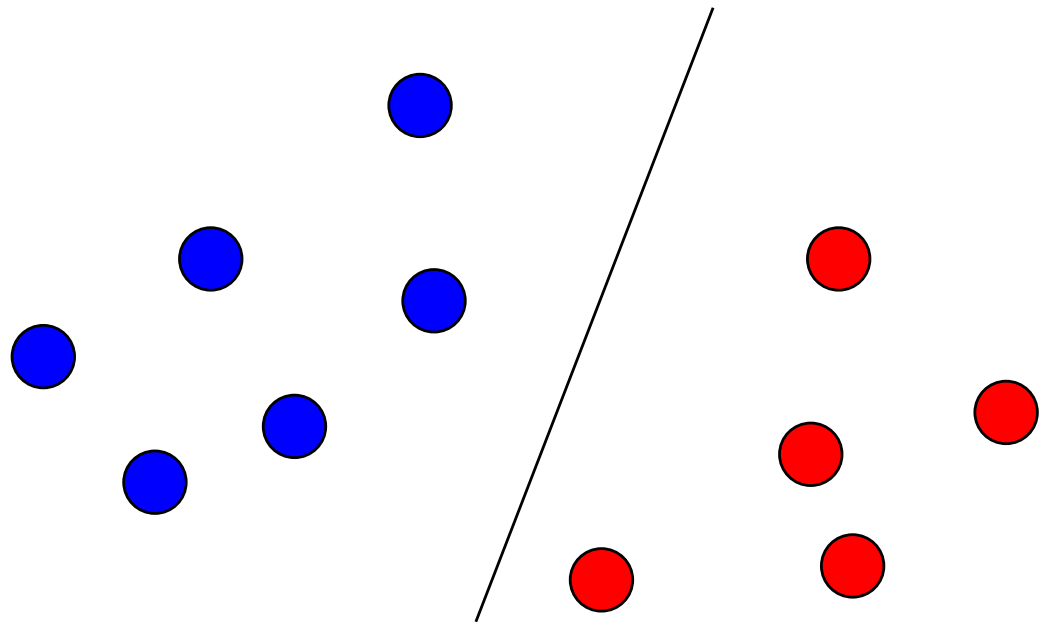
Classification Separation Surfaces for Vectors



Given two sets of points...

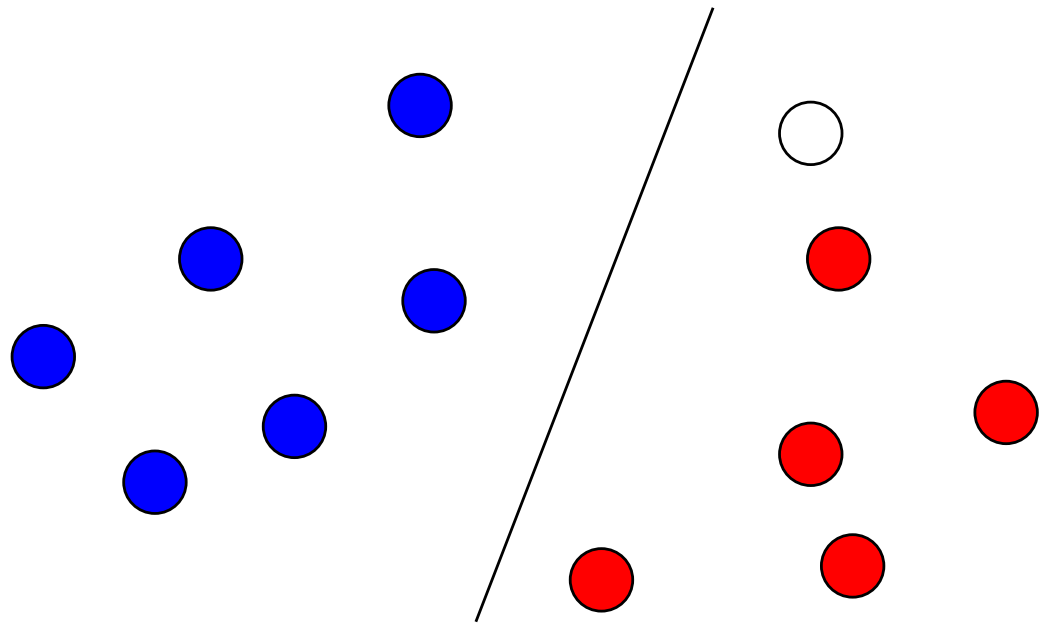
Some slides from now on are taken from Jean-Philippe Vert's lectures

Classification Separation Surfaces for Vectors



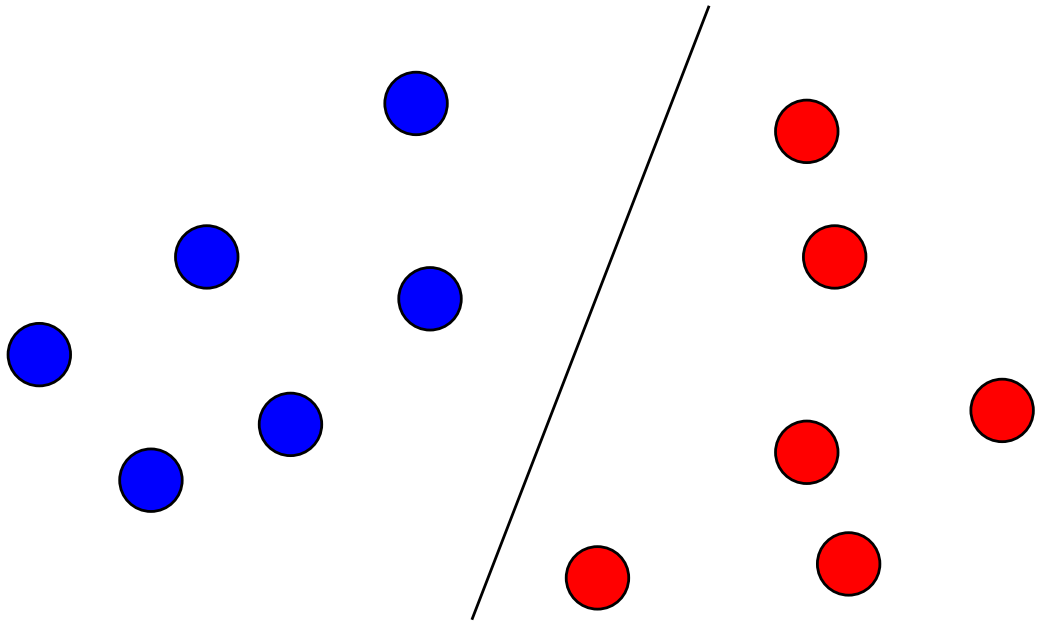
It is sometimes possible to separate them perfectly

Classification Separation Surfaces for Vectors

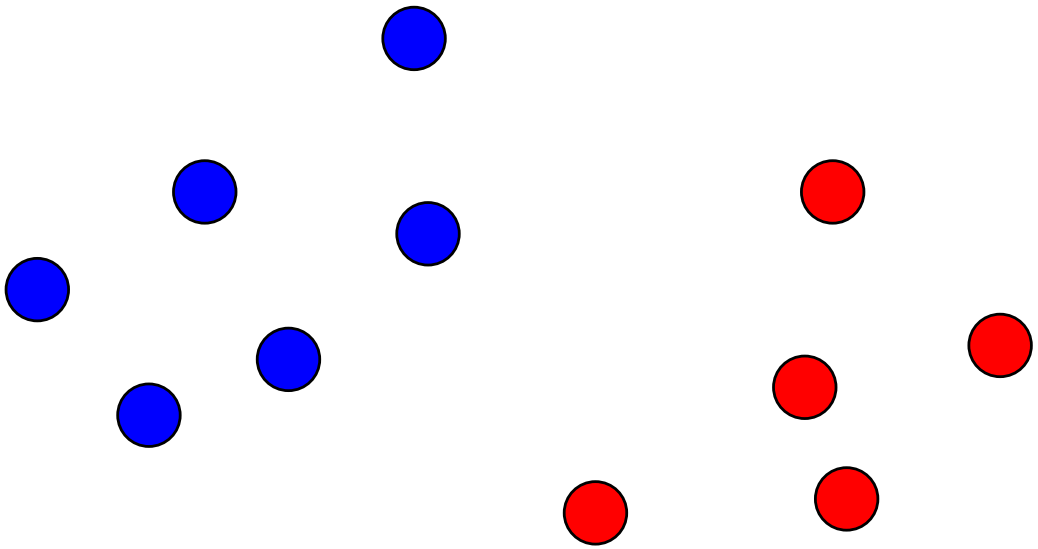


Each choice might look equivalently good on the training set,
but it will have obvious impact on new points

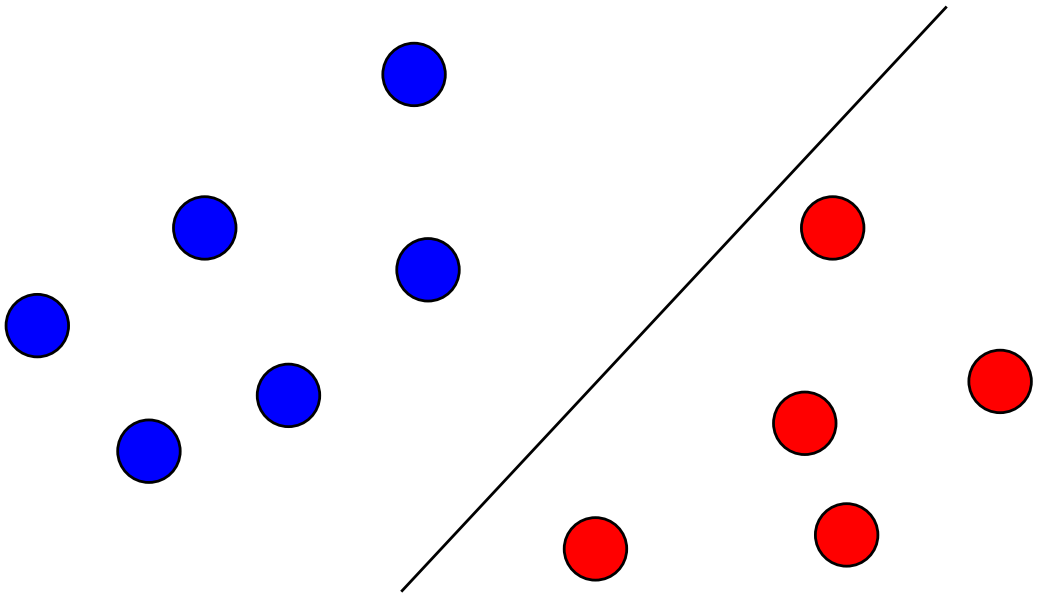
Classification Separation Surfaces for Vectors



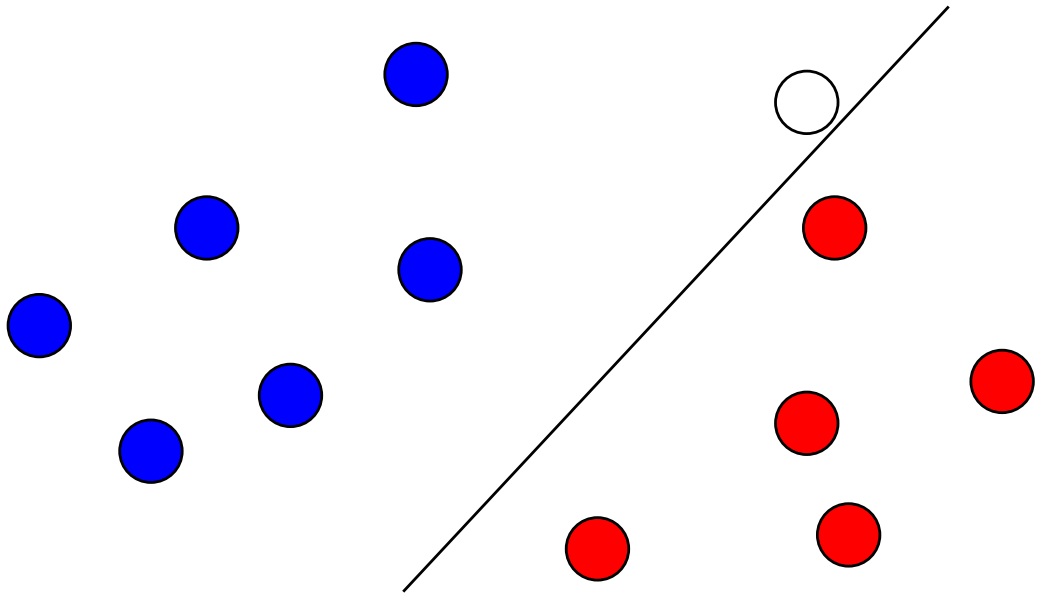
Linear classifier, some degrees of freedom



Linear classifier, some degrees of freedom

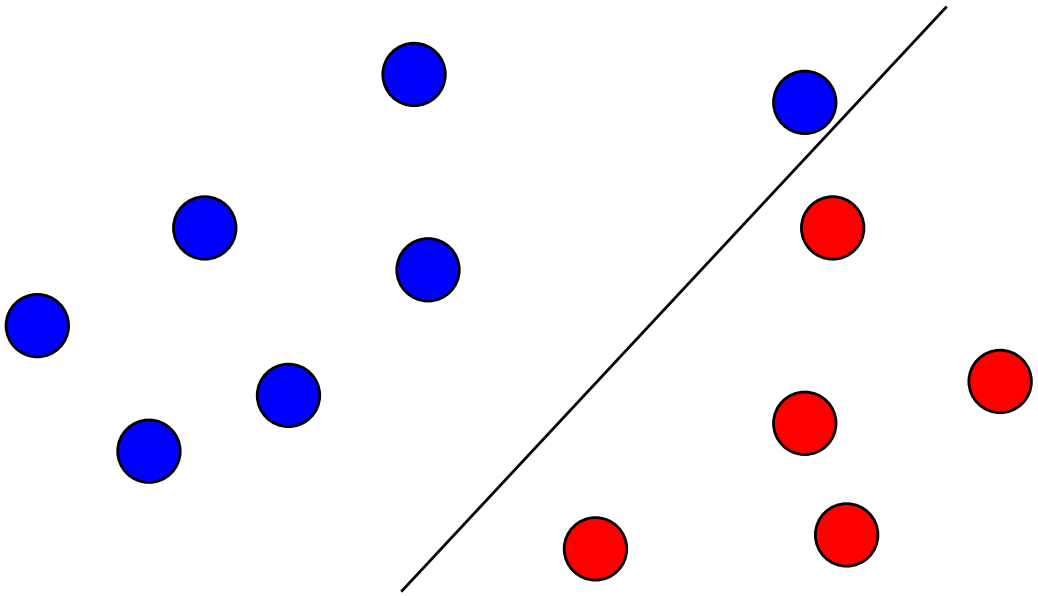


Linear classifier, some degrees of freedom

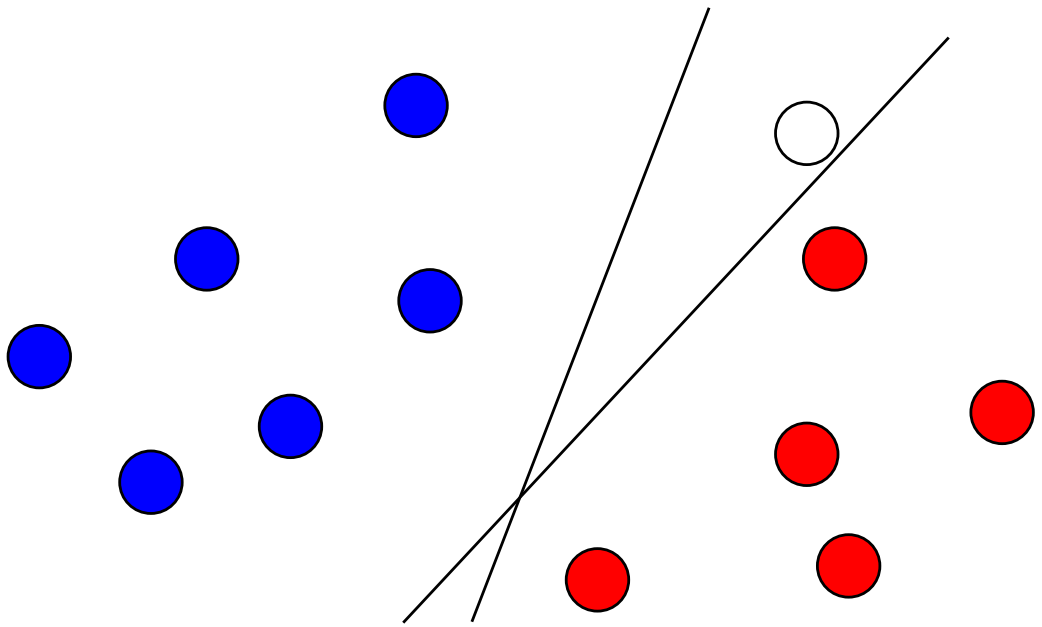


Specially close to the border of the classifier

Linear classifier, some degrees of freedom

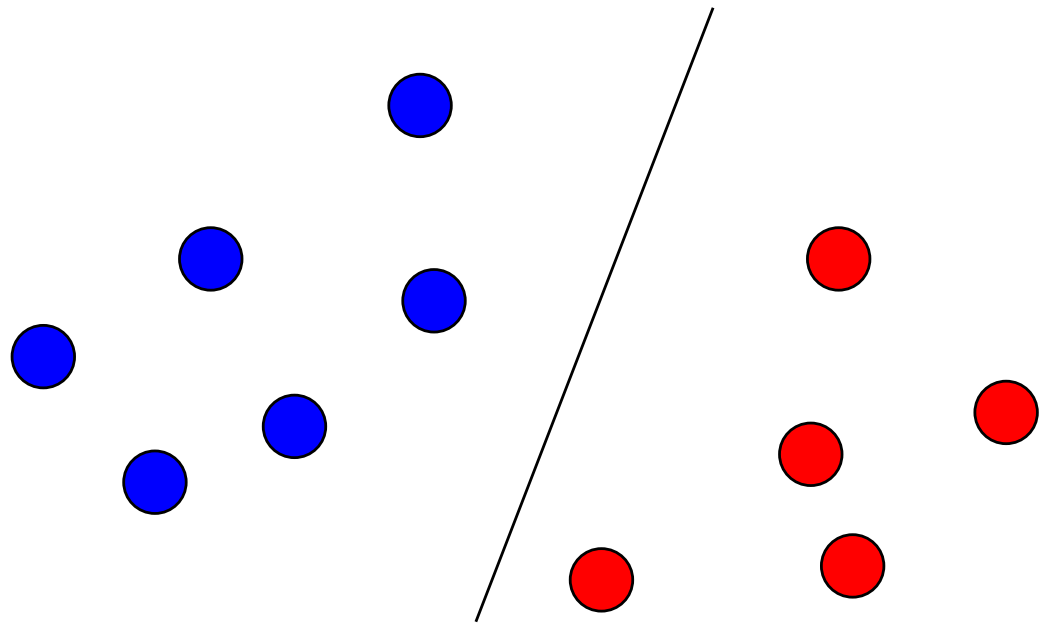


Linear classifier, some degrees of freedom



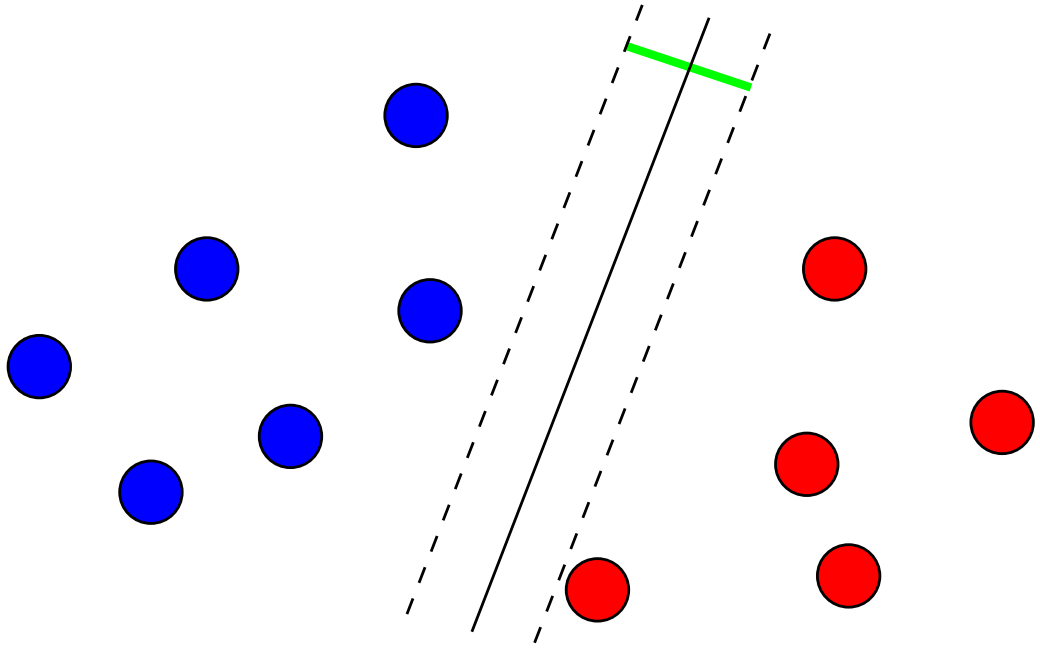
For each different technique, different results, different performance.

A criterion to select a linear classifier: the margin

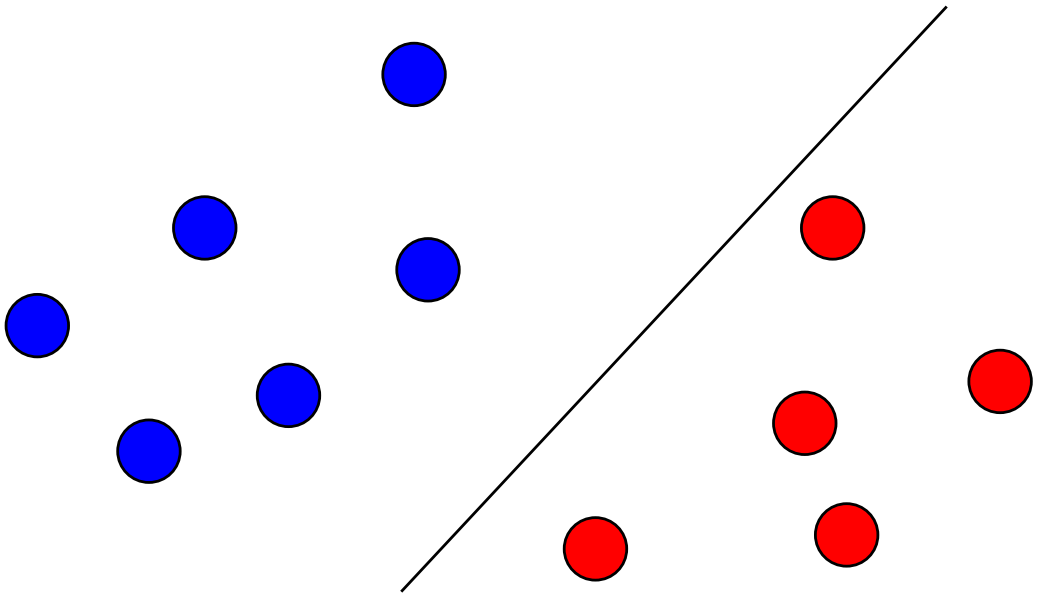


Idea: look for the biggest possible "buffer" between red and blue points.

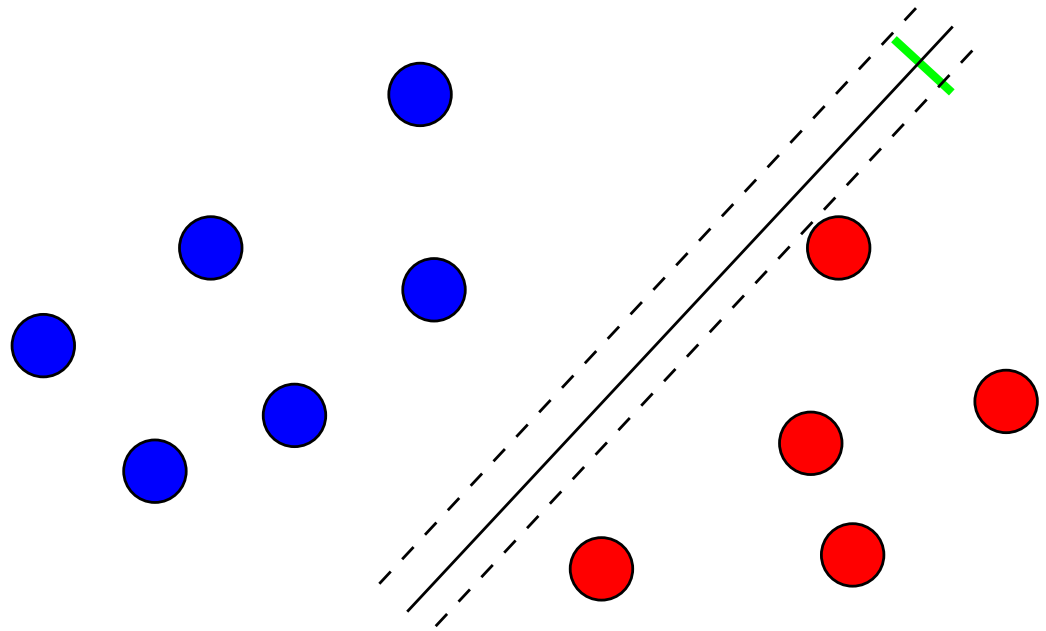
A criterion to select a linear classifier: the margin



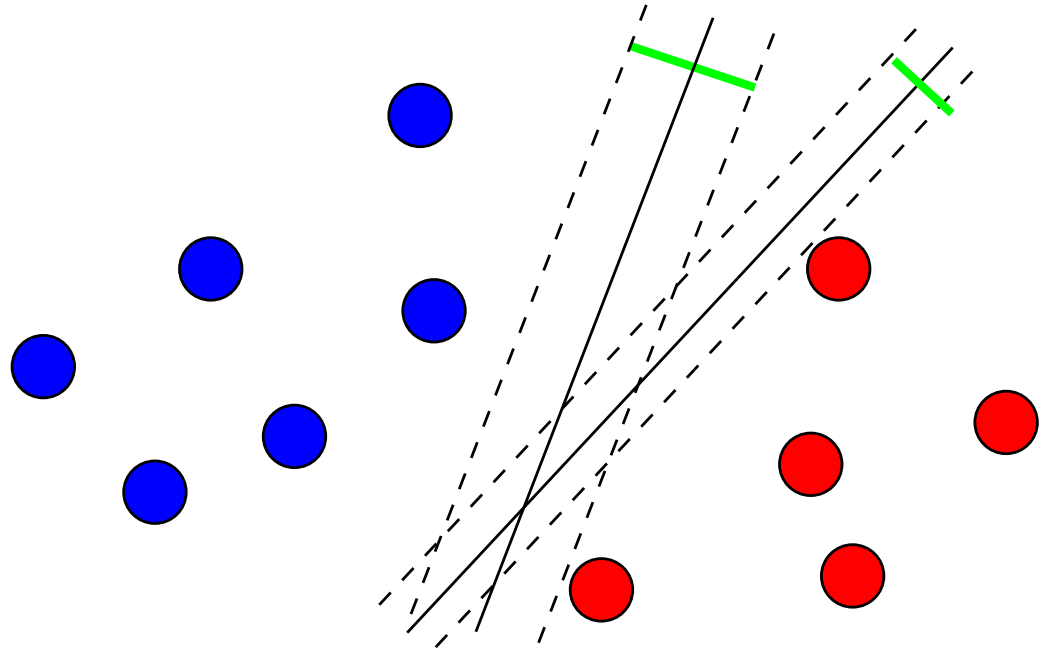
A criterion to select a linear classifier: the margin



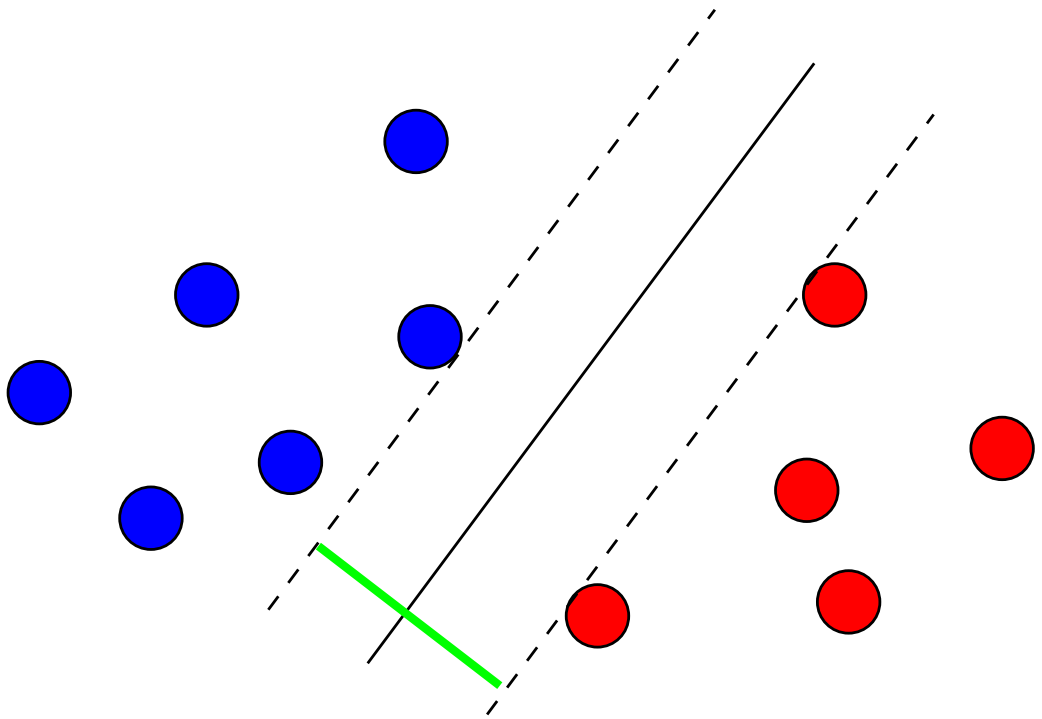
A criterion to select a linear classifier: the margin



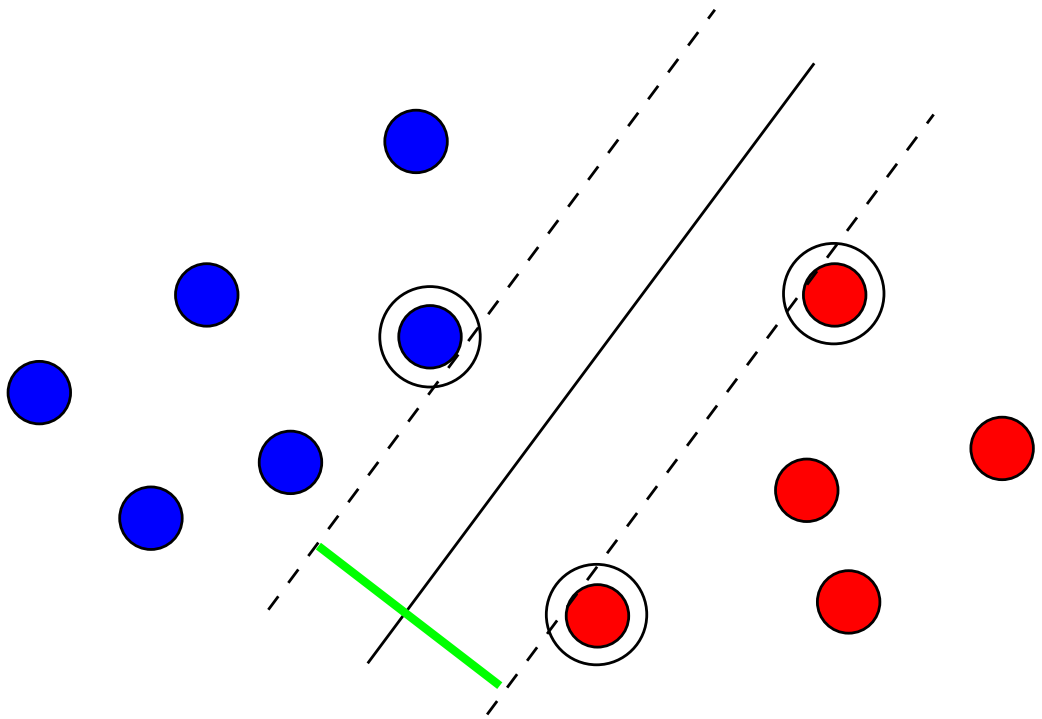
A criterion to select a linear classifier: the margin



Largest Margin Linear Classifier ?



Support Vectors with Large Margin



Finding the optimal hyperplane

- Consider n **labeled points** $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$, with $i = 1, \cdots, n$.
- Finding the optimal hyperplane is equivalent to finding $(\mathbf{w}, b)$ which minimize:

$$\|\mathbf{w}\|^2$$

under the constraints:

$$\forall i = 1, \ldots, n, \qquad y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) - 1 \geq 0.$$

This is a classical quadratic program on $\mathbb{R}^{d+1}$
linear constraints - quadratic objective

Dual problem

- introduce **one dual variable α_i for each constraint**,

The dual problem is

$$\begin{array}{ll} \text{maximize} & g(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \boldsymbol{\alpha_i} - \frac{1}{2} \sum_{i,j=1}^{n} \boldsymbol{\alpha_i \alpha_j y_i y_j} \mathbf{x}_i^{\boldsymbol{T}} \mathbf{x}_j \\ \text{such that} & 0 \leq \boldsymbol{\alpha_i}, \sum_{i=1}^{n} \boldsymbol{\alpha_i y_i} = 0. \end{array}$$

This is a **quadratic program** in $\mathbb{R}^n$, with *box constraints*.
α^* can be computed using elementary optimization software
(*e.g.* built-in `matlab` function)

- Strong duality holds. KKT gives us $\alpha_i(y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) - 1) = 0$,
...*hence*, either $\boldsymbol{\alpha_i = 0}$ or $\boldsymbol{y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) = 1}$.
- $\alpha_i \neq 0$ **only** for points on the support hyperplanes $\{(\mathbf{x}, y) | \, y_i(\mathbf{w}^T \mathbf{x}_i + b) = 1\}$.

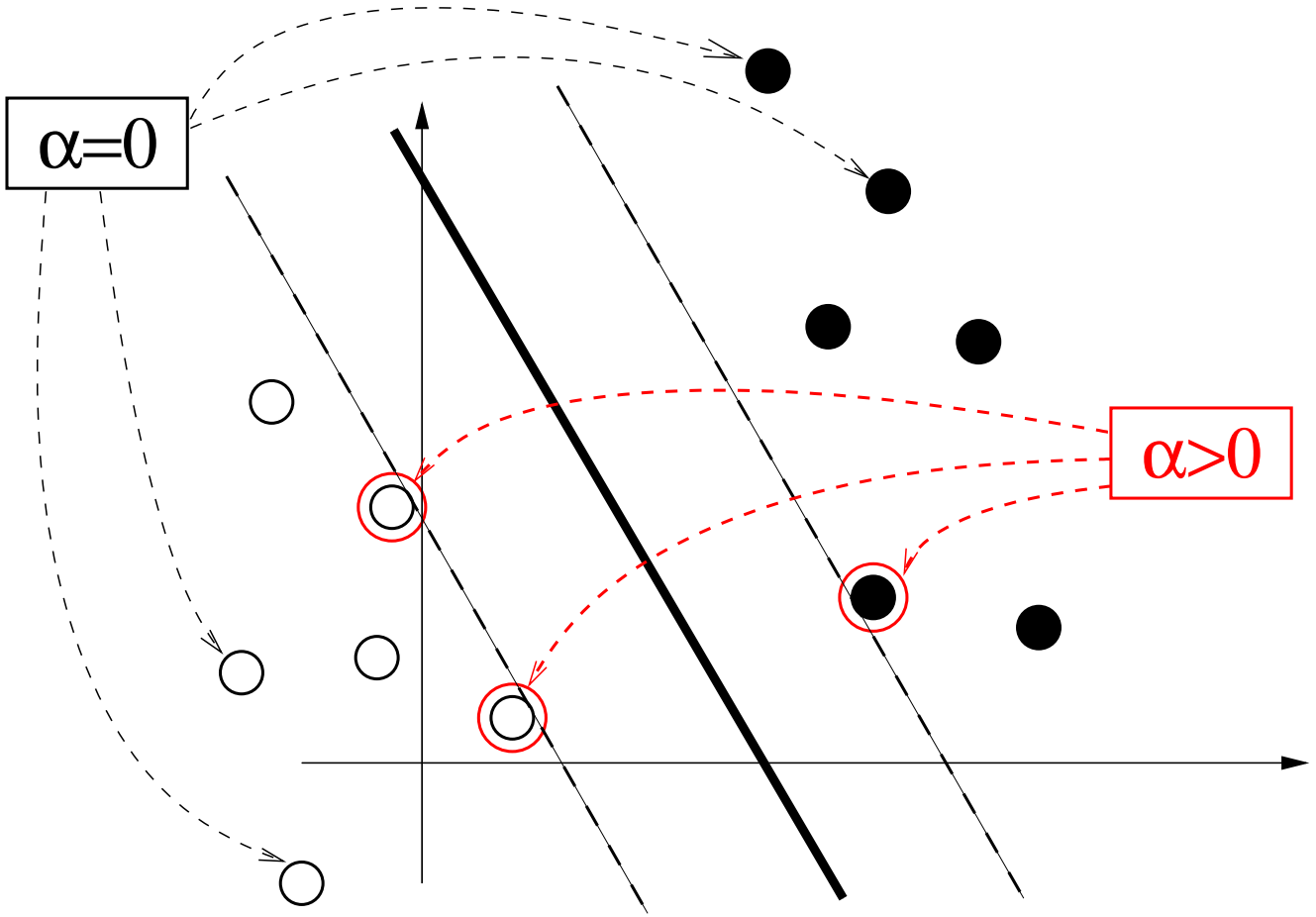
The final solution

- With α^*, we can recover $(\mathbf{w}^*, b^*)$.
- the **decision function** is therefore:

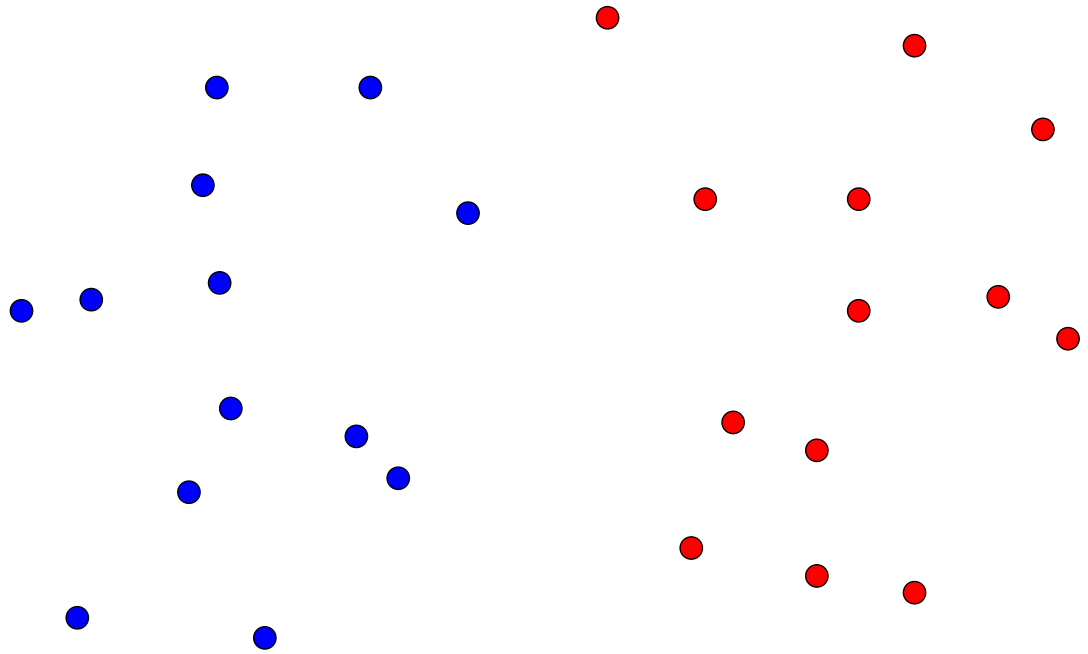
$$\begin{aligned} f^*(\mathbf{x}) &= (\mathbf{w}^*)^T \mathbf{x} + b^* \\ &= \left(\sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i^T \right) \mathbf{x} + b^*. \end{aligned}$$

- Here the **dual** solution gives us directly the **primal** solution.

Interpretation: support vectors

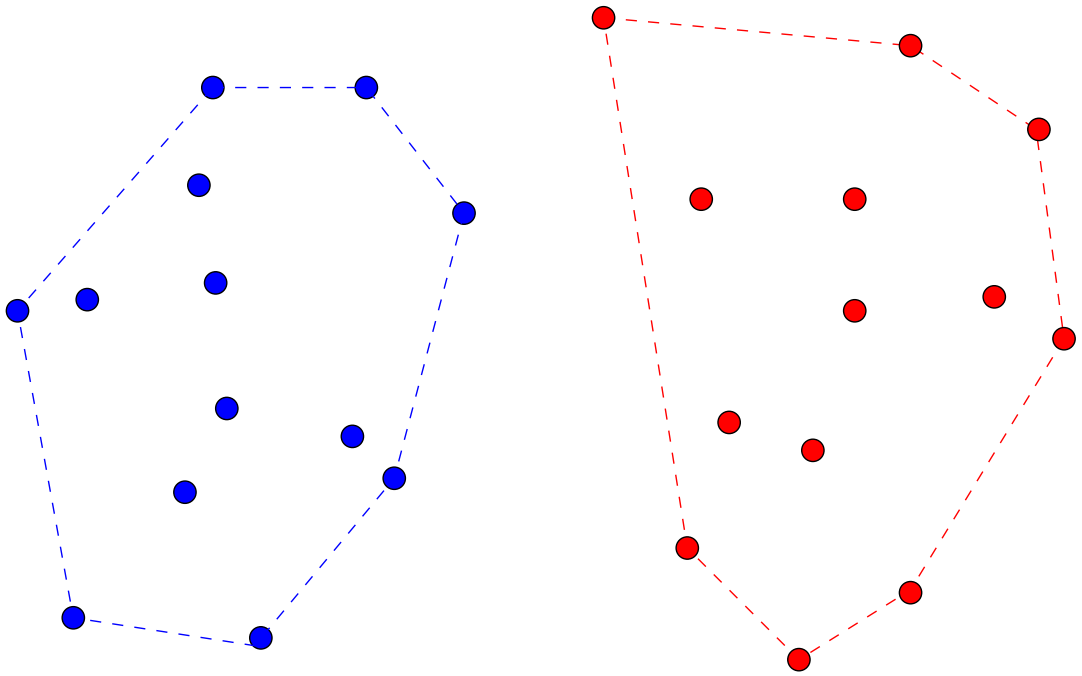


Another interpretation: Convex Hulls



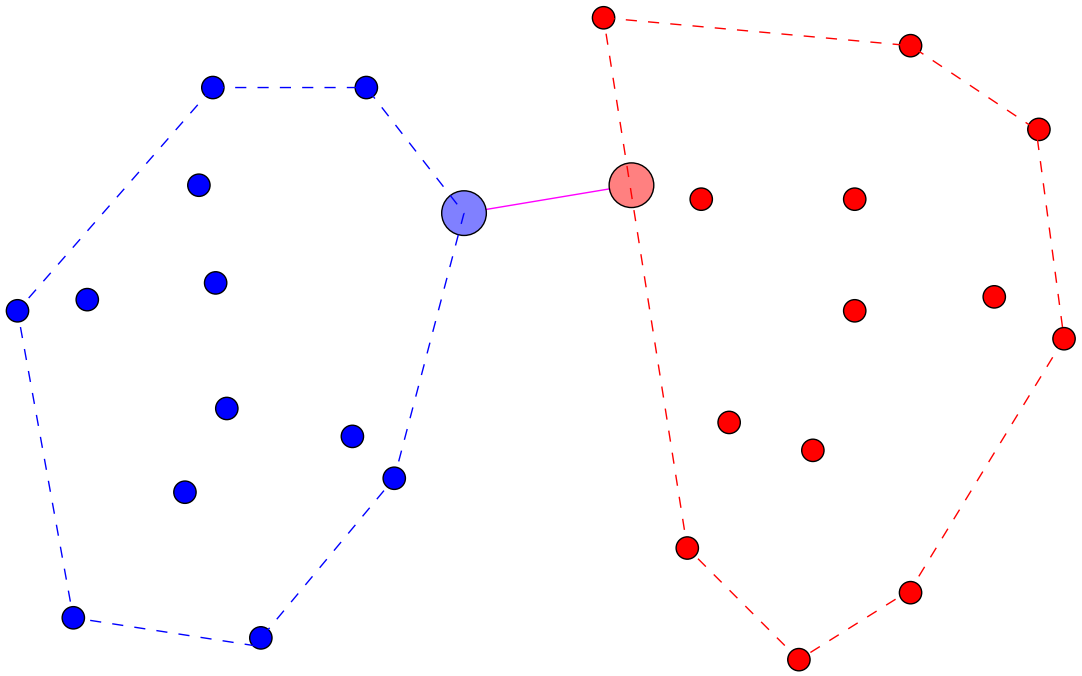
go back to 2 sets of points that are linearly separable

Another interpretation: Convex Hulls



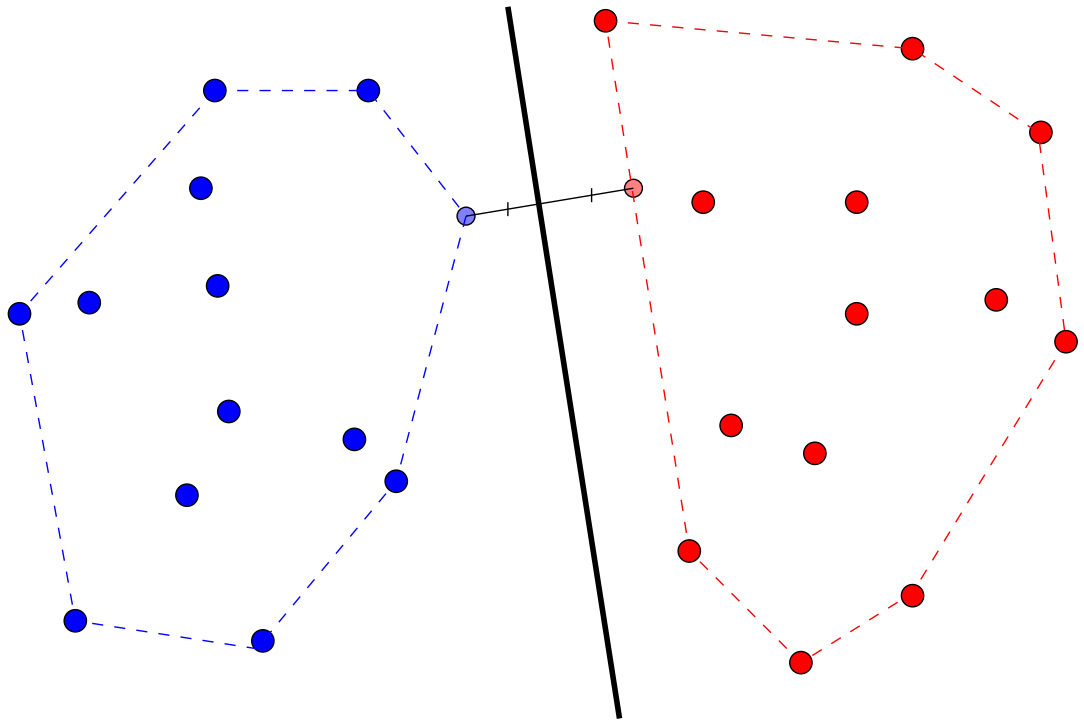
Linearly separable = convex hulls do not intersect

Another interpretation: Convex Hulls



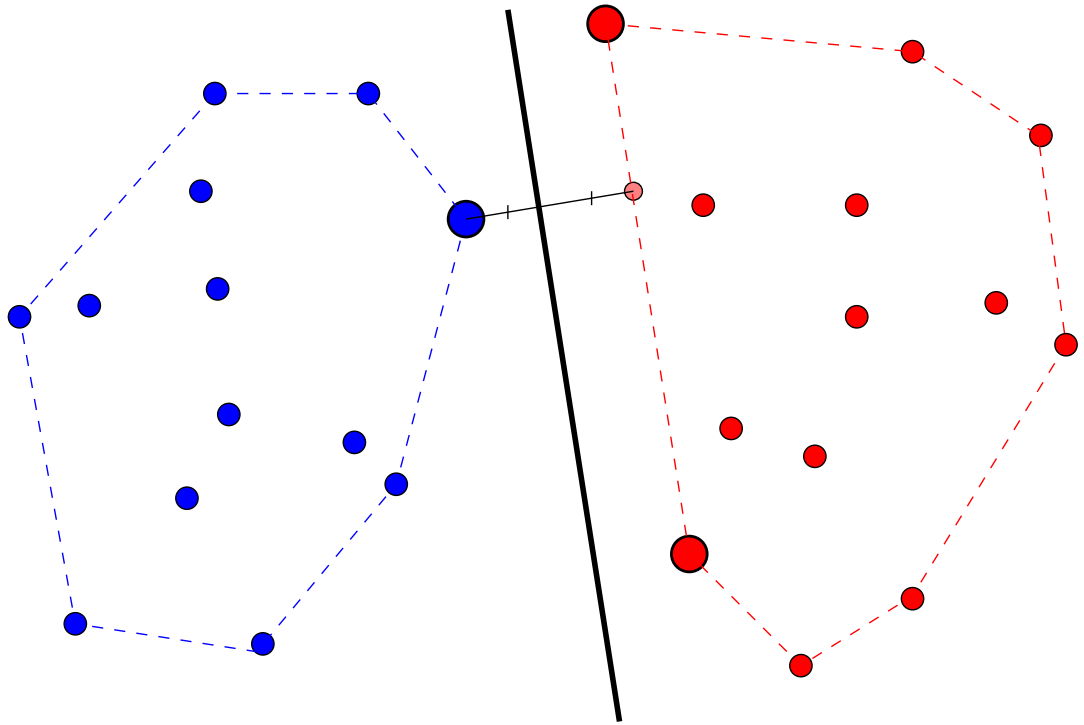
Find two closest points, one in each convex hull

Another interpretation: Convex Hulls



The SVM = bisection of that segment

Another interpretation: Convex Hulls

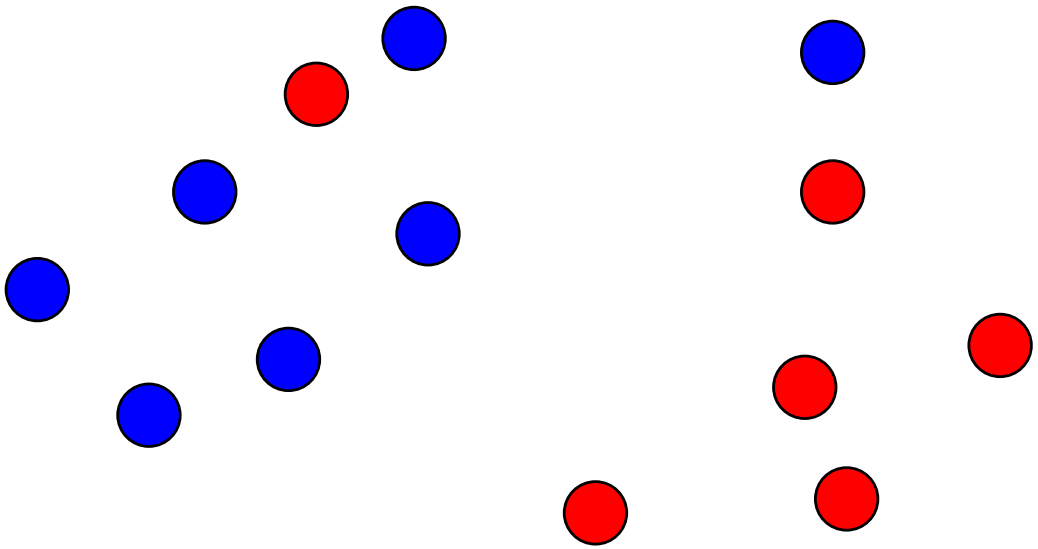


support vectors = extreme points of the faces on which the two points lie

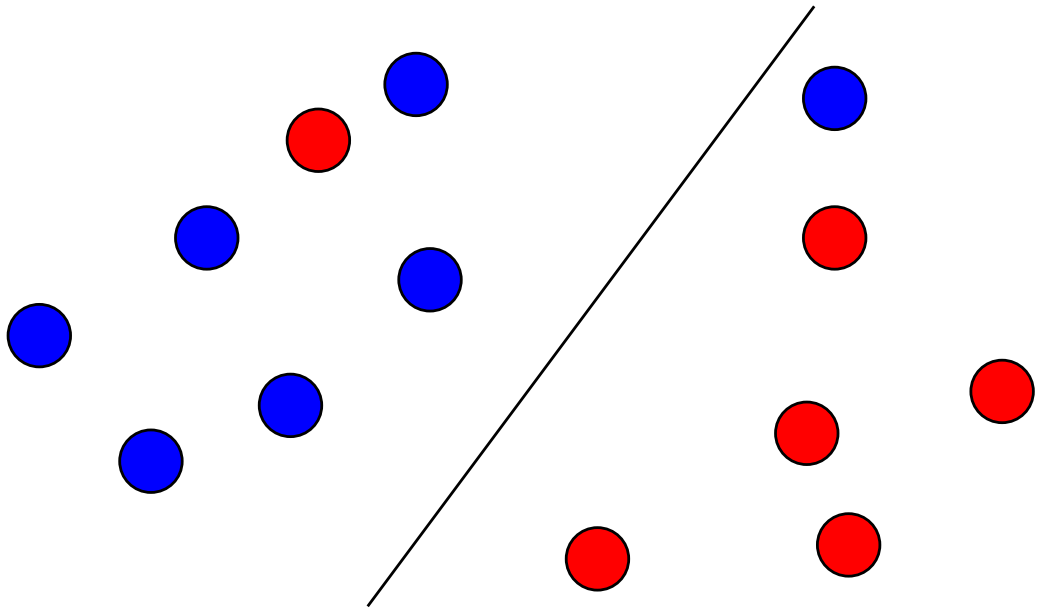
The non-linearly separable case

(when convex hulls intersect)

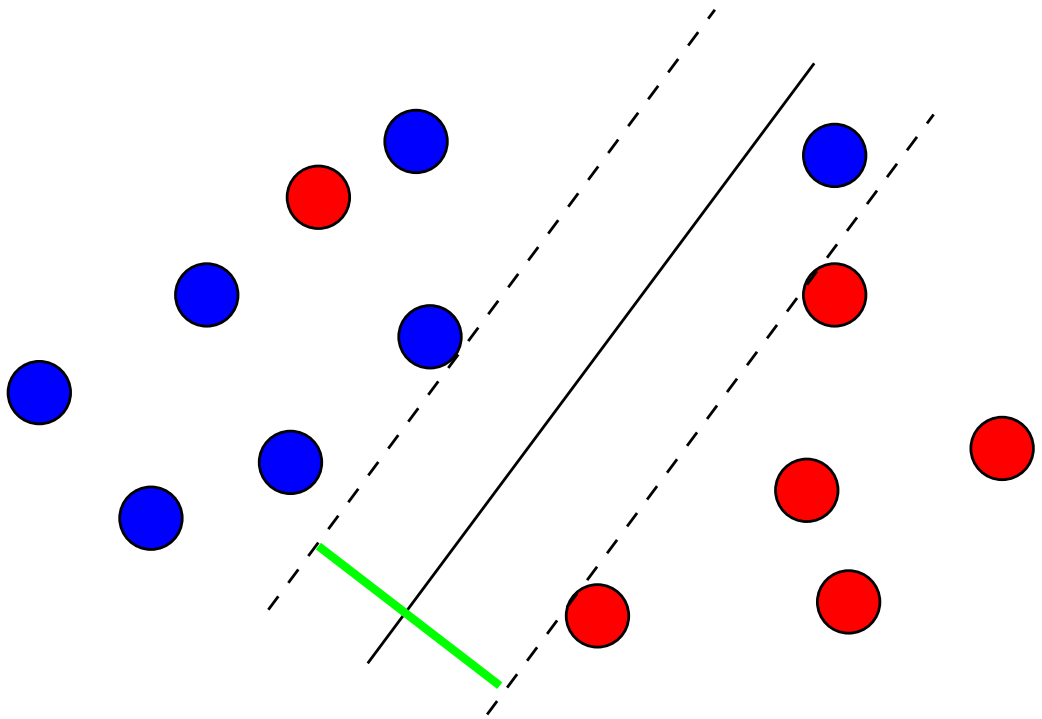
What happens when the data is not linearly separable?



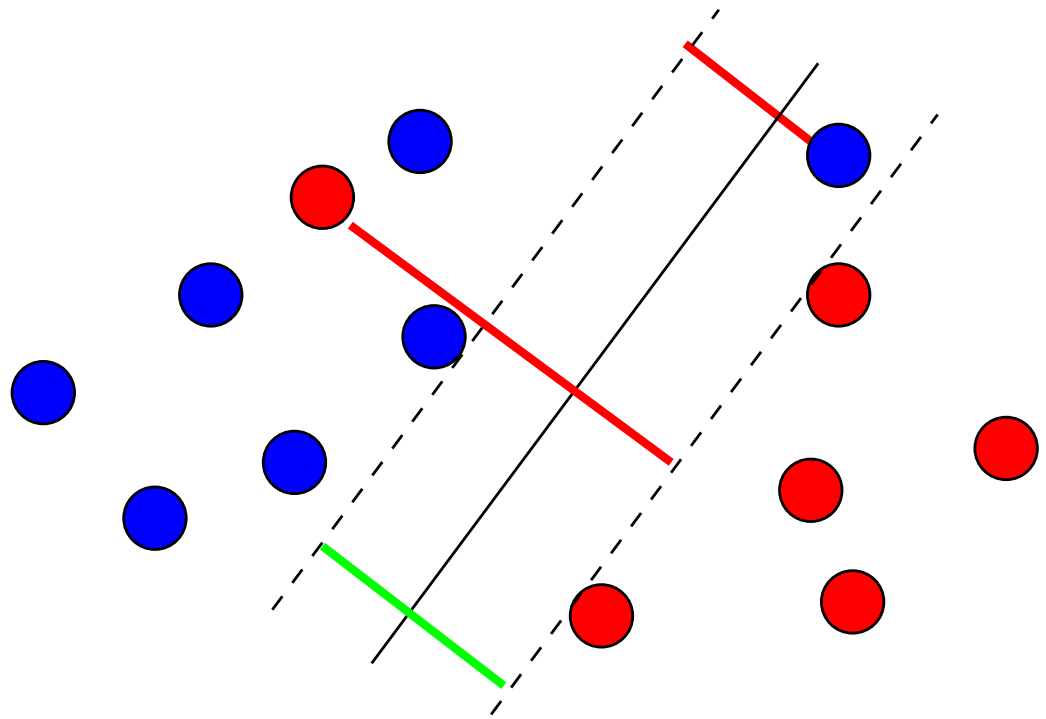
What happens when the data is not linearly separable?



What happens when the data is not linearly separable?



What happens when the data is not linearly separable?

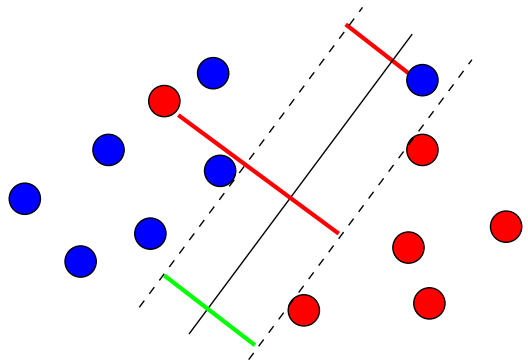


Soft-margin SVM ?

- Find a trade-off between **large margin** and **few errors**.
- Mathematically:

$$\min_f \left\{ \frac{1}{\text{margin}(f)} + C \times \text{errors}(f) \right\}$$

- C is a parameter

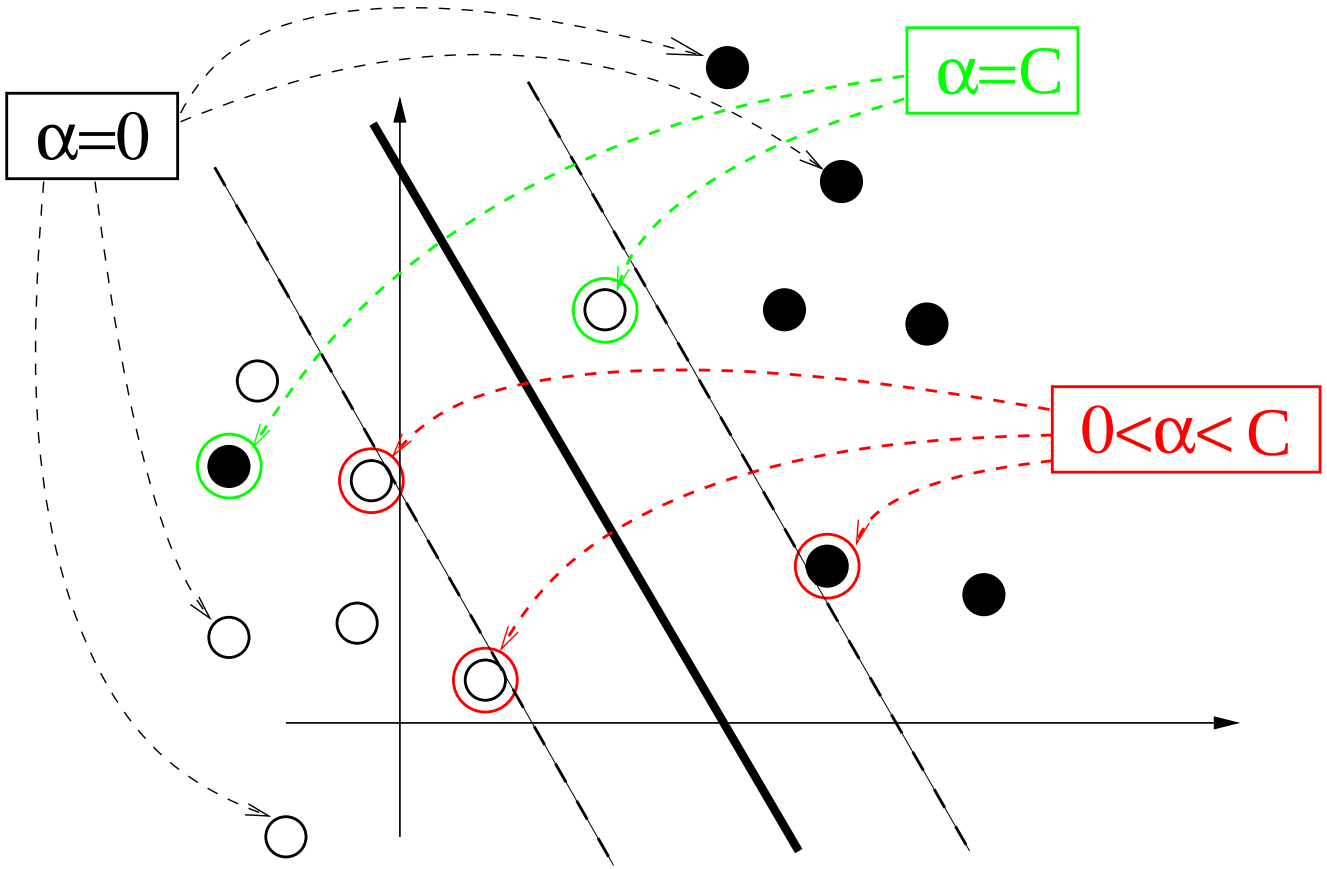


Dual formulation of soft-margin SVM

The **dual program** corresponding to this "softer" formulation is

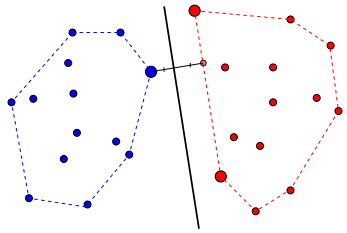
$$\begin{array}{ll} \text{maximize} & g(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{such that} & 0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n, \\ & \sum_{i=1}^{n} \alpha_i y_i = 0. \end{array}$$

Interpretation: bounded and unbounded support vectors

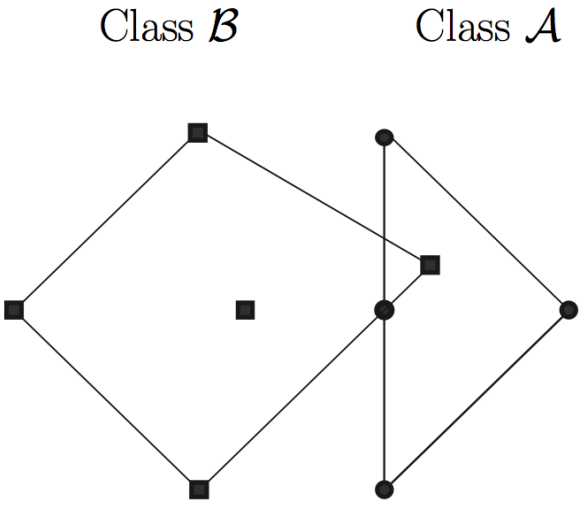


What about the convex hull analogy?

- Remember the separable case



- Here we consider the case where the two sets are not linearly separable, $i.e.$ their convex hulls **intersect.**



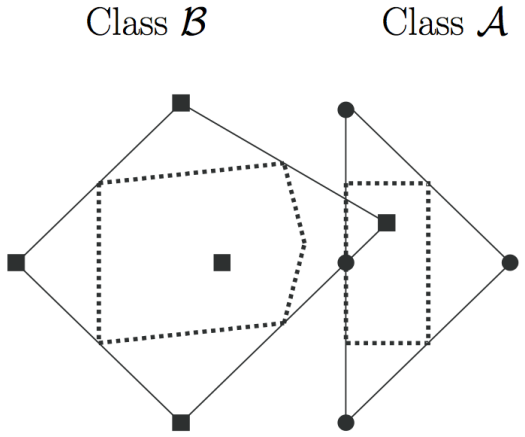
What about the convex hull analogy?

Definition 1. *Given a set of n points $\mathcal{A}$, and $0 \leq C \leq 1$, the set of finite combinations*

$$\sum_{i=1}^{n} \lambda_i \mathbf{x}_i, 1 \leq \lambda_i \leq C, \sum_{i=1}^{n} \lambda_i = 1,$$

is the (C) reduced convex hull of $\mathcal{A}$

- Using $C = 1/2$, the reduced convex hulls of $\mathcal{A}$ and $\mathcal{B}$,



- Soft-SVM with $C =$ closest two points of C-reduced convex hulls.

Images taken from *Duality and geometry in SVM classifiers*, Bennett and Bredensteiner

The Kernel Trick in SVM's

Kernel trick for SVM's

- use a mapping ϕ from $\mathcal{X}$ to a feature space,
- which corresponds to the **kernel** k:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

- Example: if $\phi(\mathbf{x}) = \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix}$, then

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle = (x_1)^2 (x_1')^2 + (x_2)^2 (x_2')^2.$$

Training a SVM in the feature space

Replace each $\mathbf{x}^T\mathbf{x}'$ in the SVM algorithm by $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle = k(\mathbf{x}, \mathbf{x}')$

- The dual problem becomes

$$g(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \, \alpha_j \, y_i \, y_j \, k(\mathbf{x}_i, \mathbf{x}_j),$$

under the constraints:

$$\begin{cases} 0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n \\ \sum_{i=1}^{n} \alpha_i y_i = 0. \end{cases}$$

- The **decision function** becomes:

$$\begin{aligned} f(\mathbf{x}) &= \langle \mathbf{w}, \phi(x) \rangle + b^* \\ &= \sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b^*. \end{aligned} \tag{1}$$

The Kernel Trick ?

The explicit computation of $\phi(\mathbf{x})$ is not necessary.
The kernel $k(\mathbf{x}, \mathbf{x}')$ is enough.

- the SVM optimization for α works **implicitly** in the feature space.
- the SVM is a kernel algorithm: only need to input $\boldsymbol{K}$ and $\boldsymbol{y}$:

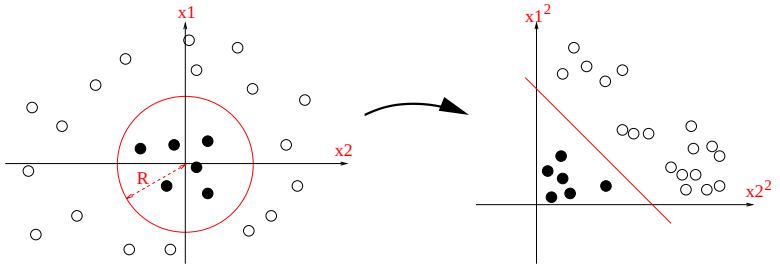
$$\begin{array}{ll} \text{maximize} & g(\alpha) = \alpha^T \mathbf{1} - \frac{1}{2}\alpha^T(\boldsymbol{K} \odot \mathbf{y}\mathbf{y}^T)\alpha \\ \text{such that} & 0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n \\ & \sum_{i=1}^n \alpha_i \boldsymbol{y_i} = 0. \end{array}$$

- $\boldsymbol{K}$'s **positive definite** $\Rightarrow \boldsymbol{K} \odot \mathbf{y}\mathbf{y}^T \Leftrightarrow$ **problem is convex**
- the decision function is $f(\cdot) = \sum_{i=1}^n \alpha_i \, \boldsymbol{k}(\mathbf{x}_i, \cdot) + b$.

Kernel example: polynomial kernel

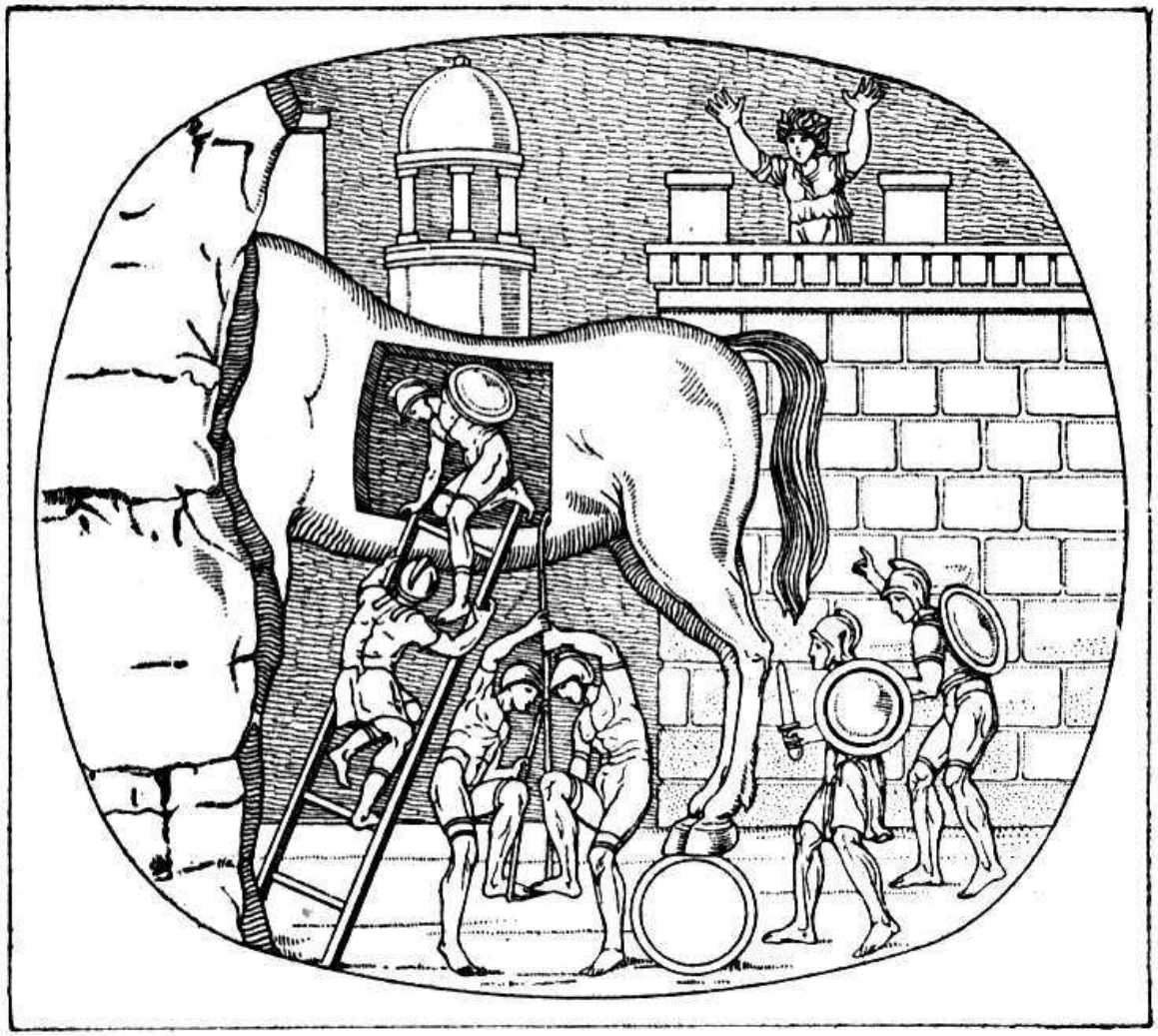
- For $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$, let $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$:

$$\begin{aligned} \boldsymbol{K(\mathbf{x}, \mathbf{x}')} &= x_1^2x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2 \\ &= (x_1x_1' + x_2x_2')^2 \\ &= \boldsymbol{\left(\mathbf{x}^T\mathbf{x}'\right)^2} . \end{aligned}$$



Kernels are Trojan Horses onto Linear Models

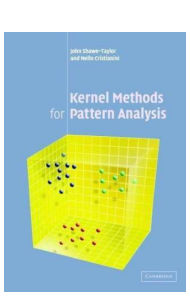
- With kernels, complex structures can enter the realm of linear models

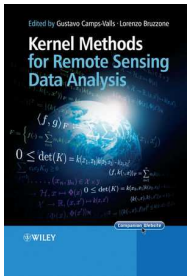


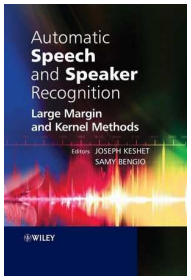
A few words about Kernel Methods

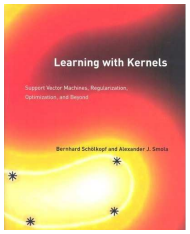
Kernel Methods

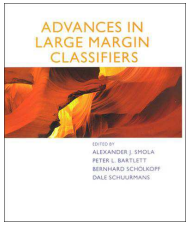
- Popular in machine learning now

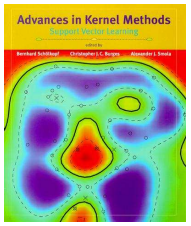


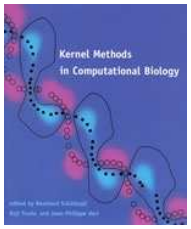


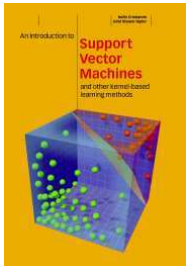












. . .

- Gained momentum in the late 90's with the support vector machine,
- Cross-disciplinary: Statistics, Optimization, Functional Analysis, Linear Algebra

A kernel on a set $\mathcal{X}$ is...

any function

$$\begin{array}{rccc} k: & \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}} & \longmapsto & \mathbb{R} \\ & (\mathbf{x}, \mathbf{y}) & \longrightarrow & k(\mathbf{x}, \mathbf{y}), \end{array}$$

which is **symmetric**

$$k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x}),$$

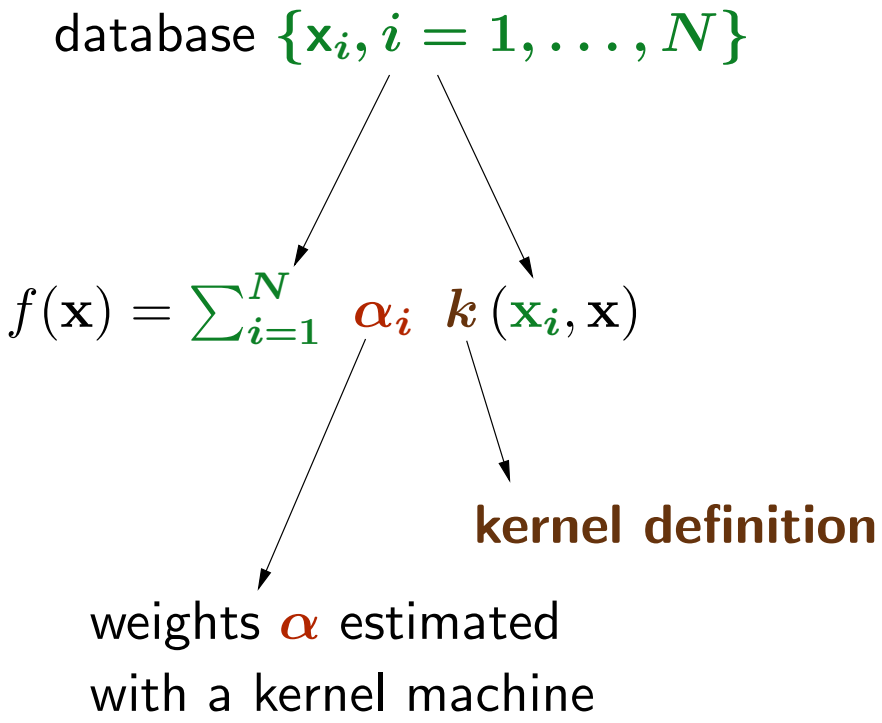
and **positive-definite**:

for any family of points $\mathbf{x}_1, \cdots, \mathbf{x}_n$ of $\mathcal{X}$, the matrix

$$K = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_i) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_i, \mathbf{x}_1) & \cdots & k(\mathbf{x}_i, \mathbf{x}_i) & \cdots & k(\mathbf{x}_i, \mathbf{x}_n) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_i) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \succeq 0$$

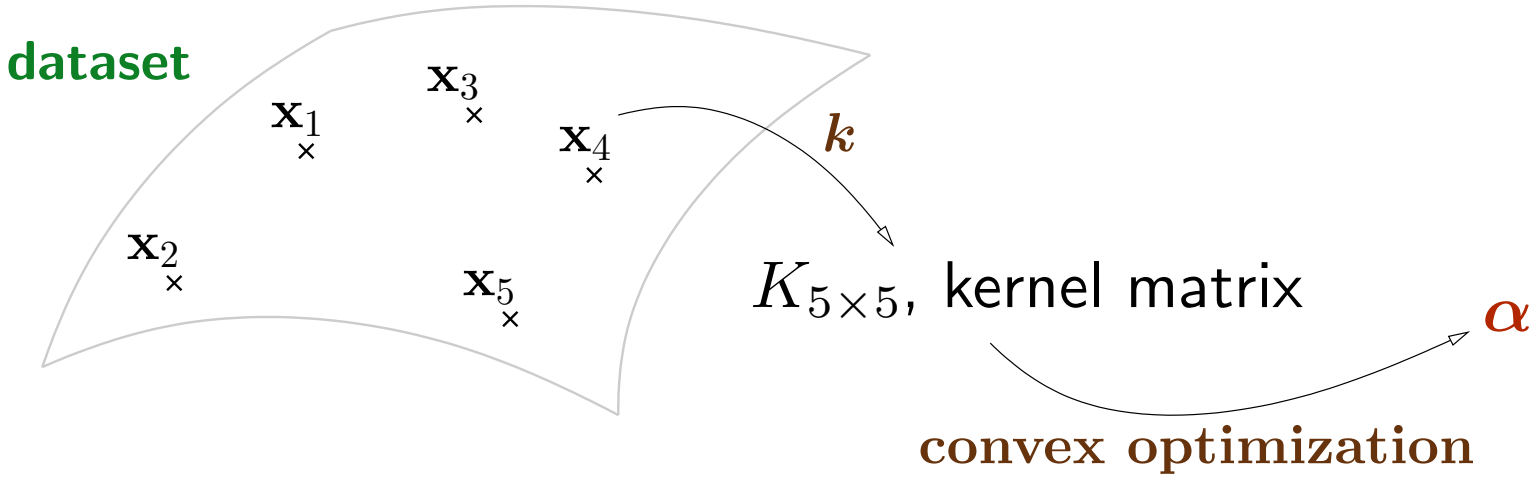
is positive (semi)definite (has nonnegative eigenvalues).

The general framework of kernel methods



Kernel methods **optimize** weights α to
avoid overfitting & **improve performance**
by using **convex optimization**

Positive Definiteness of $K \Rightarrow$ Convex Optimization



!! Important remark !!

convex optimization only works
because the kernel is **positive definite**

Kernels for Time Series

very Few Kernels on Time Series

Kernels for structured objects

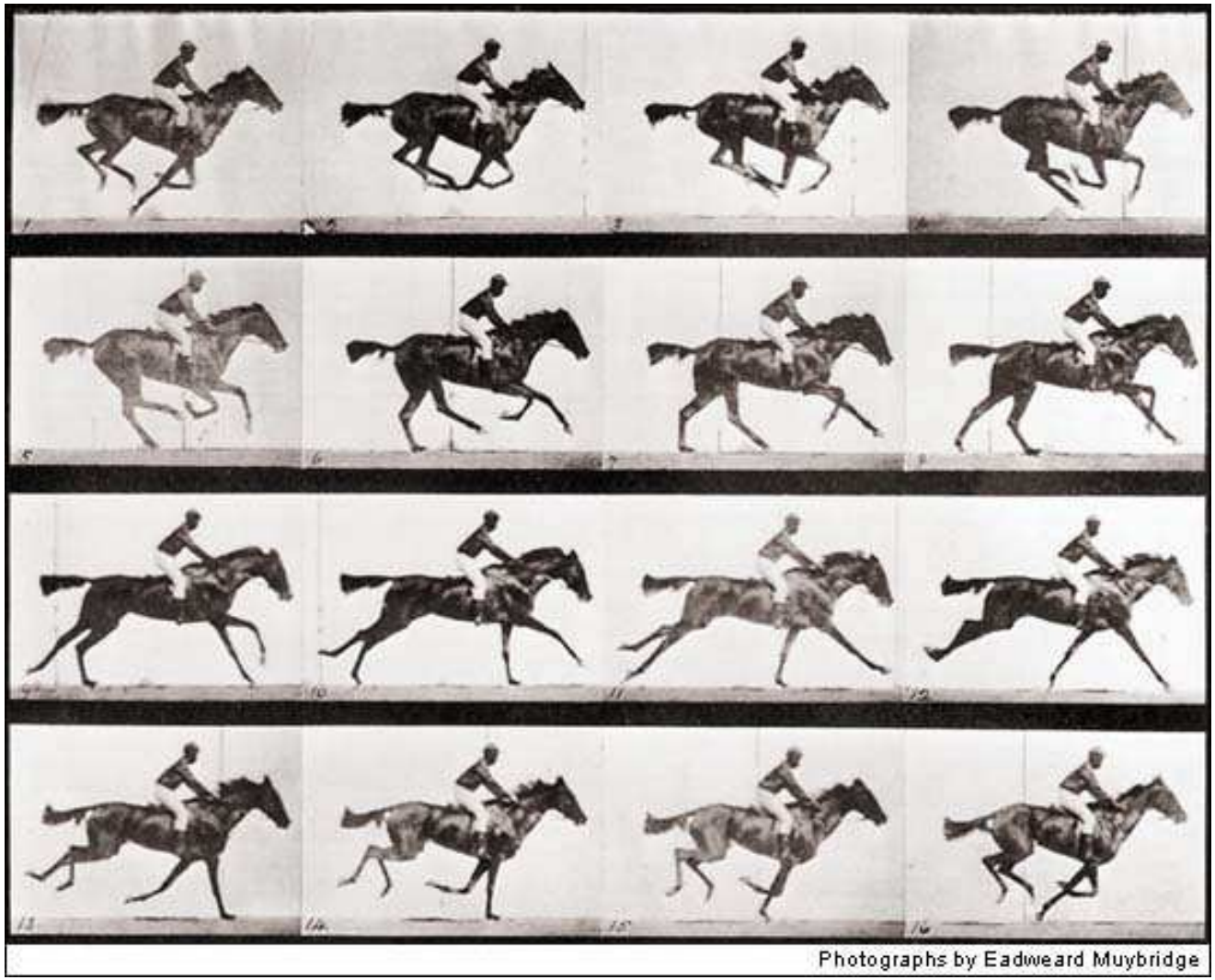
- Large literature:
  - Kernels for images,
  - Kernels for graphs,
  - Kernels for histograms, Bags-of-Words representations
  - Kernels for sequences: DNA, proteins: *discrete* symbols.

What about time-series?

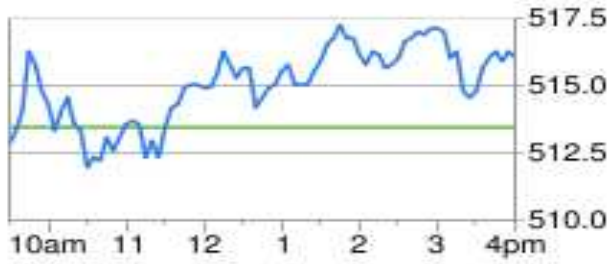
- **Important task**: **Ubiquitous** in science and engineering
- **Room for improvement**: very few proposals in literature so far

Time-series: a collection of objects indexed by time

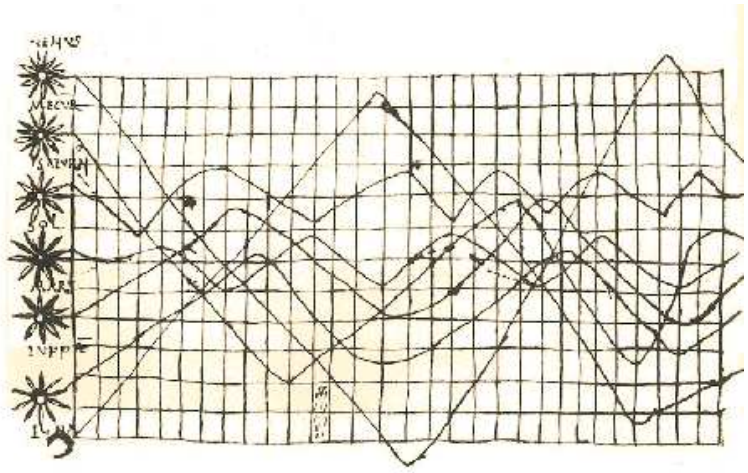
- Images



- **Univariate** time-series (google stock on a day)



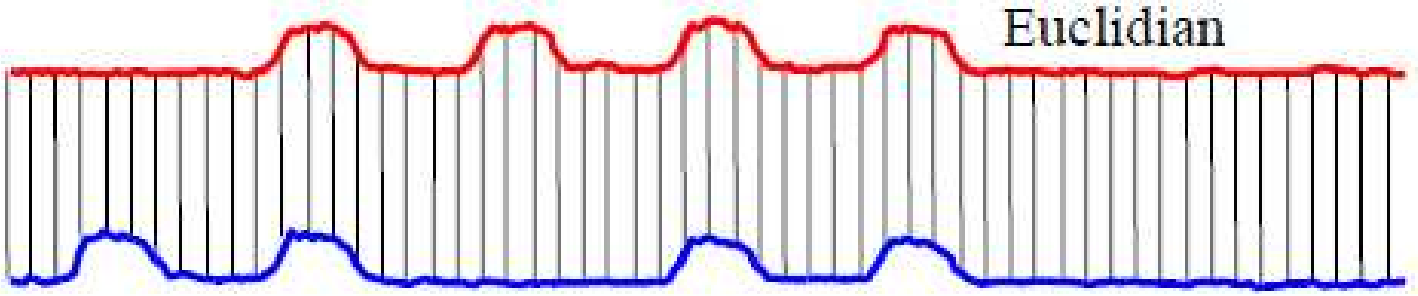
- **Multivariate** time-series (compiled by monks in 11th century!)



Objective: propose **positive definite** kernels between two time-series $\mathbf{x} = (x_1, \cdots, x_{\boldsymbol{n}})$ $\mathbf{y} = (y_1, \cdots, y_{\boldsymbol{m}})$ where the x_i, y_j belong to the same arbitrary set $\mathcal{X}$

Measuring similarity between time-series

- Time-series look like vectors, yet, in most cases:
  - neighboring coefficients x_i and x_{i+1} are not **independent** (smoothness)
  - **causality**: early observations x_i condition ulterior observations $x_{i+...}$
  - time-series in a dataset have **different lengths**.
- Even if we assume $n = m$, the Euclidean distance is blind to these subtleties:

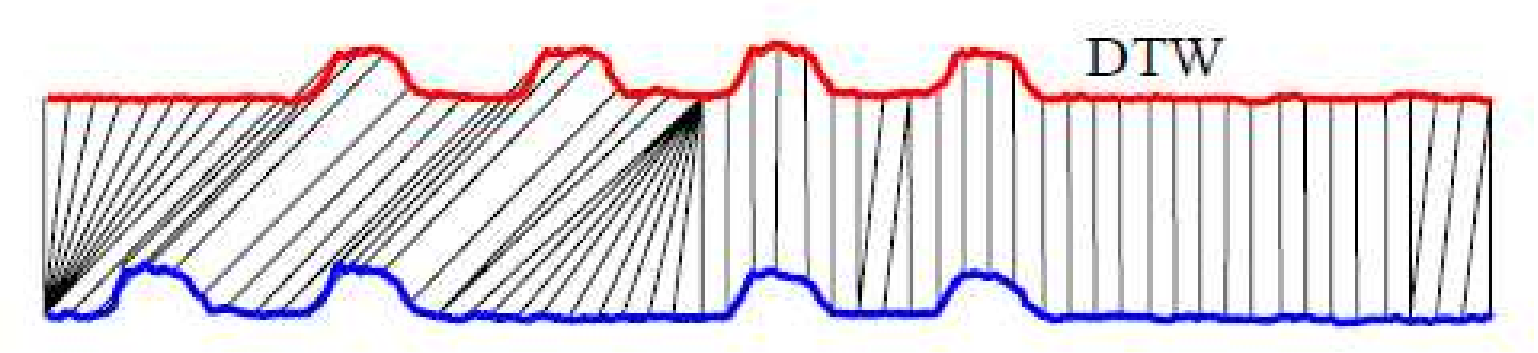


$$d_{\text{Euclide}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} d(x_i, y_i).$$

image taken from `http://www.markcorbyn.com`

Dynamic Time Warping (1971)

- First proposed by Japanese researchers in Japan: H. Sakoe & S. Chiba
- **Huge impact** in engineering: first in speech, now all domains of science
- **idea**: find a **good alignment** between **x** and **y** before computing d_{Euclide}.

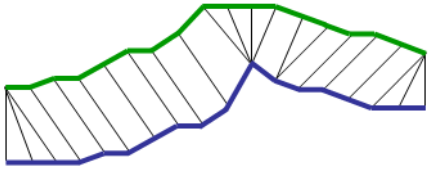


$$d_{\text{DTW}}(\mathbf{x},\mathbf{y}) = \min_{\pi\in\mathcal{A}(\mathbf{x},\mathbf{y})} \sum_{i=1} d\left(x_{\pi_1(i)}, y_{\pi_2(i)}\right) = \min_{\pi\in\mathcal{A}(\mathbf{x},\mathbf{y})} d_{\text{Euclide}}(\mathbf{x}_{\pi_1}, y_{\pi_2})$$

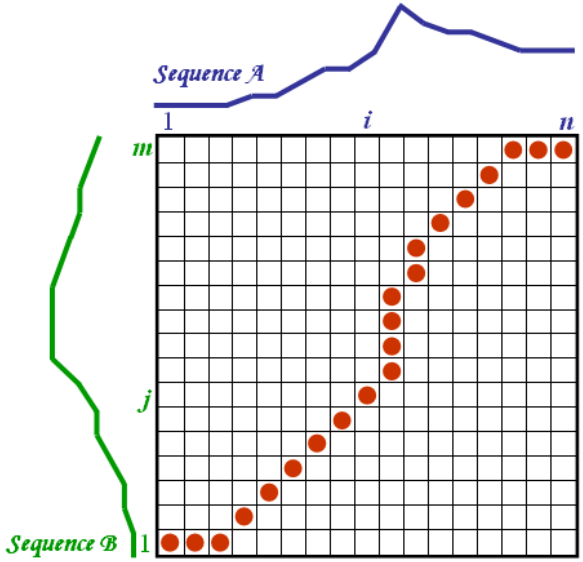
image taken from `http://www.markcorbyn.com`

Alignments

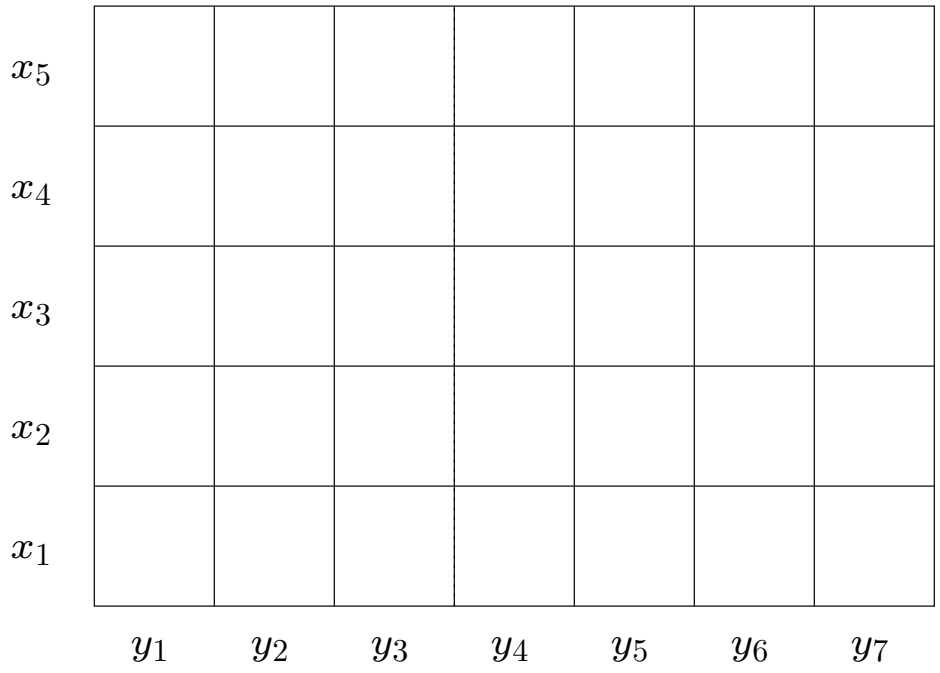
- Here are two sequence aligned



- An alignment is an increasing path on a grid.

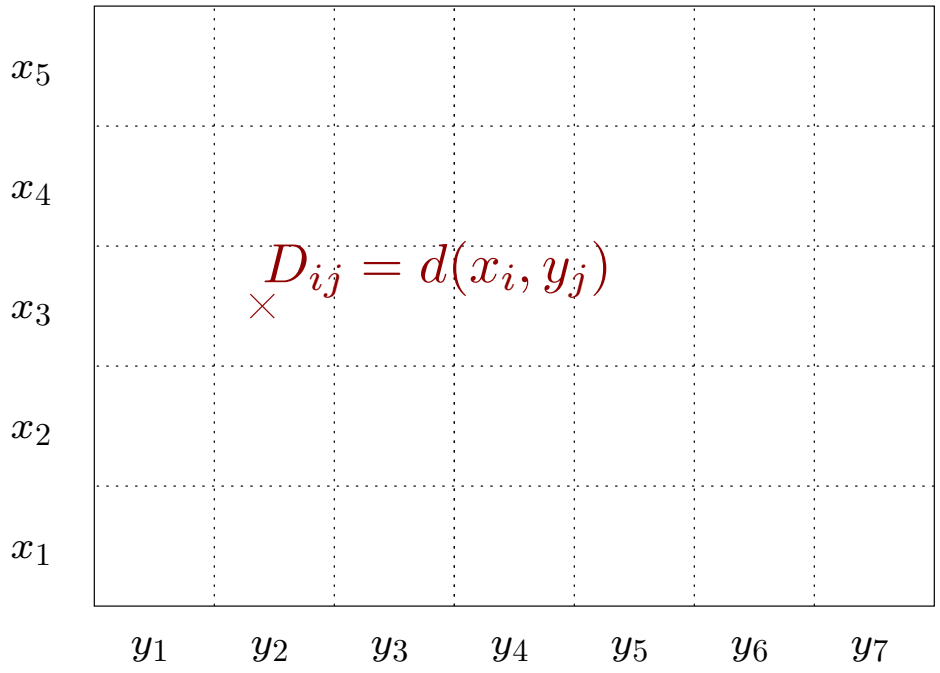


Optimal Alignment



We first "lay out" the $n \times m$ grid,
corresponding to $\mathbf{x} = (x_1, \cdots, x_5)$ $\mathbf{y} = (y_1, \cdots, y_7)$

Optimal Alignment



The grid is filled with pairwise distances.

Optimal Alignment

x_5	D_{51}	D_{52}	D_{53}	D_{54}	D_{55}	D_{56}	D_{57}
x_4	D_{41}	D_{42}	D_{43}	D_{44}	D_{45}	D_{46}	D_{47}
x_3	D_{31}	D_{32}	D_{33}	D_{34}	D_{35}	D_{36}	D_{37}
x_2	D_{21}	D_{22}	D_{23}	D_{24}	D_{25}	D_{26}	D_{27}
x_1	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}	D_{17}
	y_1	y_2	y_3	y_4	y_5	y_6	y_7

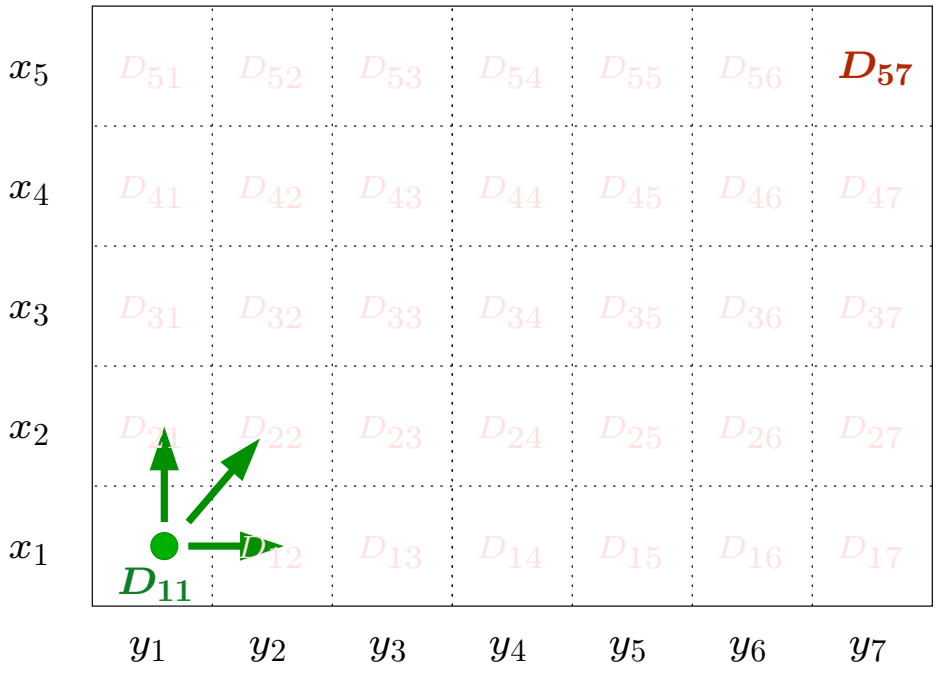
This rectangular matrix is the only thing we need.

Optimal Alignment

x_5	D_{51}	D_{52}	D_{53}	D_{54}	D_{55}	D_{56}	$\boldsymbol{D_{57}}$
x_4	D_{41}	D_{42}	D_{43}	D_{44}	D_{45}	D_{46}	D_{47}
x_3	D_{31}	D_{32}	D_{33}	D_{34}	D_{35}	D_{36}	D_{37}
x_2	D_{21}	D_{22}	D_{23}	D_{24}	D_{25}	D_{26}	D_{27}
x_1	$\boldsymbol{D_{11}}$	D_{12}	D_{13}	D_{14}	D_{15}	D_{16}	D_{17}
	y_1	y_2	y_3	y_4	y_5	y_6	y_7

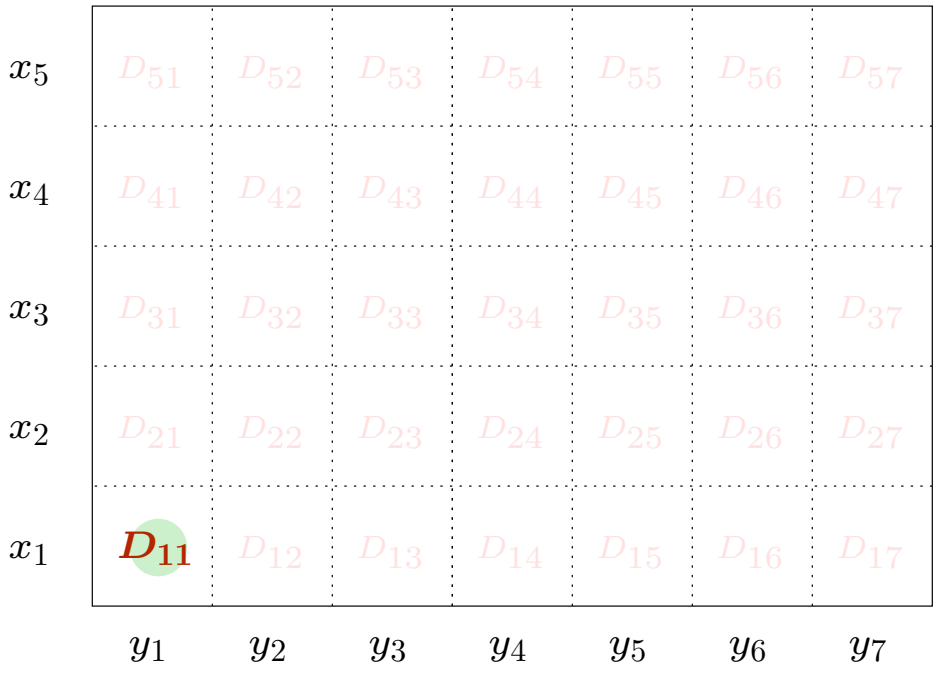
An alignment is a path that starts from $\mathbf{(1,1)}$ to reach $\mathbf{(5,7)}$

Optimal Alignment



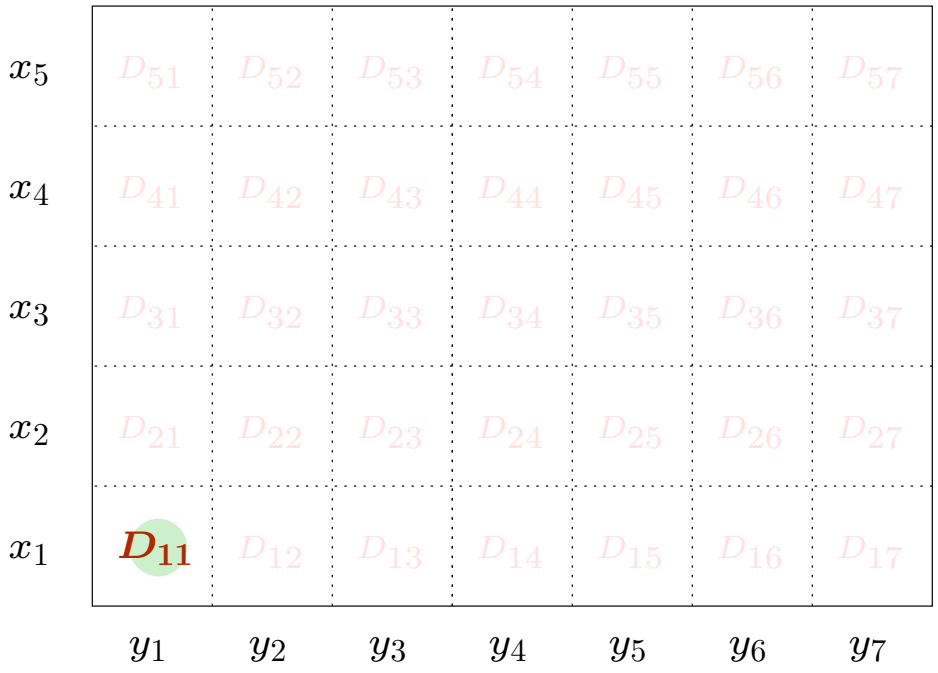
The only admissible moves from one cell to the next are $\rightarrow, \uparrow$ and $\nearrow$

Optimal Alignment



The cost of a path is the sum of contributions D_{ij} it walks through.

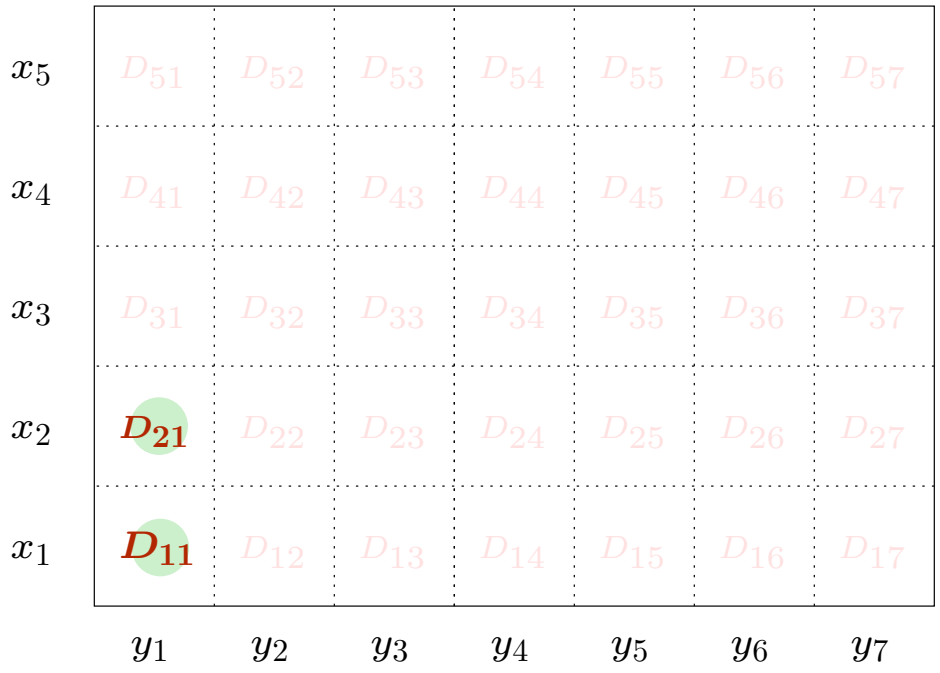
Optimal Alignment



So far,

$$C = \boldsymbol{D_{11}}.$$

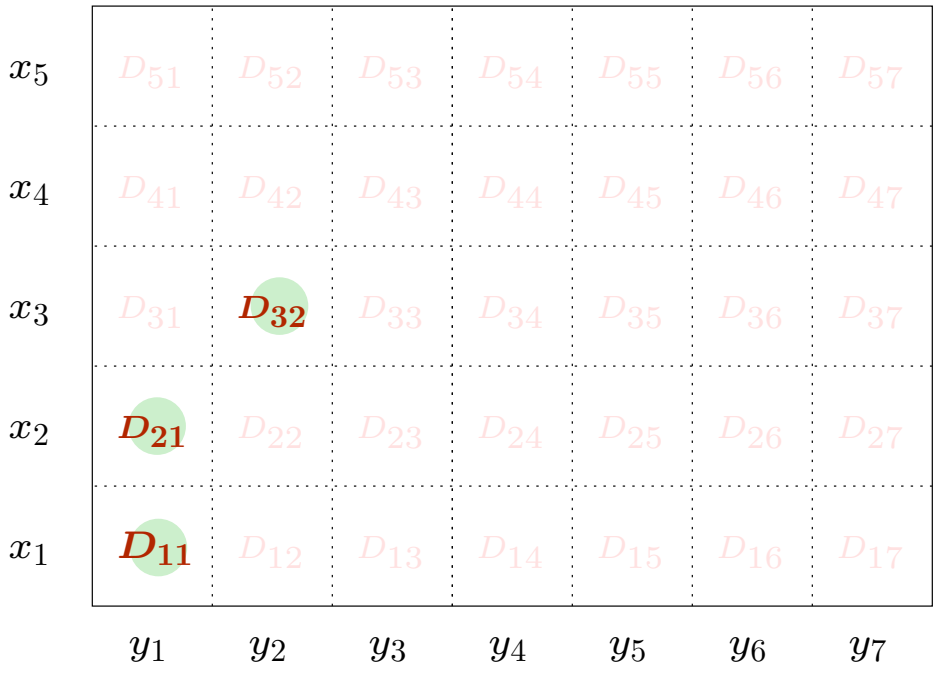
Optimal Alignment



Moving up,

$$C = D_{11} + \boldsymbol{D_{21}}.$$

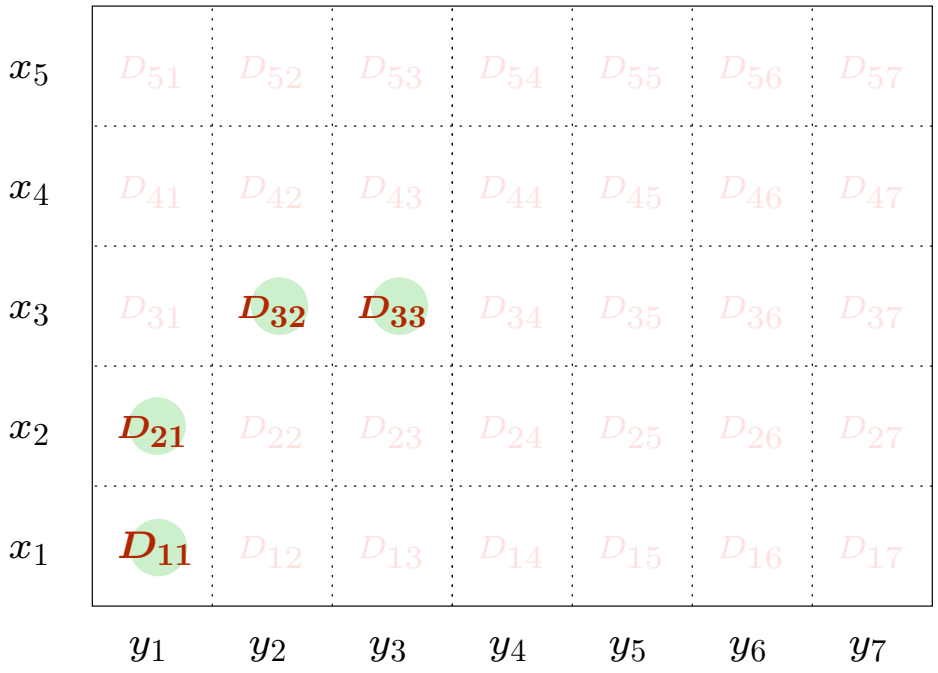
Optimal Alignment



Moving diagonally,

$$C = D_{11} + D_{21} + \boldsymbol{D_{32}}.$$

Optimal Alignment



Moving right,

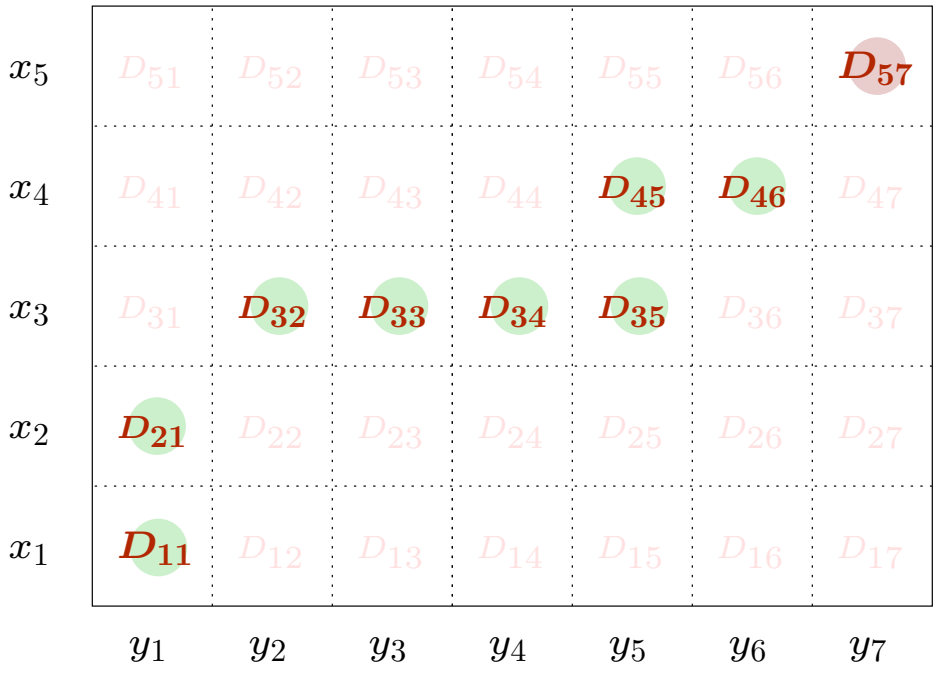
$$C = D_{11} + D_{21} + D_{32} + \boldsymbol{D_{33}}.$$

Optimal Alignment

etc., until we reach the upper right corner

$$C = D_{11} + D_{21} + D_{32} + D_{33} + D_{34} + D_{35} + D_{45} + D_{46} + \boldsymbol{D_{57}}.$$

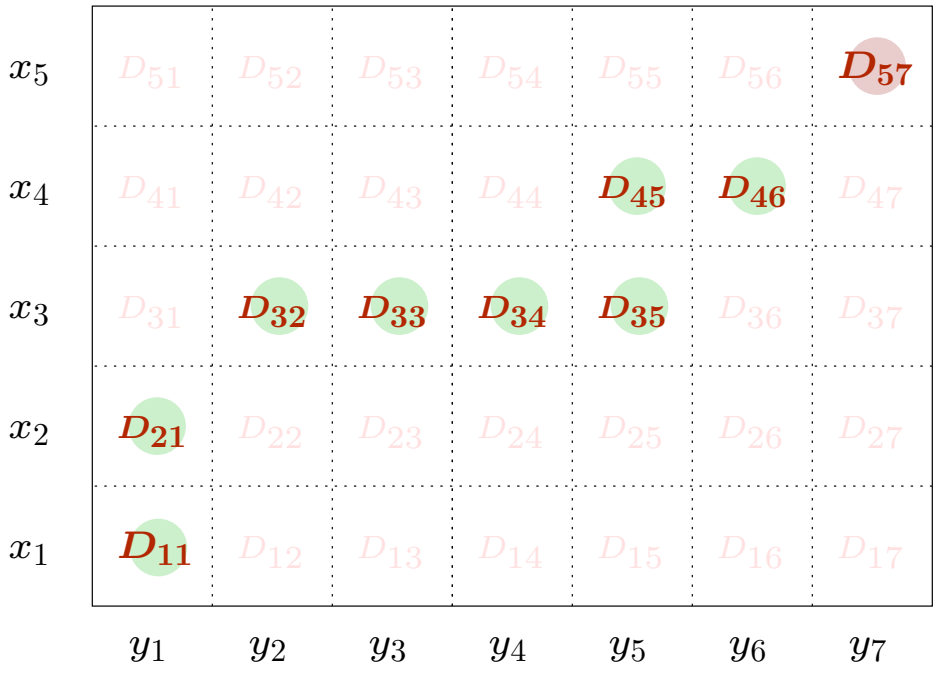
Optimal Alignment



A path is uniquely defined by 2 rows vectors:

$$C = D_{11} + D_{21} + D_{32} + D_{33} + D_{34} + D_{35} + D_{45} + D_{46} + D_{57}.$$

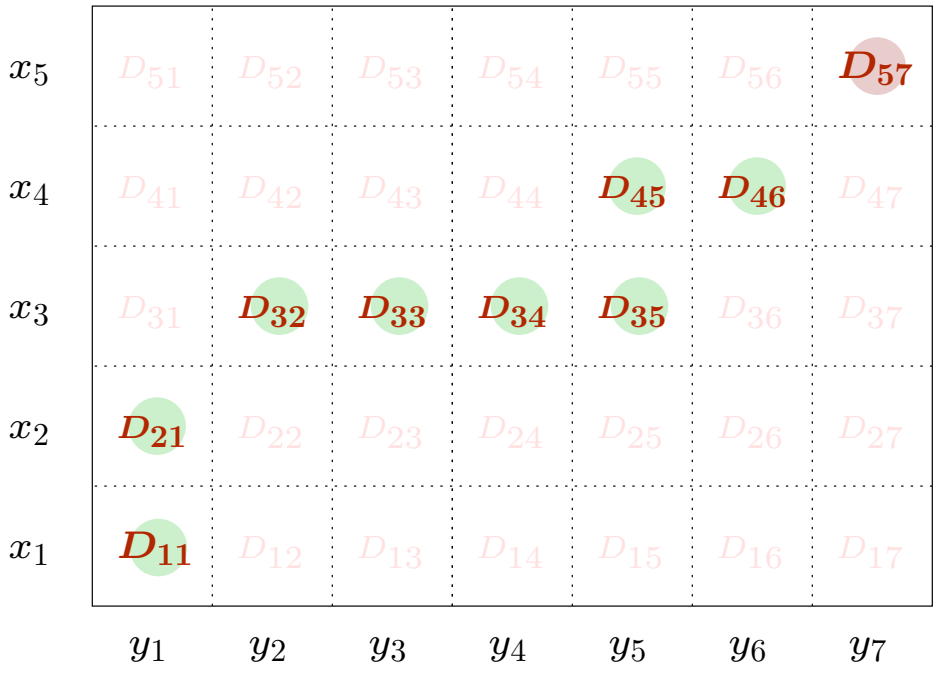
Optimal Alignment



A path is uniquely defined by 2 rows vectors:

$$\pi = \begin{pmatrix} \pi_1 \\ \pi_2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 3 & 3 & 3 & 4 & 4 & 5 \\ 1 & 1 & 2 & 3 & 4 & 5 & 5 & 6 & 7 \end{pmatrix}$$

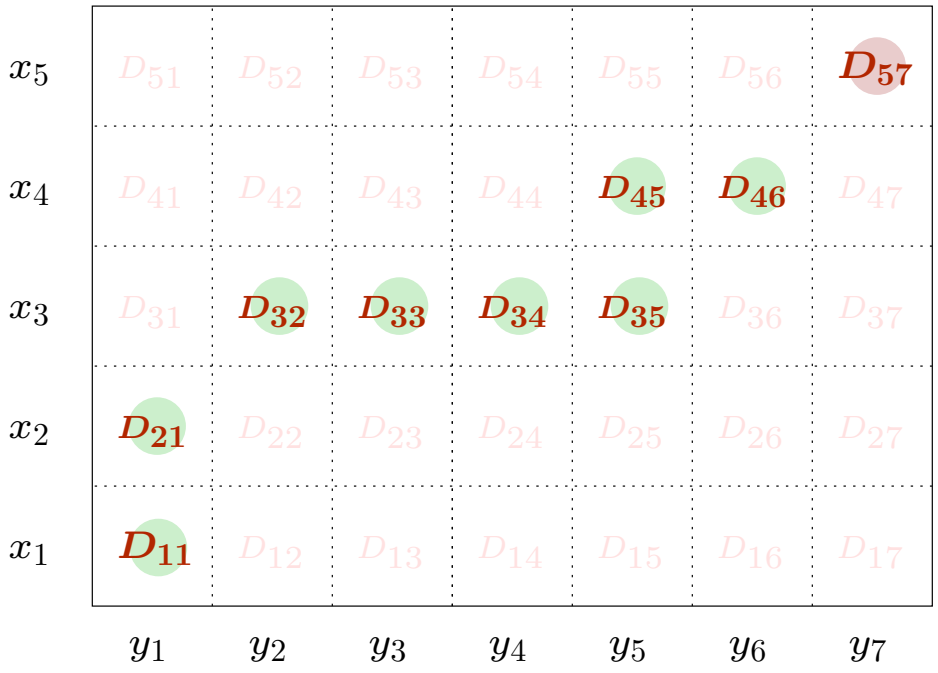
Optimal Alignment



Given a path π, we call $C(\pi)$ the sum of distances:

$$C(\pi) = D_{11} + D_{21} + D_{32} + D_{33} + D_{34} + D_{35} + D_{45} + D_{46} + D_{57}.$$

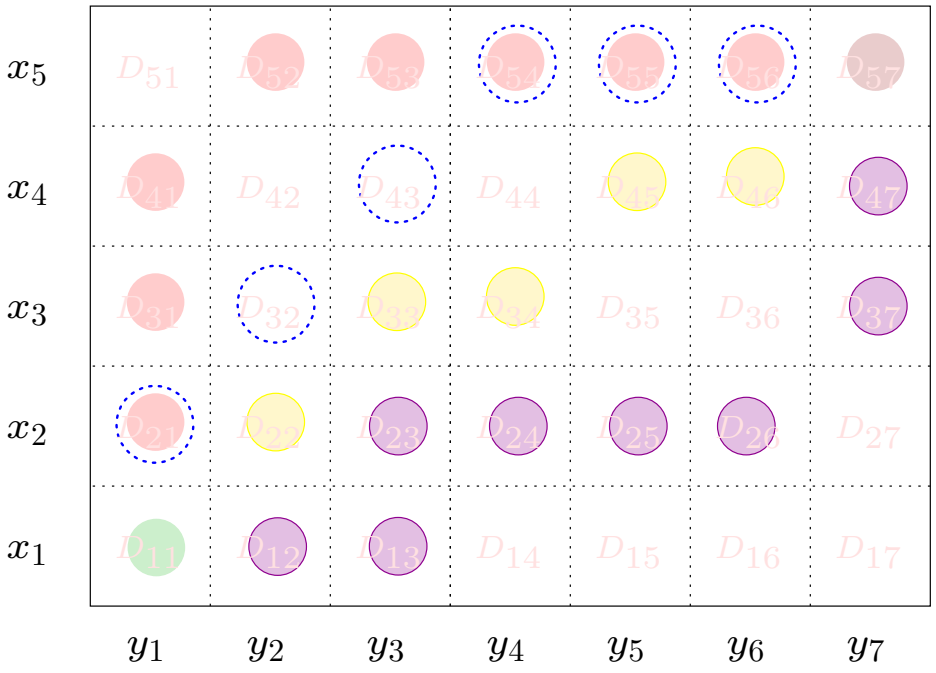
Optimal Alignment



We defined the distance d_{DTW} as

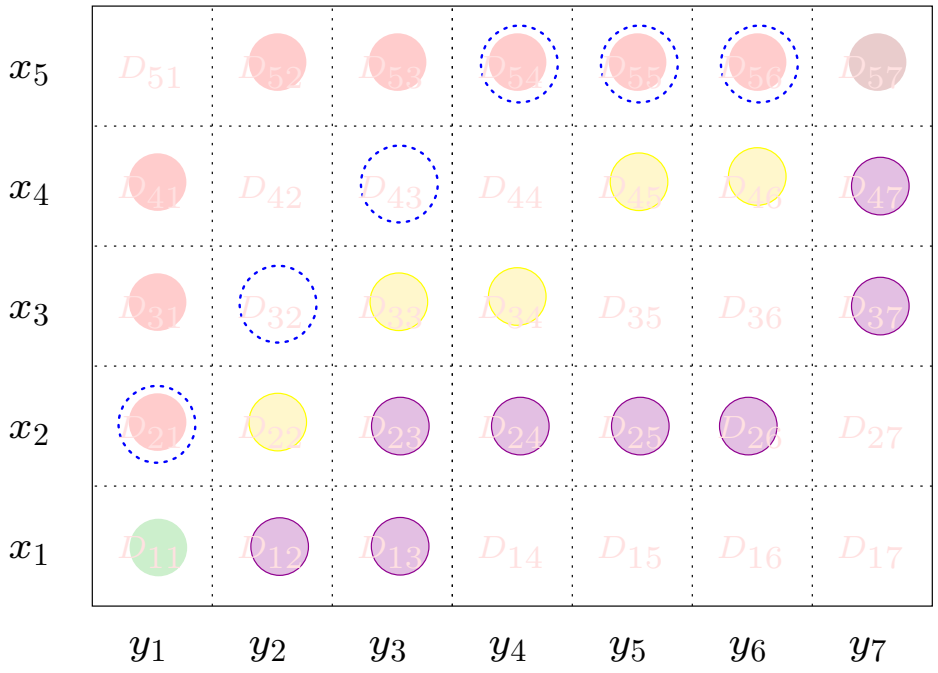
$$d_{\mathsf{DTW}}(\mathbf{x},\mathbf{y}) = \min_{\pi \in \mathcal{A}(\mathbf{x},\mathbf{y})} \sum_{i=1}^{|pi|} d\left(x_{\boldsymbol{\pi}_1(i)}, y_{\boldsymbol{\pi}_2(i)}\right) = \min_{\pi \in \mathcal{A}(\mathbf{x},\mathbf{y})} C_{\mathbf{x},\mathbf{y}}(\pi).$$

Optimal Alignment



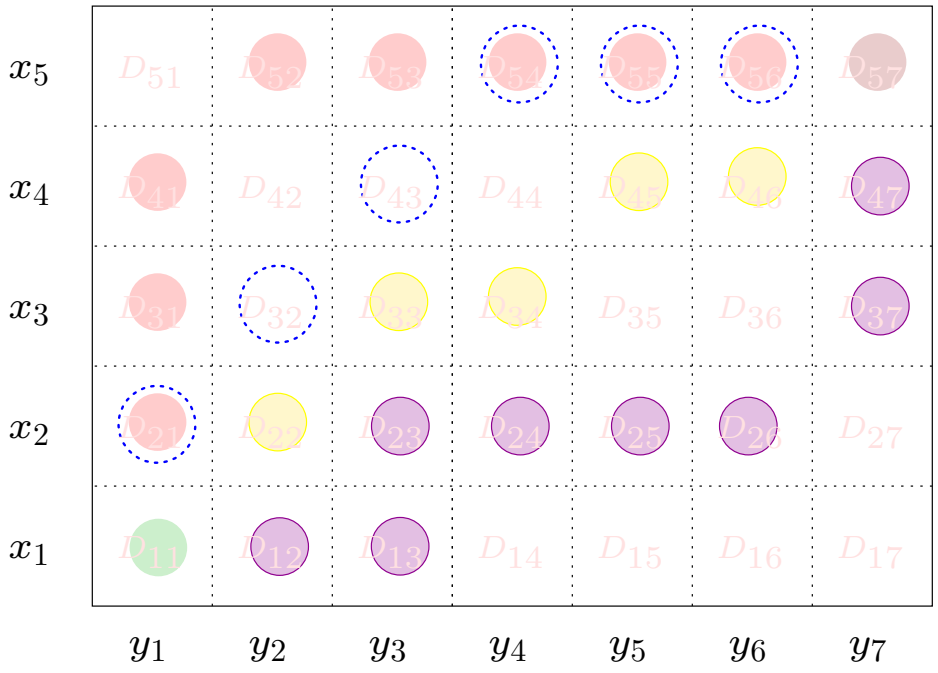
$\mathcal{A}(\mathbf{x}, \mathbf{y}) \Leftrightarrow$ the set of all paths on this grid.
Only depends on the $|\mathbf{x}|$ **and** $|\mathbf{y}|$, 5 and 7 here.

Optimal Alignment



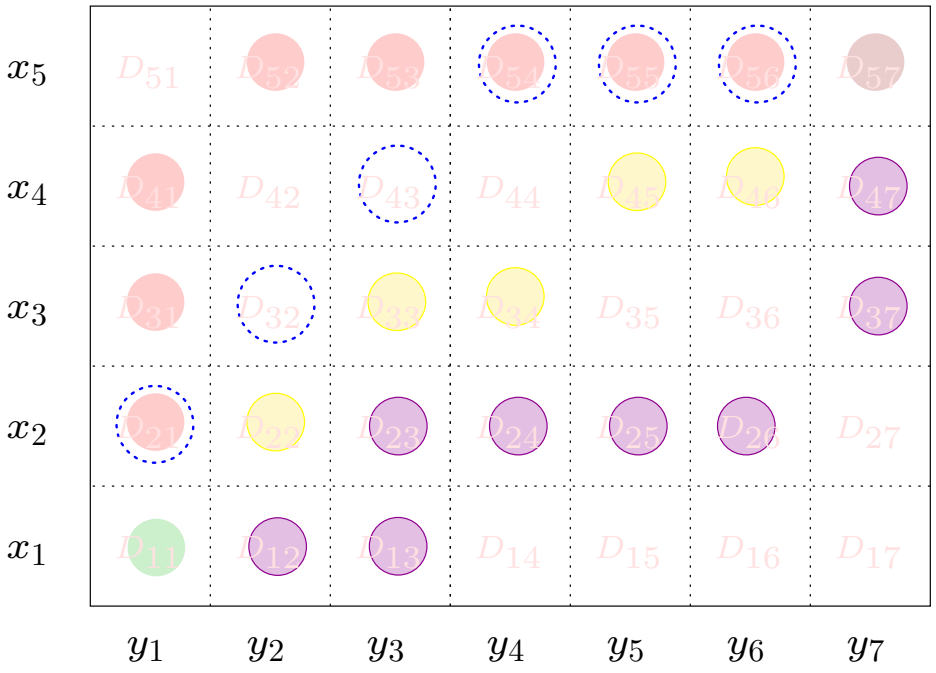
To clarify this, we write $\mathcal{A}(|\mathbf{x}|, |\mathbf{y}|)$ for the set of all alignments between **x** and **y**.

Optimal Alignment



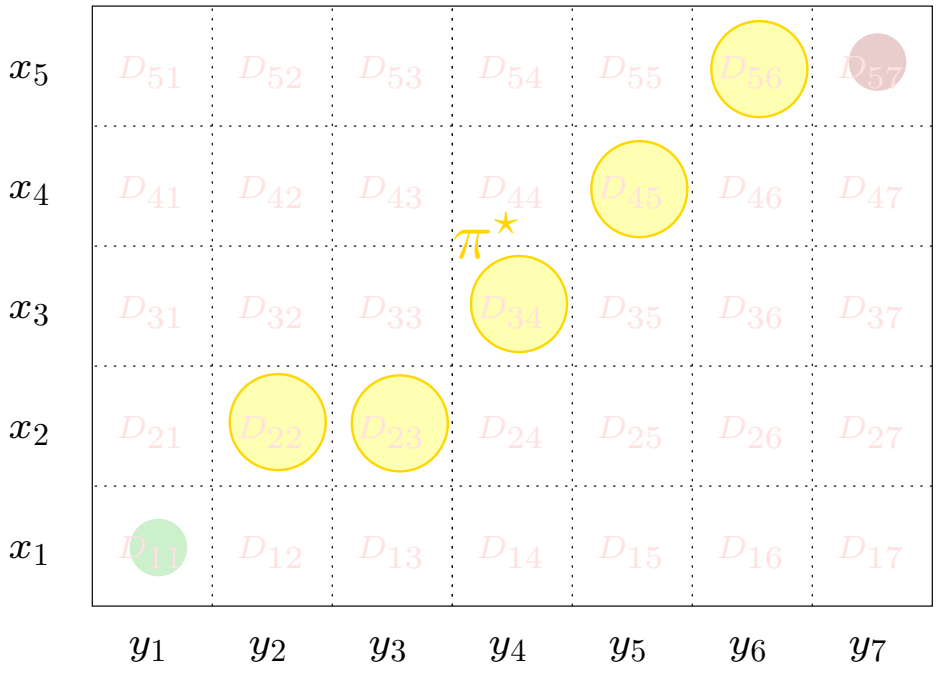
$\operatorname{card} \mathcal{A}(n, m)$ is equal to the **Delannoy number** *Delannoy*(n, m).

Optimal Alignment



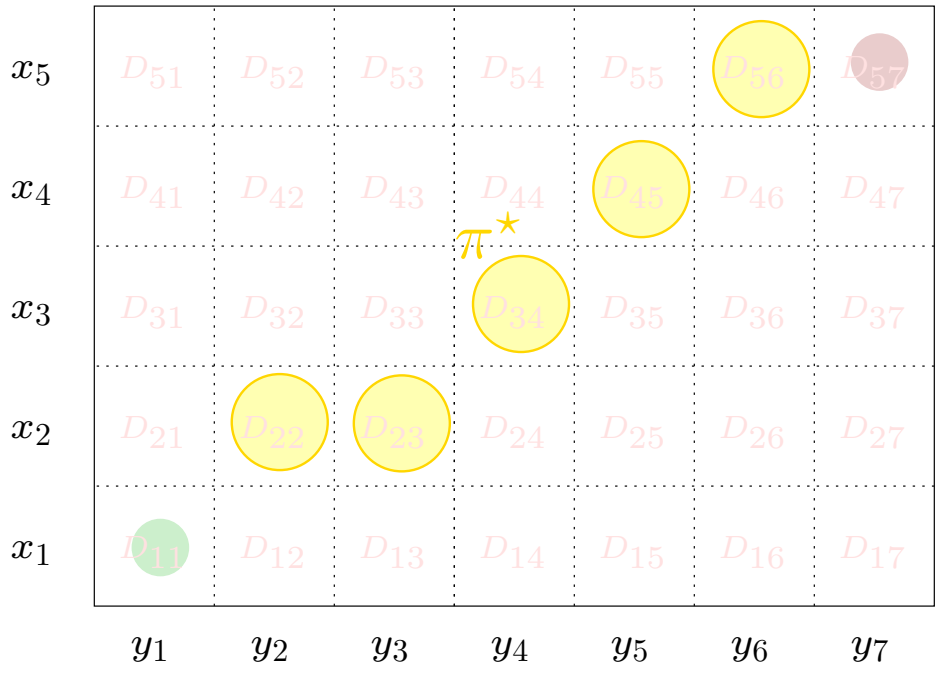
$$\mathit{Delannoy}(5,7) = 2241$$
$$\vdots$$
$$\mathit{Delannoy}(20,20) = 4.53e+13$$

Optimal Alignment



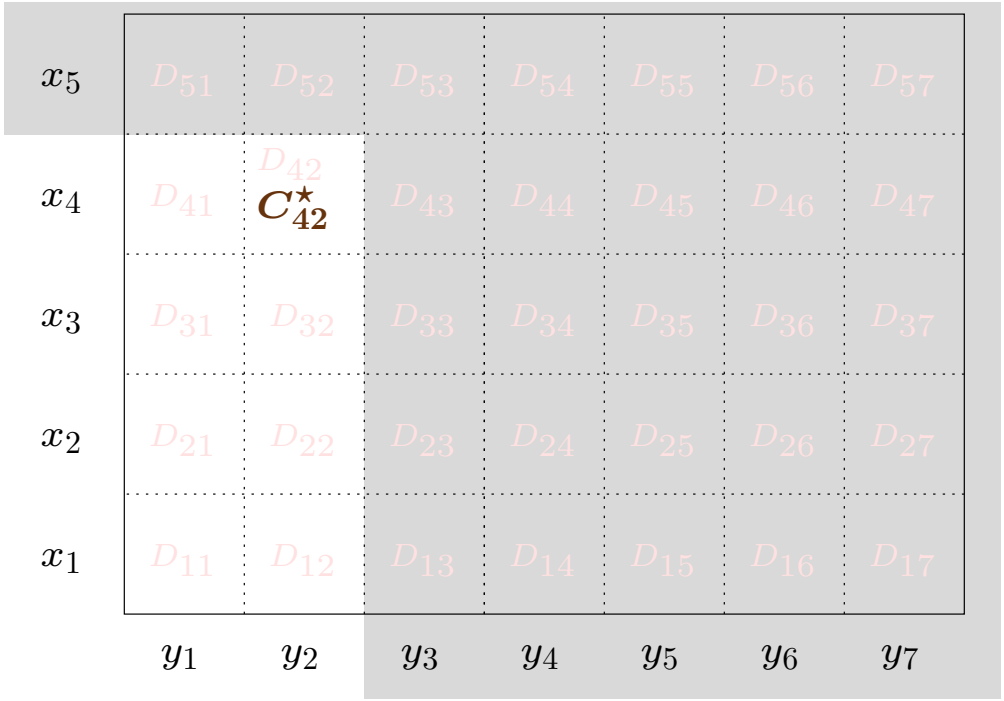
DTW finds the minimum among all paths: **discrete optimization**.
Obviously, checking each would be **computationally intractable.**

Optimal Alignment



Key idea: use **Bellman's Dynamic programming**

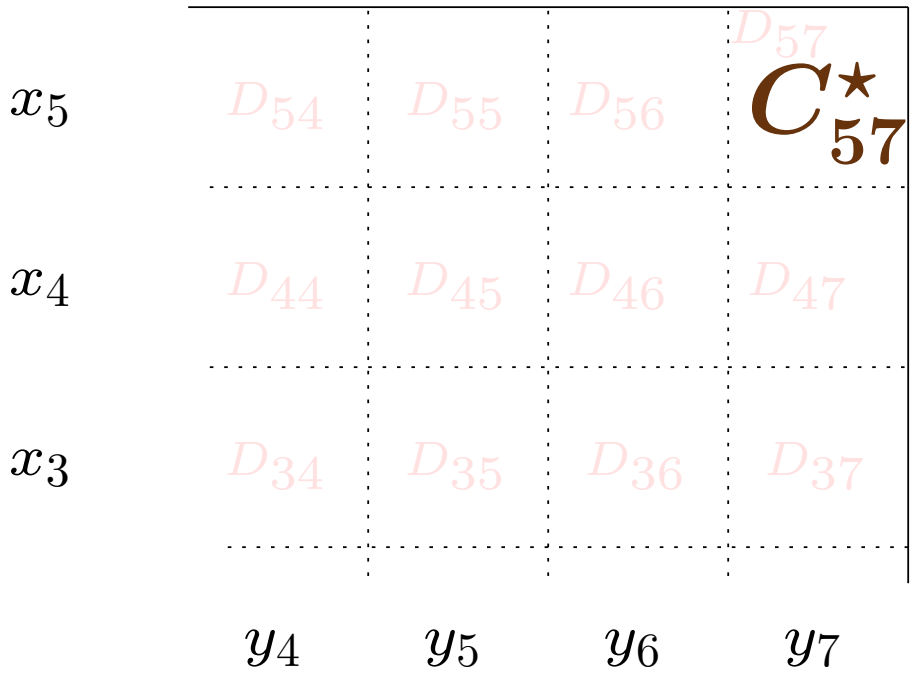
Optimal Alignment



Define $C^{\star}_{ij}$ as the **cost of the optimal sub-path** up to the i-th symbol of **x** and the j-th symbol of **y**.

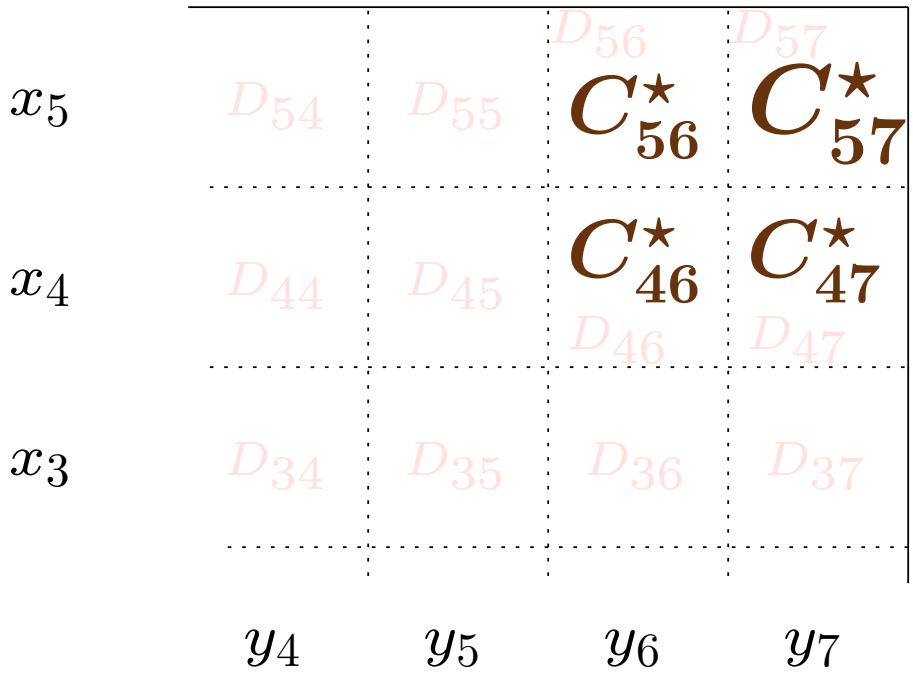
$$C^{\star}_{ij} = \min_{\pi \in \mathcal{A}(i,j)} C_{\mathbf{x}_1^i, y_1^j}(\pi).$$

Optimal Alignment



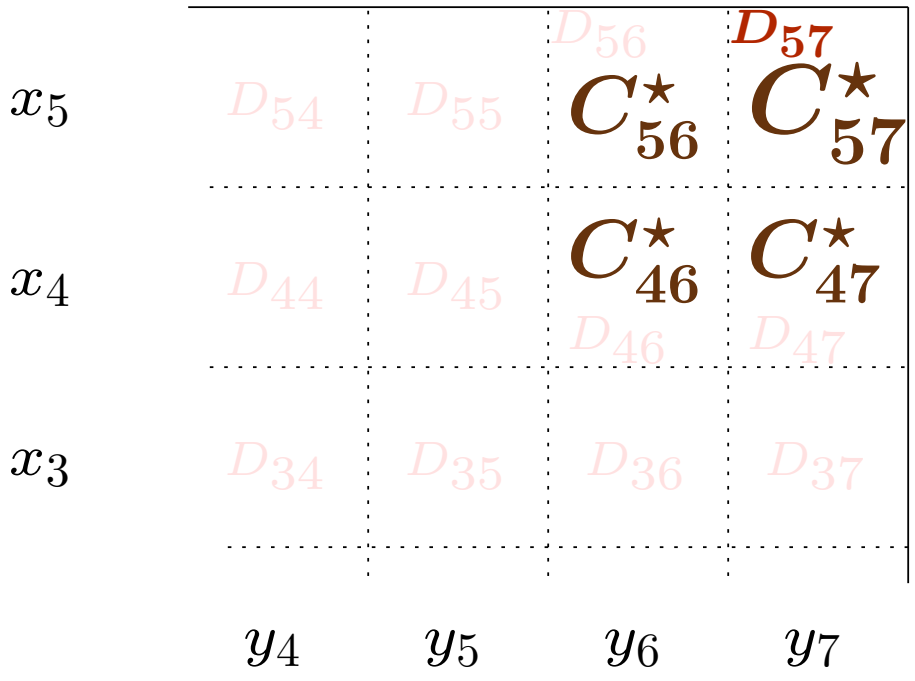
Obviously $C^{\star}_{57}$ **is** the quantity we want to compute.

Optimal Alignment



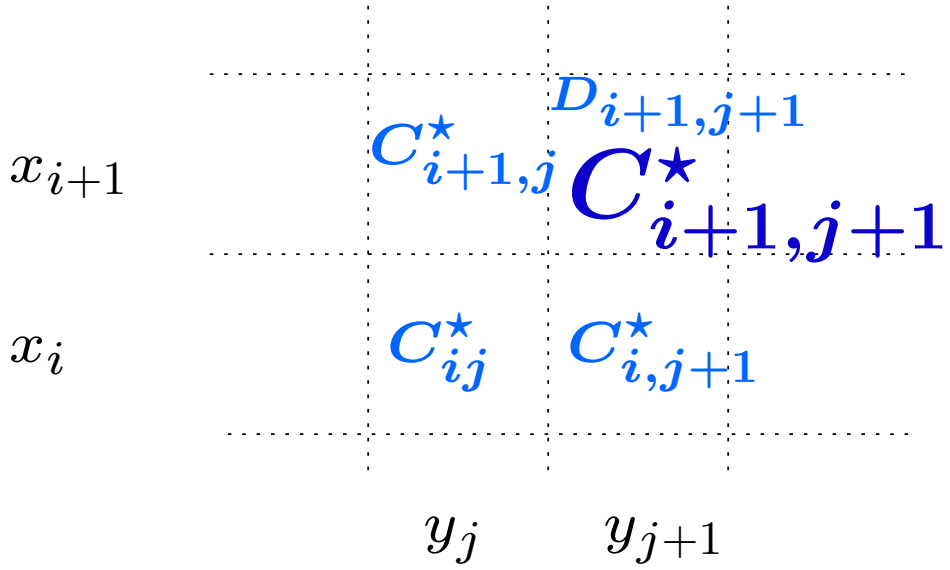
Relationship between $C^\star_{57}$ its neighbours $C^\star_{56}$, $C^\star_{46}$, $C^\star_{47}$?

Optimal Alignment



$$C^\star_{57} = \min(C^\star_{56}, C^\star_{46}, C^\star_{47}) + D_{57}$$

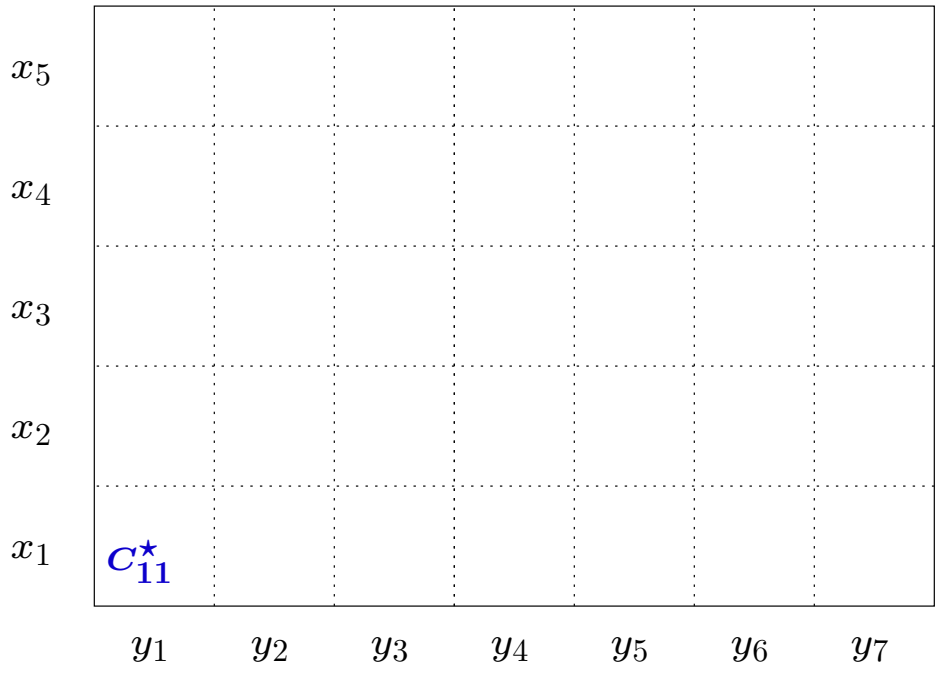
Optimal Alignment



More generally, for all $i \leq n-1, j \leq m-1$,

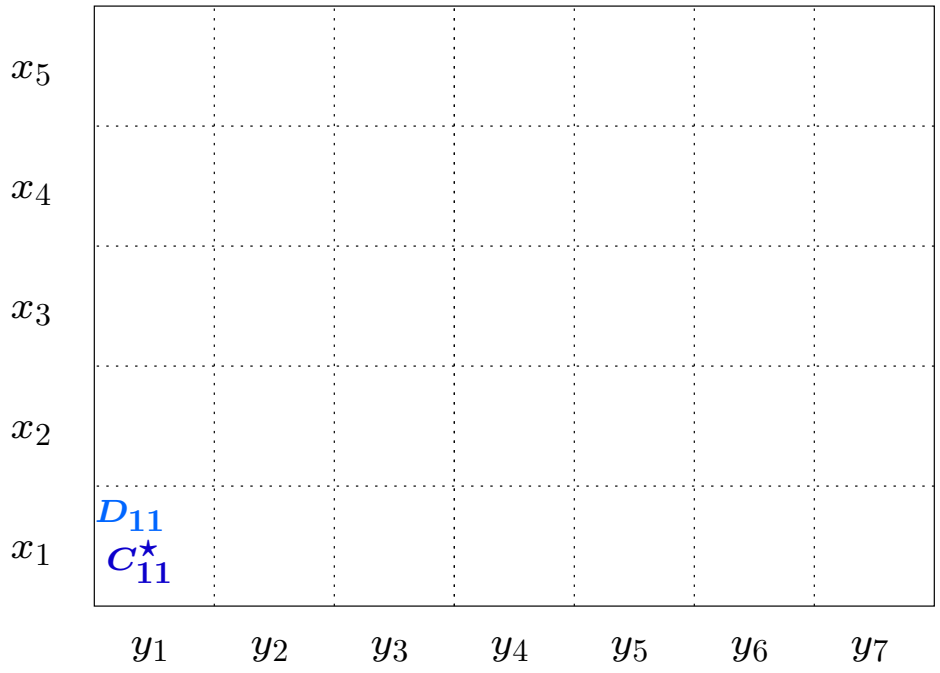
$$C^\star_{i+1,j+1} = \min(C^\star_{i+1,j}, C^\star_{ij}, C^\star_{i,j+1}) + D_{i+1,j+1}$$

Optimal Alignment



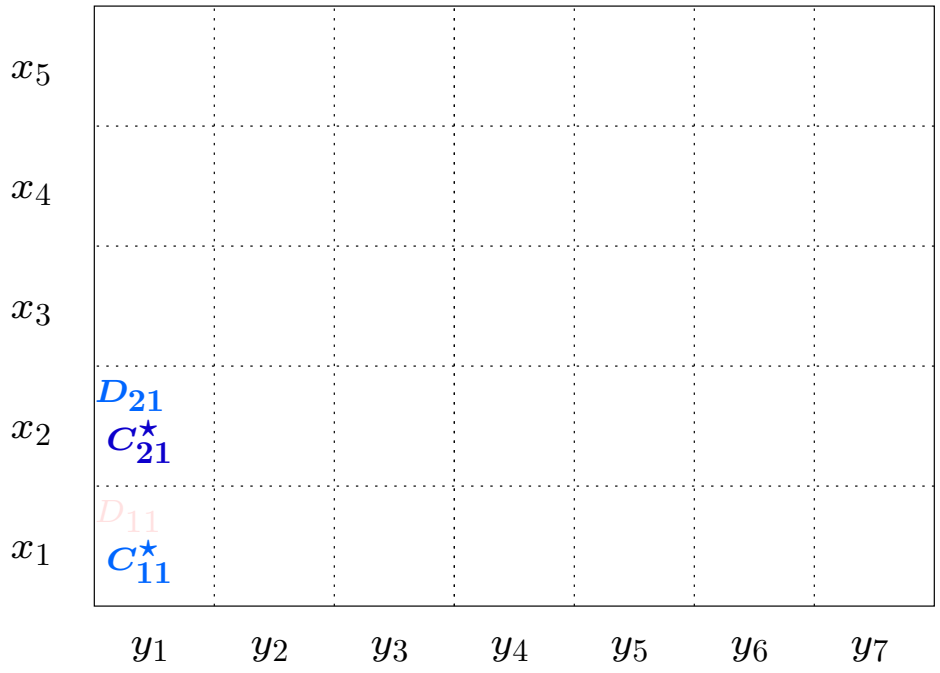
We first compute $C^{\star}_{1,1}$

Optimal Alignment



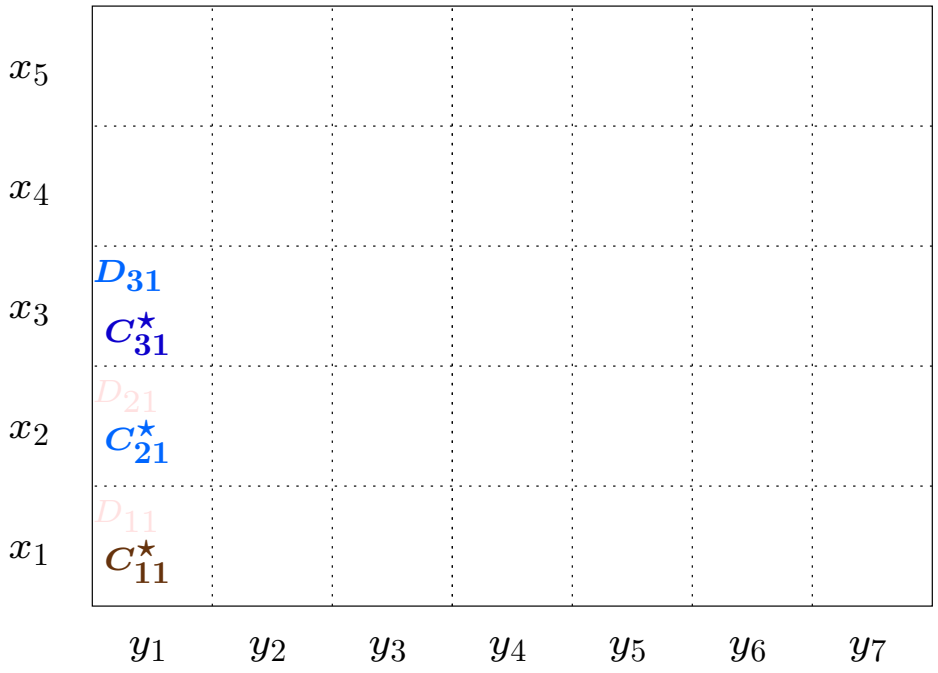
Easy, since $C^{\star}_{1,1} = D_{1,1}$

Optimal Alignment



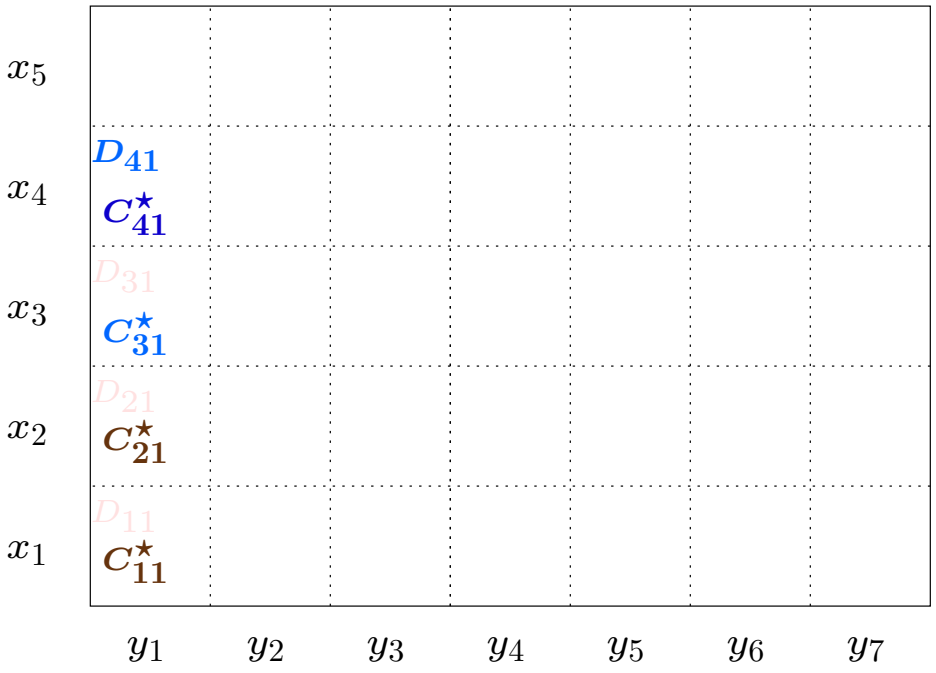
We now compute $C^\star_{2,1} = C^\star_{1,1} + D_{2,1}$

Optimal Alignment



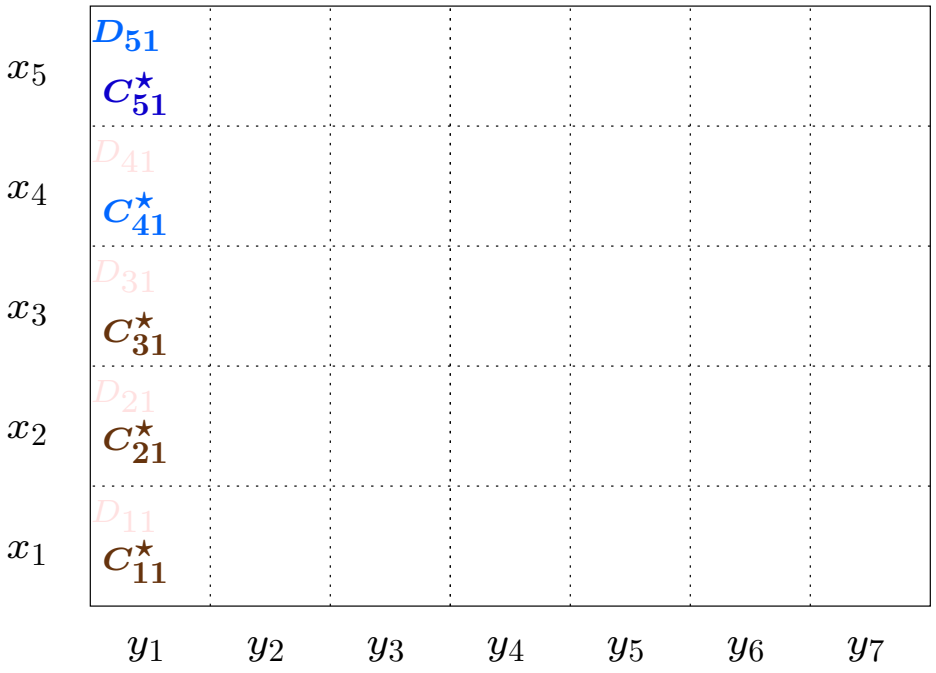
Same for $C^{\star}_{3,1}$...

Optimal Alignment



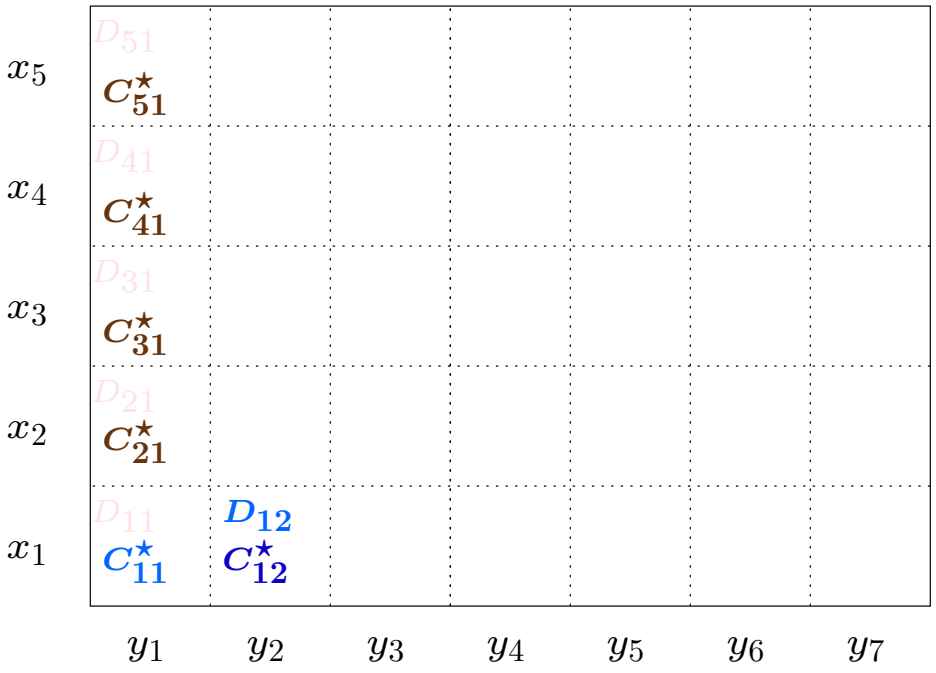
$$\ldots C^\star_{4,1} \cdots$$

Optimal Alignment



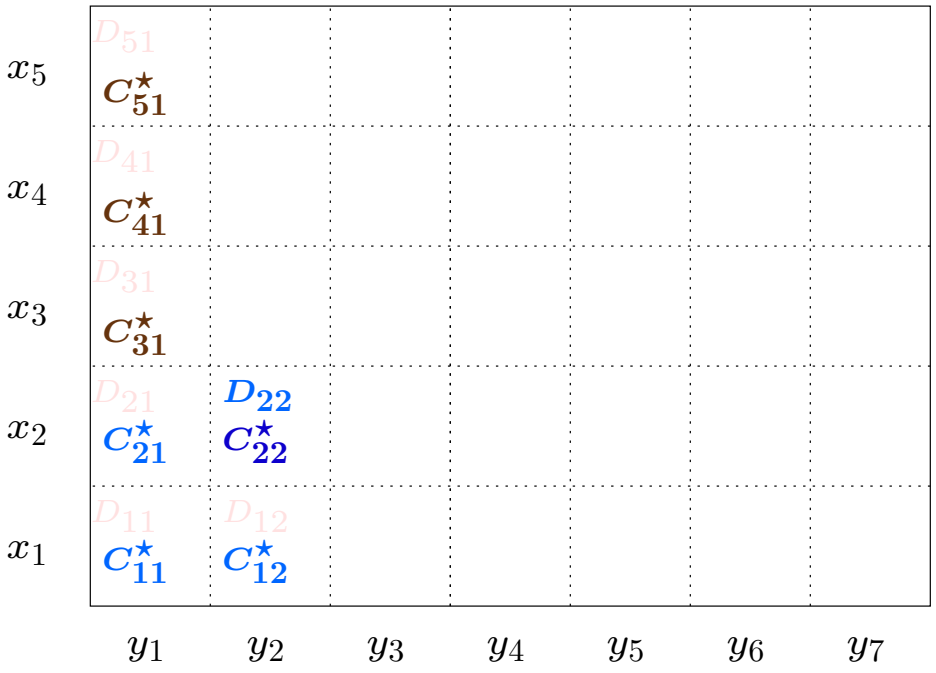
... and $C^{\star}_{5,1}$...

Optimal Alignment



$C^\star_{1,2}$ depends only on $C^\star_{1,1}$ and $D_{1,2}$

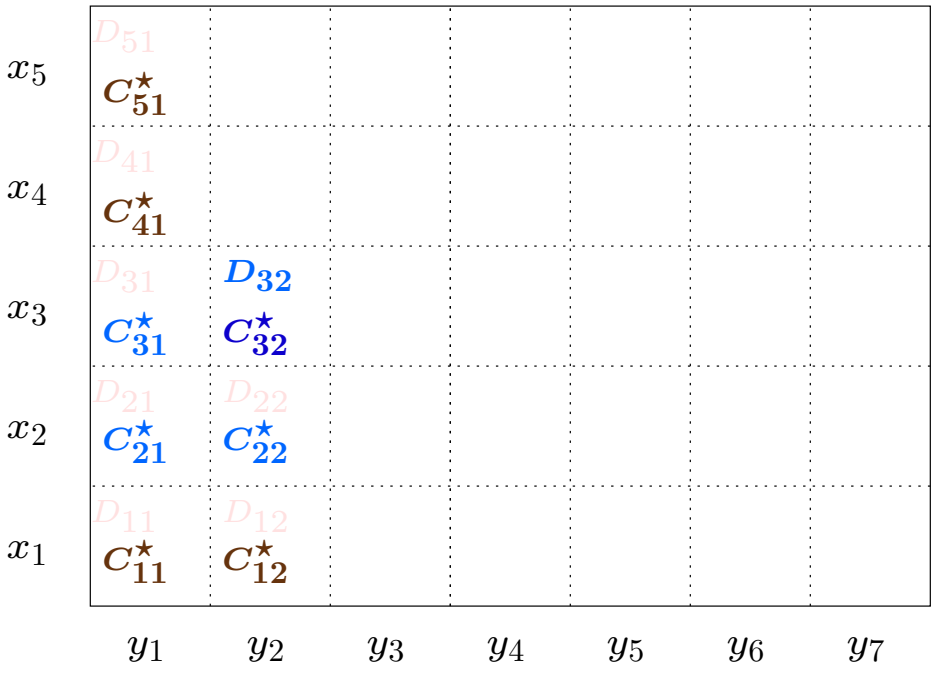
Optimal Alignment



We now apply Bellmans recurrence for the first time:

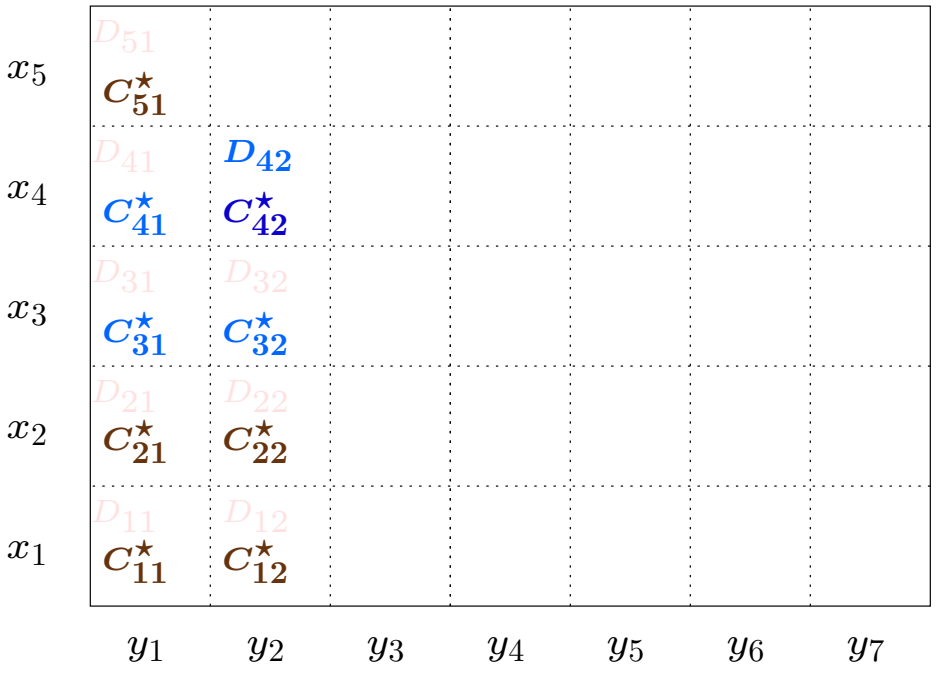
$$C^\star_{22} = \min(C^\star_{21}, C^\star_{11}, C^\star_{12}) + D_{22}$$

Optimal Alignment



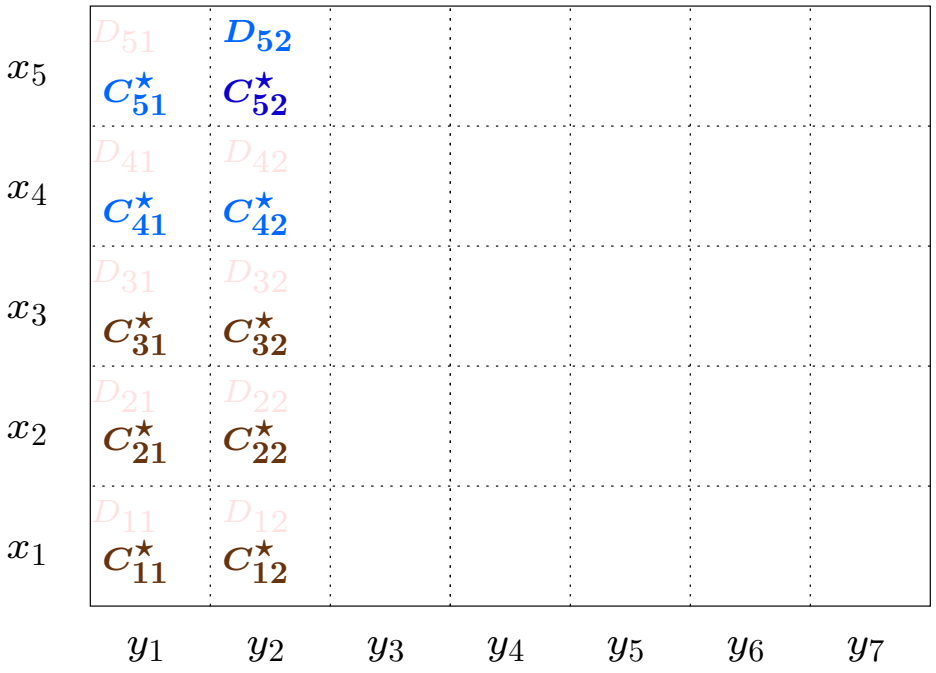
...

Optimal Alignment



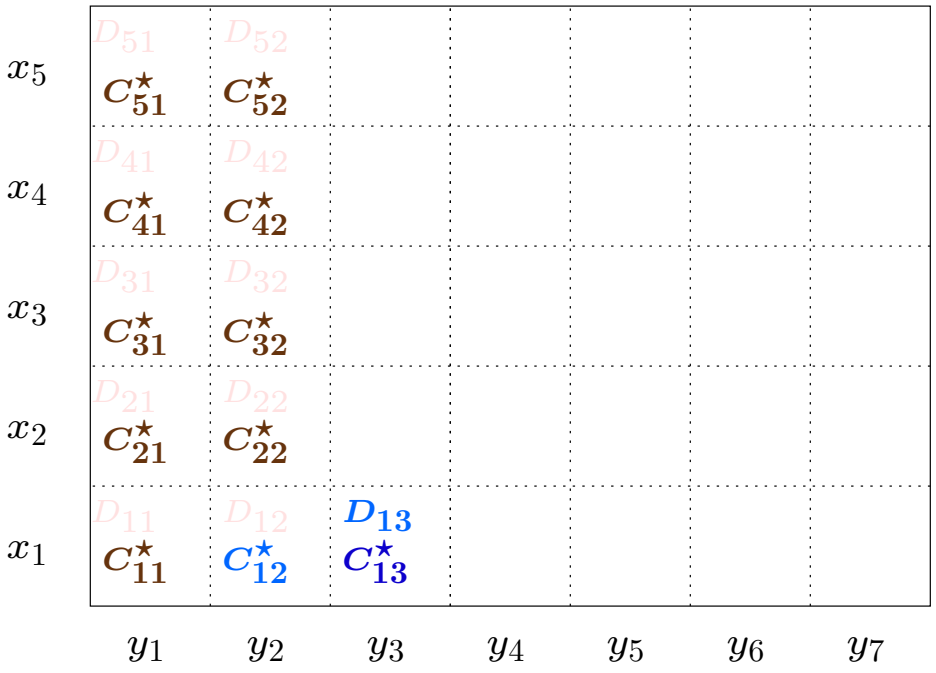
...

Optimal Alignment



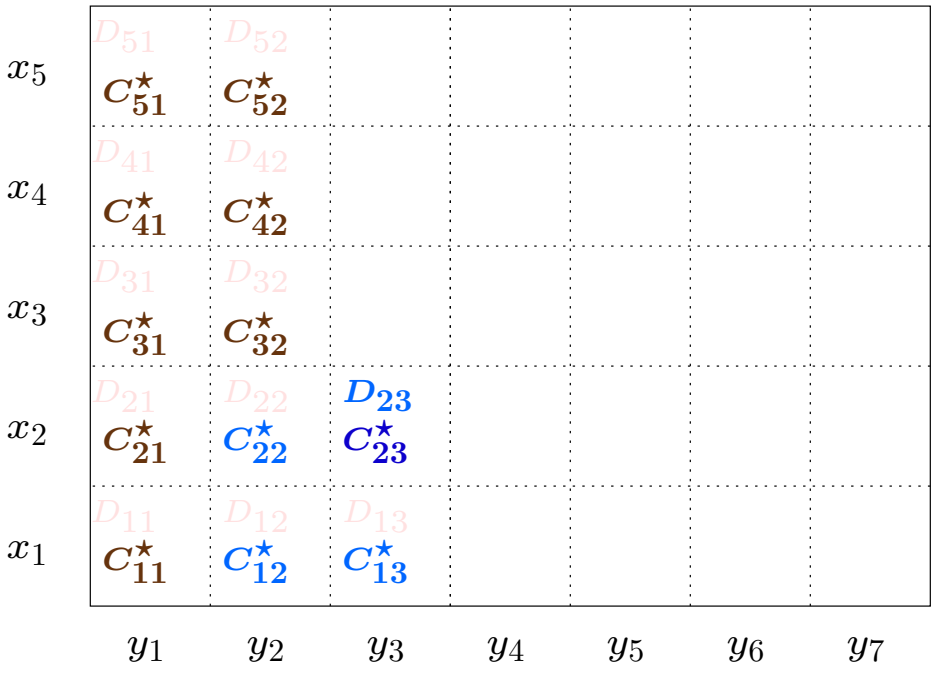
...

Optimal Alignment



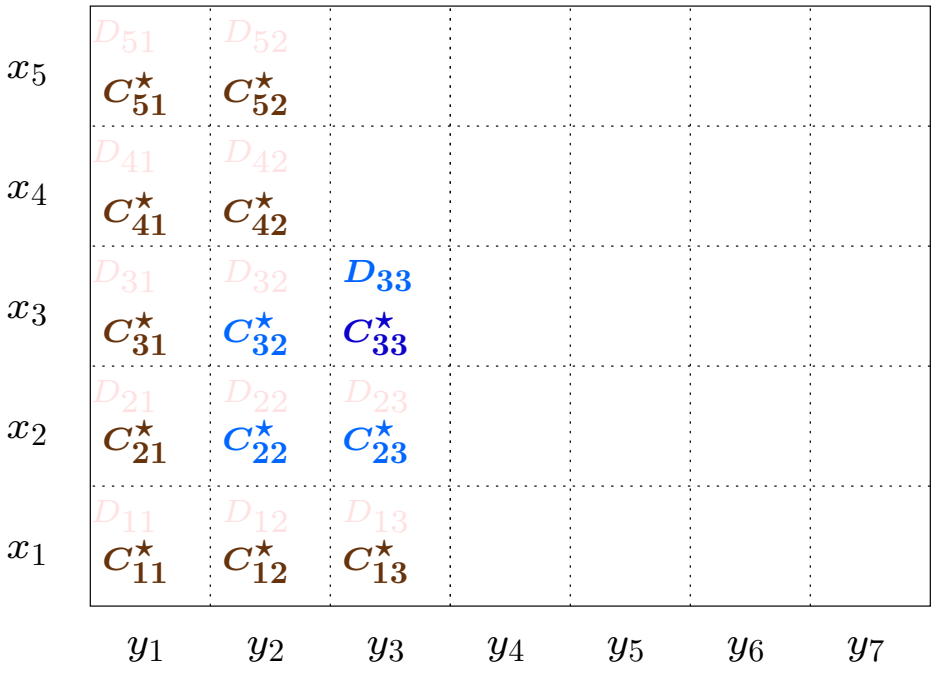
...

Optimal Alignment



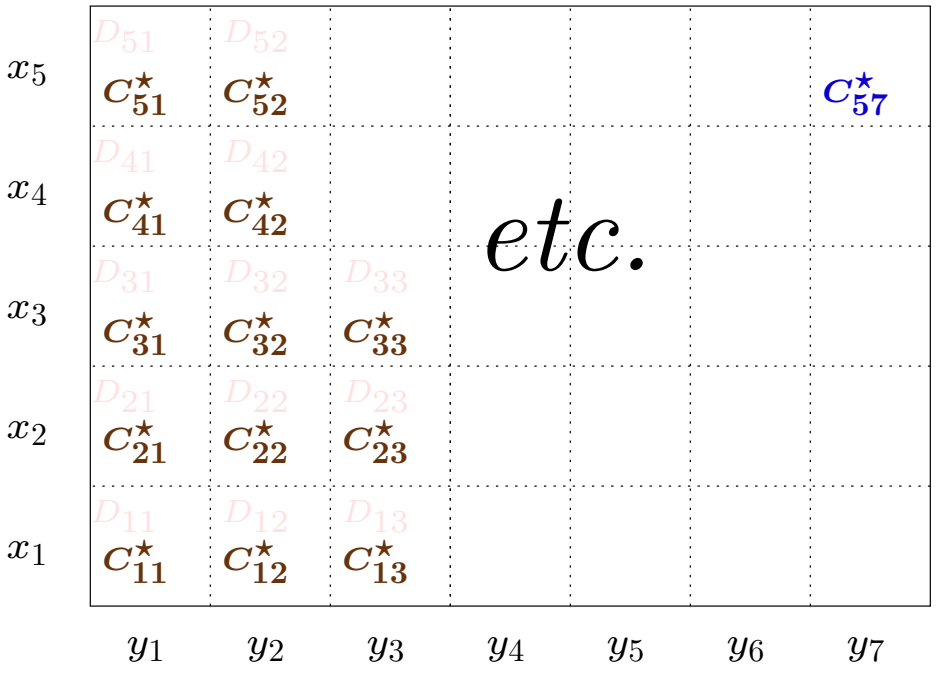
...

Optimal Alignment



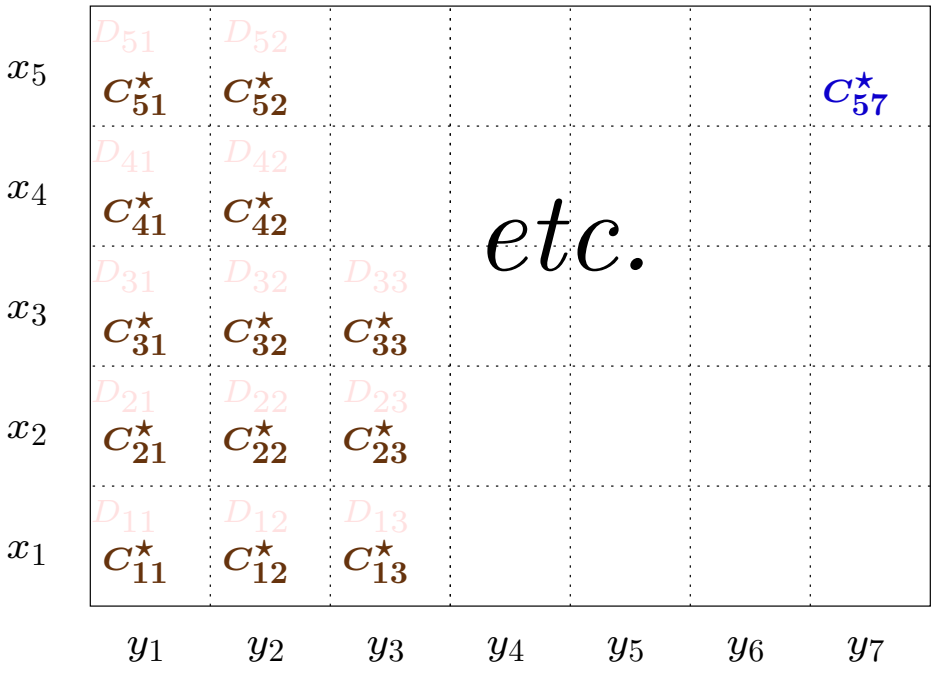
...

Optimal Alignment



until we recover the final value $C^\star_{57}$

Optimal Alignment



Complexity: nm **operations**

... substantial improvement over *Delannoy*$(n, m)\times$ cost per path ...

DTW distance

To recapitulate

- Sakoe & Chiba defined the distance

$$d_{\mathrm{DTW}}(\mathbf{x}, \mathbf{y}) = \min_{\pi \in \mathcal{A}(|\mathbf{x}|, |\mathbf{y}|)} \sum_{i=1} d\left(x_{\pi_1(i)}, y_{\pi_2(i)}\right),$$

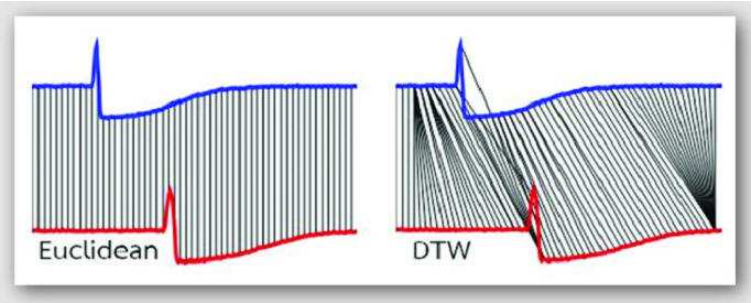
where $d(x, y)$ is usually $d(x, y) = \|x - y\|$.

- Can be computed in $O(nm)$ iterations.
- Can be proved to be a distance (triangular inequality, etc...)

What are the strengths & weaknesses of the DTW?

Strengths of the DTW distance

- Intuitive, works well for simple examples in practice.



- Can be easily generalized to time-series in **metric spaces** - just need $d(x_i, y_j)$
- Used extensively in **information retrieval / nearest neighbour** search:
  - Given **x**, scan in a large database and return its closest matches
  - Clever approaches to speed up these searches

Image taken from `http://www.eng.chula.ac.th/` (Chulalongkorn University)

Weaknesses of the DTW distance

- The **distance** DTW is **NOT** a **negative definite kernel**. The similarity

$$k_{\text{DTW}}(\mathbf{x}, \mathbf{y}) = e^{-d_{\text{DTW}(\mathbf{x},\mathbf{y})}},$$

is **NOT** positive-definite in general.

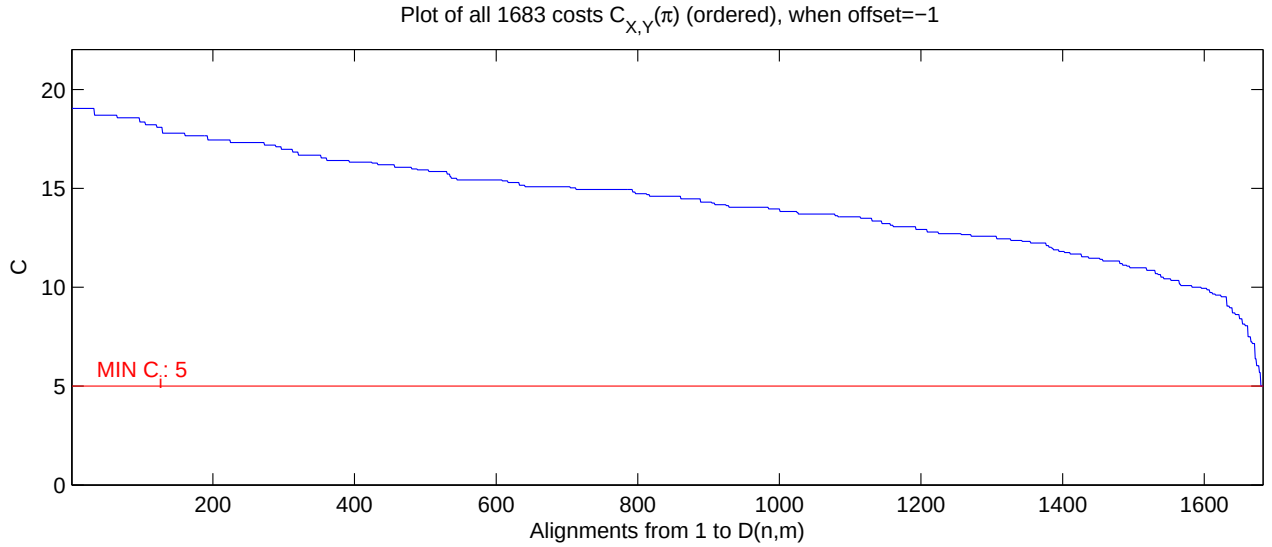
  - You can use it with a SVM... but you have to **tweak** it or be lucky

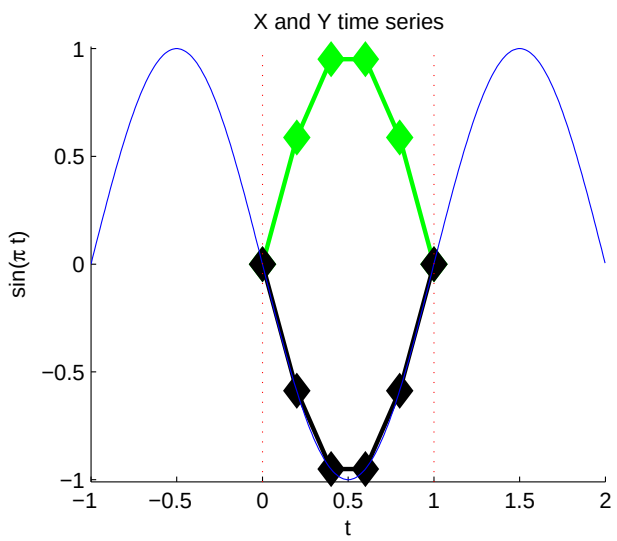
- More worryingly, DTW is a very arbitrary choice:

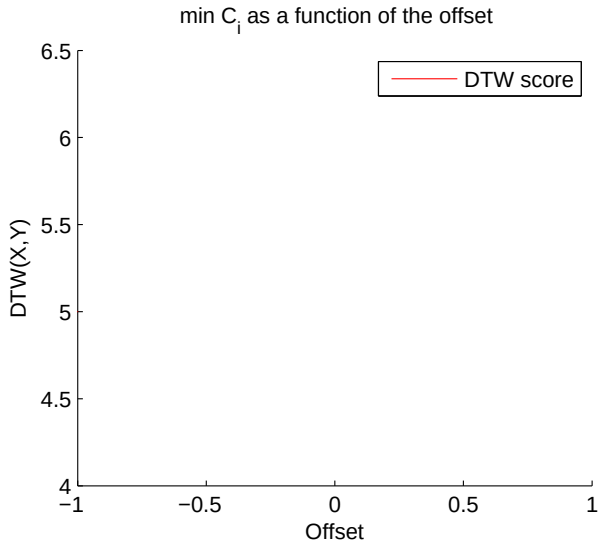
> Given **x** and **y**, DTW quantifies their similarity by looking at the set of all costs $\{C_{\mathbf{x},\mathbf{y}}(\pi), \pi \in \mathcal{A}(|\mathbf{x}|, |\mathbf{y}|)\}$ but only considers its **minimum**!.

  - This leads to **unexpected** and **counter-intuitive** behavior in some cases:

Weaknesses of the DTW distance







A different idea, more robust

- Rather than the **minimum**, consider the consider the **soft-minimum** of $C_{\mathbf{x},\mathbf{y}}$:

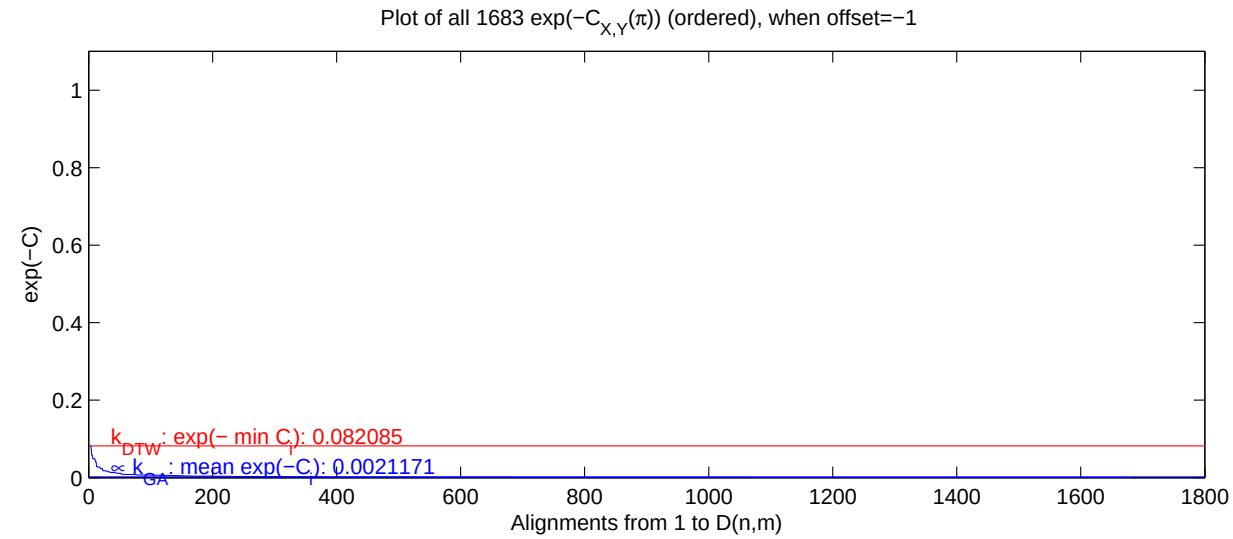
$$\text{soft-minimum}(C_{\mathbf{x},\mathbf{y}}) = -\log \sum_{\pi \in \mathcal{A}(|\mathbf{x}|,|\mathbf{y}|)} e^{-C_{\mathbf{x},\mathbf{y}}(\pi)}$$

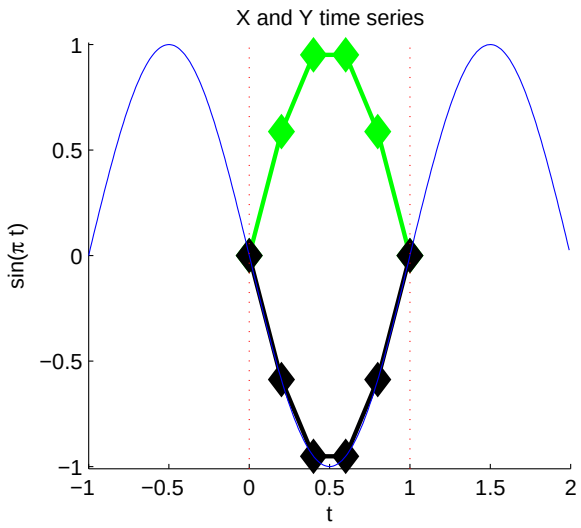
- Since we need a **similarity**, we consider $\exp(-\textbf{soft-minimum})$,

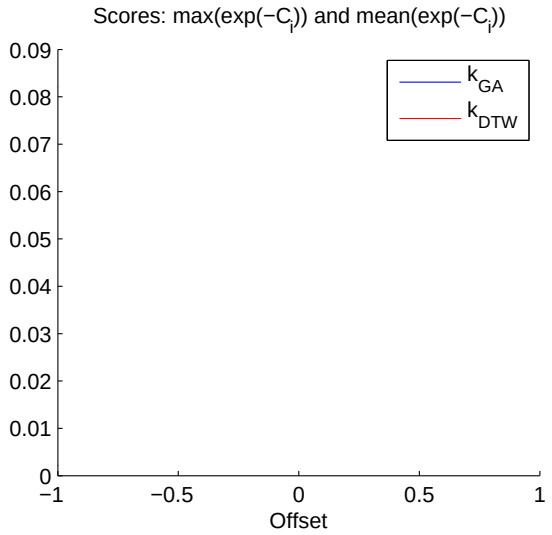
$$k_{\text{GA}} = \sum_{\pi \in \mathcal{A}(|\mathbf{x}|,|\mathbf{y}|)} e^{-C_{\mathbf{x},\mathbf{y}}(\pi)}$$

- **First proposed here!** J.P. Vert, H. Saigo & Prof. Akutsu in a 2004 paper
- Also considered on trees currently (joint work with K.Shin & T. Kuboyama)
- Let's compare $k_{\text{DTW}} = e^{-\text{DTW}}$ and k_{GA}

$e^{-\textbf{min}C(\pi)}$ **vs** $e^{-\textbf{soft-min}C(\pi)} = \sum e^{-C_i}$







Minimal-cost alignment vs. all alignments

- **Soft-minimum** is intuitively more appealing than minimum.

Yet, not enough... two important issues remain:

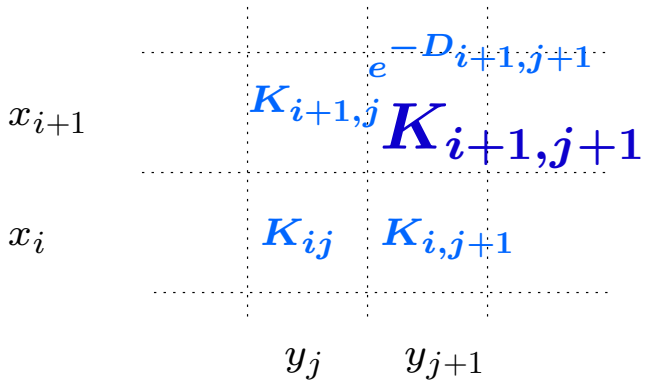
- Do we have to sum over **all** $A(|\mathbf{x}|, |\mathbf{y}|)$ **alignments** to compute k_{GA}?
- k_{DTW} is $\boldsymbol{NOT}$ positive definite, what about k_{GA}?

These two questions were answered in our ICASSP 2007 paper:

A kernel for Time-Series based on Global Alignments, M.C, J.-P. Vert, O. Birkenes, T. Matsui

All alignments: cheap to compute

- Do we have to sum over **all** $A(|\mathbf{x}|, |\mathbf{y}|)$ **alignments** to compute k_{GA}? **NO**
  - Computing k_{GA} has the **same complexity** than DTW: $\boldsymbol{O(nm)}$.
  - Change Bellman recursion $\boldsymbol{C^\star_{i+1,j+1}} = \min(\boldsymbol{C^\star_{i+1,j}}, \boldsymbol{C^\star_{ij}}, \boldsymbol{C^\star_{i,j+1}}) + \boldsymbol{D_{i+1,j+1}}$



$$\text{to } \boldsymbol{K_{i+1,j+1}} = (\boldsymbol{K_{i+1,j}} + \boldsymbol{K_{ij}} + \boldsymbol{K_{i,j+1}})\, e^{-\boldsymbol{D_{i+1,j+1}}}$$

  - Recover kernel value as $k_{\text{GA}(\mathbf{x},\mathbf{y})} = \boldsymbol{K_{|\mathbf{x}|,|\mathbf{y}|}}$.
  - Similar to the work of Vert-Saigo-Akutsu.

All alignments: Positive Definite

- k_{DTW} is $\boldsymbol{NOT}$ positive definite, what about k_{GA}? **YES, BUT...**
  - k_{GA} **is positive definite** if the function $f(x,y) \stackrel{\text{def}}{=} e^{-d(x,y)}$ is such that

$$\frac{f}{1+f}$$

is a positive definite kernel.

  - Simple trick to define functions f: take a p.d. kernel $\kappa < 1$, define

$$f \stackrel{\text{def}}{=} \frac{\kappa}{1-\kappa}.$$

  - in such a case,

$$\frac{f}{1+f} = \frac{\frac{\kappa}{1-\kappa}}{1+\frac{\kappa}{1-\kappa}} = \kappa$$

which is positive definite.

  - Very different proof, quite involved... please check the paper.

Still... a few challenges

The global alignment kernel k_{GA} is **not without problems**

- k_{GA} can be diagonally dominant: $k_{\mathrm{GA}}(\mathbf{x}, \mathbf{x}) \gg 1$ but $k_{\mathrm{GA}}(\mathbf{x}, \mathbf{y}) \approx 0$.
- the condition $f/(1+f)$ is positive definite is not well-understood.
- the quadratic $O(nm)$ complexity is still too high for large-scale applications.

In more recent work I look at these 3 different problems.

Cuturi, *Fast Global Alignment Kernels* (ICML 2011)

1. Diagonal Dominance

Problem: sometimes $k_{\mathsf{GA}}(\mathbf{x},\mathbf{x}) \gg 1$ but $k_{\mathsf{GA}}(\mathbf{x},\mathbf{y}) \approx 0$.

- **Solution**: use a **negative definite** distance d ($\Leftrightarrow$ **infinitely divisible** kernel κ),

$$i.e. \text{ such that } \kappa(x,y) \stackrel{\text{def}}{=} e^{-\lambda d(x,y)} \text{ is positive definite } \forall \lambda > 0$$

- When d is scaled by $\boldsymbol{\lambda \to \infty}$,

$$k_{\mathsf{GA}}(\mathbf{x},\mathbf{y}) = \sum_{\pi \in \mathcal{A}(|\mathbf{x}|,|\mathbf{y}|)} e^{-\lambda C_{\mathbf{x},\mathbf{y}}(\pi)} = 1_{\{\mathbf{x}=\mathbf{y}\}} \operatorname{card} \mathcal{A}(|\mathbf{x}|,|\mathbf{x}|)) = \mathbf{1_{\{x=y\}}}\text{Delannoy}(|\mathbf{x}|)$$

yet, when $\boldsymbol{\lambda = 0}$,

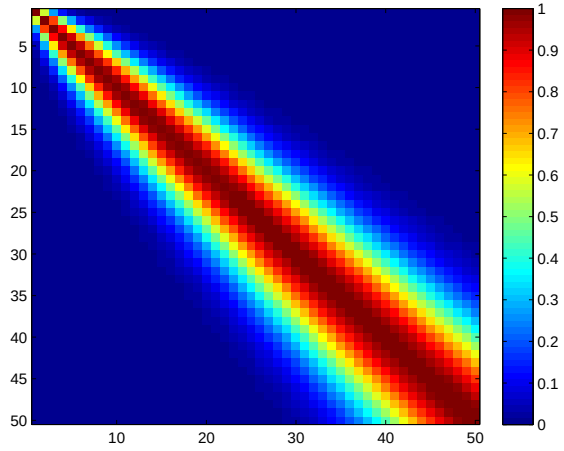
$$k_{\mathsf{GA}}(\mathbf{x},\mathbf{y}) = \sum_{\pi \in \mathcal{A}(|\mathbf{x}|,|\mathbf{y}|)} e^{-0} = \operatorname{card} \mathcal{A}(|\mathbf{x}|,|\mathbf{y}|)) = \text{Delannoy}(|\mathbf{x}|,|\mathbf{y}|)$$

- Given a database $\mathbf{x}_1, \cdots, \mathbf{x}_N$, the Gram matrix varies between
  - $\boldsymbol{\lambda = 0}$: the matrix $[\text{Delannoy}(|\mathbf{x}_i|,|\mathbf{x}_j|)]$
  - $\boldsymbol{\lambda \to \infty}$: the Diagonal matrix $\mathbf{diag}\,(\text{Delannoy}(|\mathbf{x}_i|))$.

1. Diagonal Dominance

- if $|\mathbf{x}_i| = |\mathbf{x}_j|$, we can tune λ to solve diagonal dominance.
- if $\mathbf{x}_i \neq \mathbf{x}_j$,
  - Can prove a bound on the spectrum of the Delannoy $D(n, m)$ matrix,

    Lemma 1. $$\sum_{i,j=1, i\neq j}^{n} D(i,j) > \left(1 - \frac{n}{9n-1}\right) \sum_{i=1}^{n} D_i.$$
  - $k_{\mathsf{GA}}(\mathbf{x}, \mathbf{y})$ with $\lambda = 0$ is significantly different from 0 if $\frac{1}{2} < \frac{|\mathbf{x}|}{|\mathbf{y}|} < 2$.



Conclusion: using a **scaled n.d. distance** λd,
diagonal dominance can be avoided when lengths are **not too different**.

2. New results: Geometric Divisibility

Definition 2 (Geometric Divisibility). *Let* $\boldsymbol{f}$ *be a nonnegative valued function on* $\mathcal{X} \times \mathcal{X}$. f *is said to be geometrically divisible (g.d.) if* $\boldsymbol{f/(1+f)}$ *is* ***positive definite***.

Remark 1. *If* f *is g.d. and* $\boldsymbol{\kappa} \stackrel{\text{def}}{=} \boldsymbol{f/(1+f)}$ *then* $\boldsymbol{f} = \sum_{i=1}^{\infty} \boldsymbol{\kappa}^i$ *is necessarily p.d.*

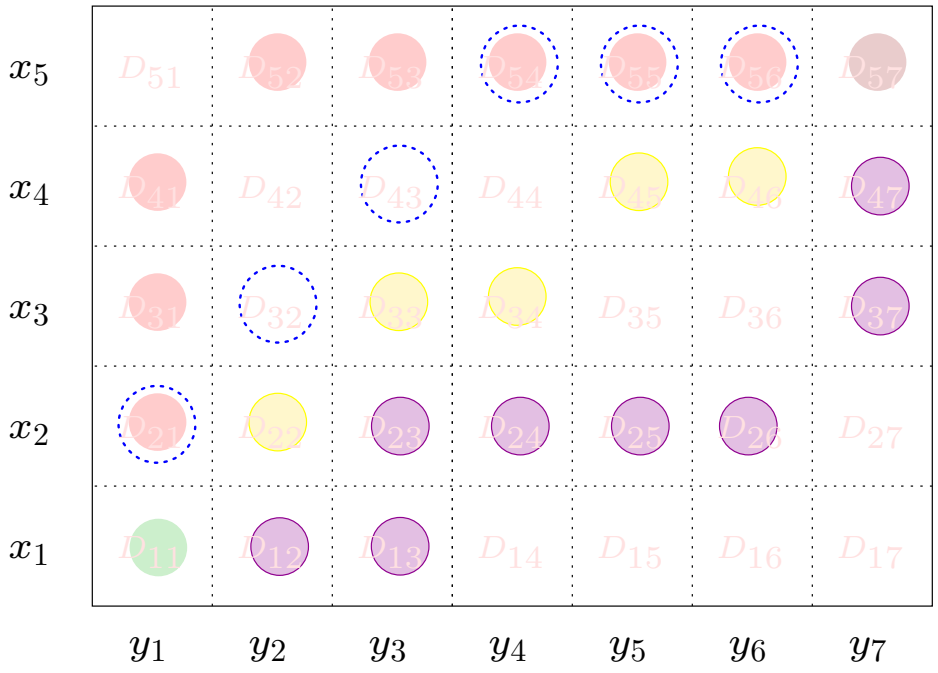
Lemma 2. *The Gaussian kernel* κ_σ *is* ***not*** *geometrically divisible.*

Lemma 3. *For an* ***infinitely divisible*** *kernel* $\boldsymbol{\kappa}$ *such that* $0 < \boldsymbol{\kappa} < 1$, $\boldsymbol{\kappa/(1-\kappa)}$ *is both geometrically divisible and* ***infinitely divisible***.

Motivated by these results, I propose to use the following distance in k_{GA},

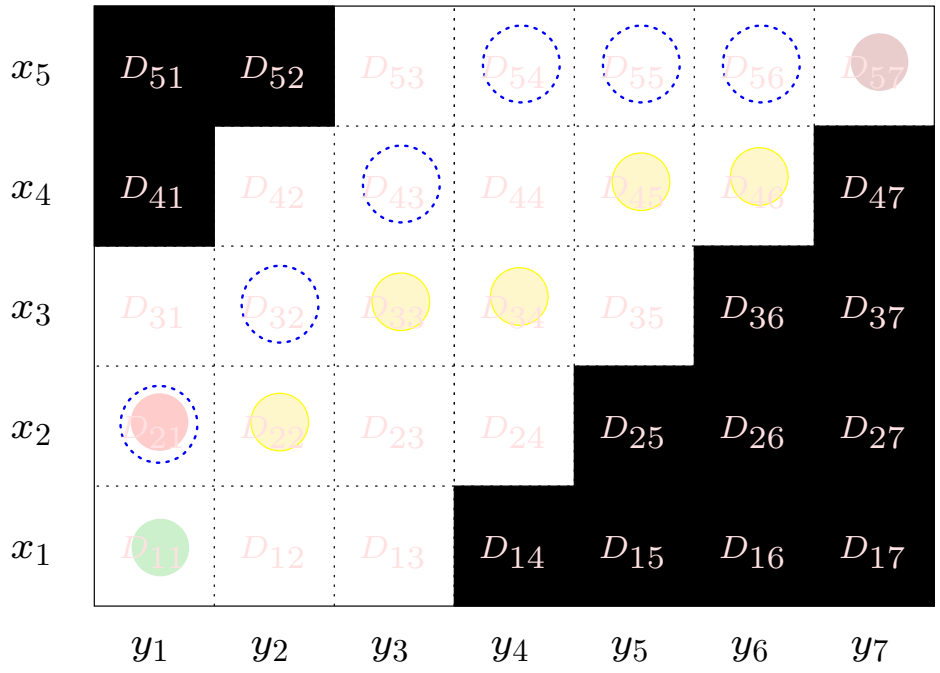
$$d(x,y) \stackrel{\text{def}}{=} \frac{1}{2\sigma^2}\|x-y\|^2 + \log\left(2 - e^{-\frac{\|x-y\|^2}{2\sigma^2}}\right).$$

3. Speeding up k_{GA}: old ideas from DTW



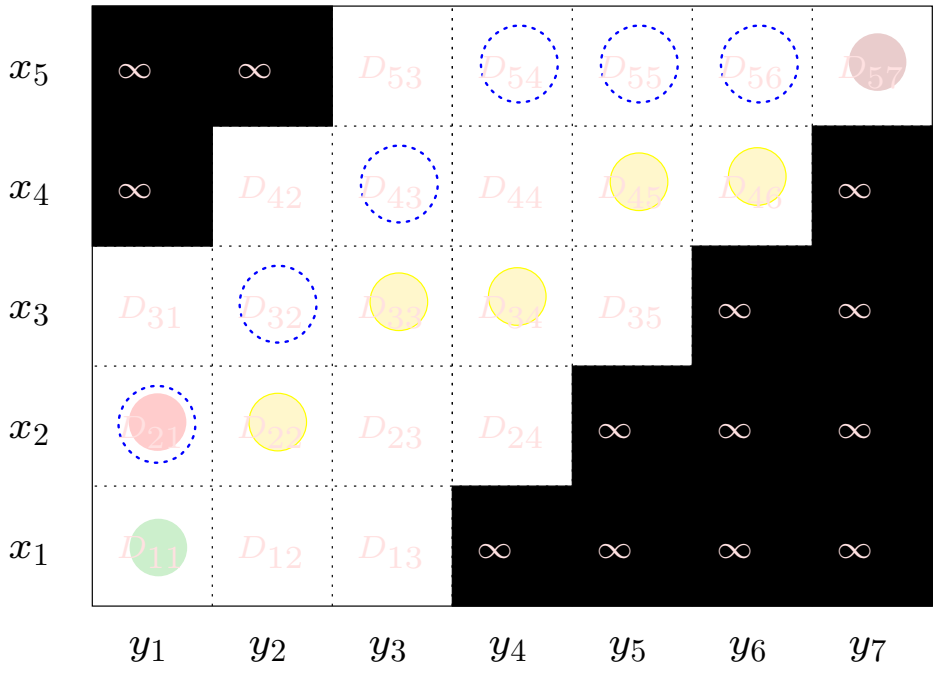
Itakura (75) and Sakoe-Chiba (78)
propose to **speed up** the DTW computation by **ignoring** zones in the grid.

3. Speeding up k_{GA}: old ideas from DTW



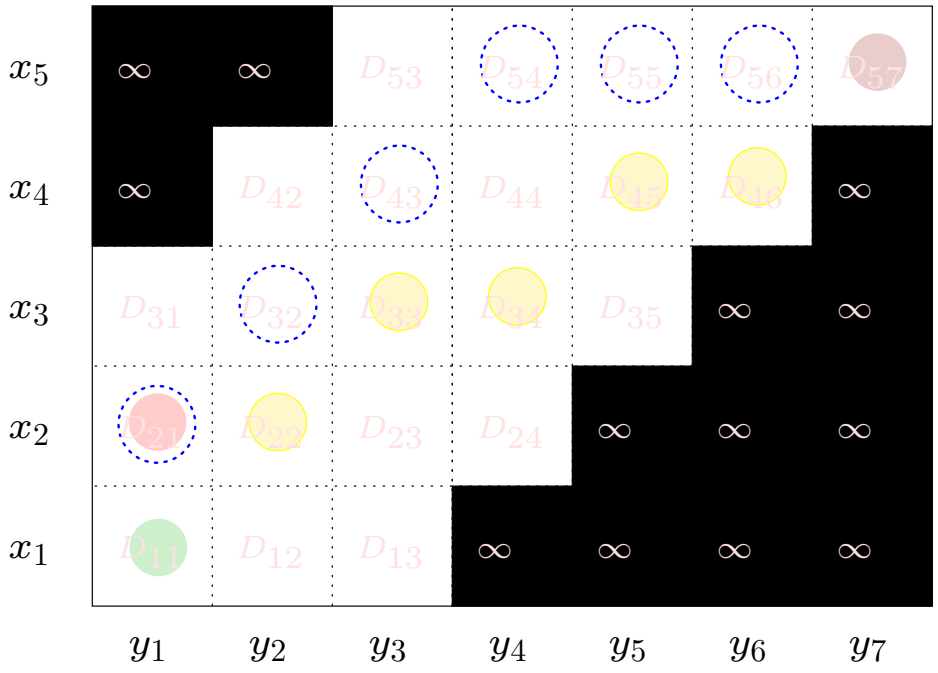
Decide a-priori that some paths are unlikely to be of interest.

3. Speeding up k_{GA}: old ideas from DTW



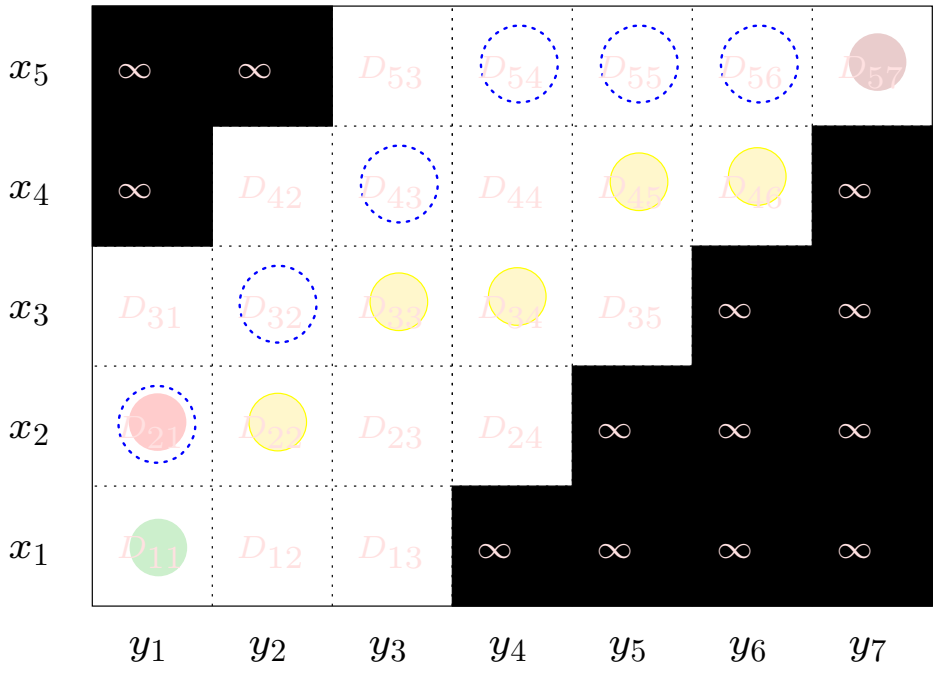
Easily done by setting distance $D_{ij} = \infty$ when $|i - j| > T$.

3. Speeding up k_{GA}: old ideas from DTW



Speed up: from $O(nm)$ to $O(T \min(n, m))$.

3. Speeding up k_{GA}: old ideas from DTW



Yet, this can be **suboptimal**! Not guaranteed to find best path!

3. Speeding up $k_{\mathbf{GA}}$

- In kernel methods, such weighting schemes need to preserve **positive definiteness**.
- Consider p.d. kernels $\omega(i, j)$ that only depend on $|i - j|$,

$$\omega(i, j) = \psi(|i - j|),$$

  where ψ is a real-valued function on $\mathbb{N}$.
- Such kernels on integers are also known as **Toeplitz kernels**.

  Definition 3. *A Toeplitz kernel ω is compactly supported of order $T \in \mathbb{N}$ if for $q \geq T, \psi(q) = 0$ and $\psi(T - 1) \neq 0$.*

3. Speeding up $k_{\mathbf{GA}}$

- Using such a kernel within GA kernels has obvious advantages

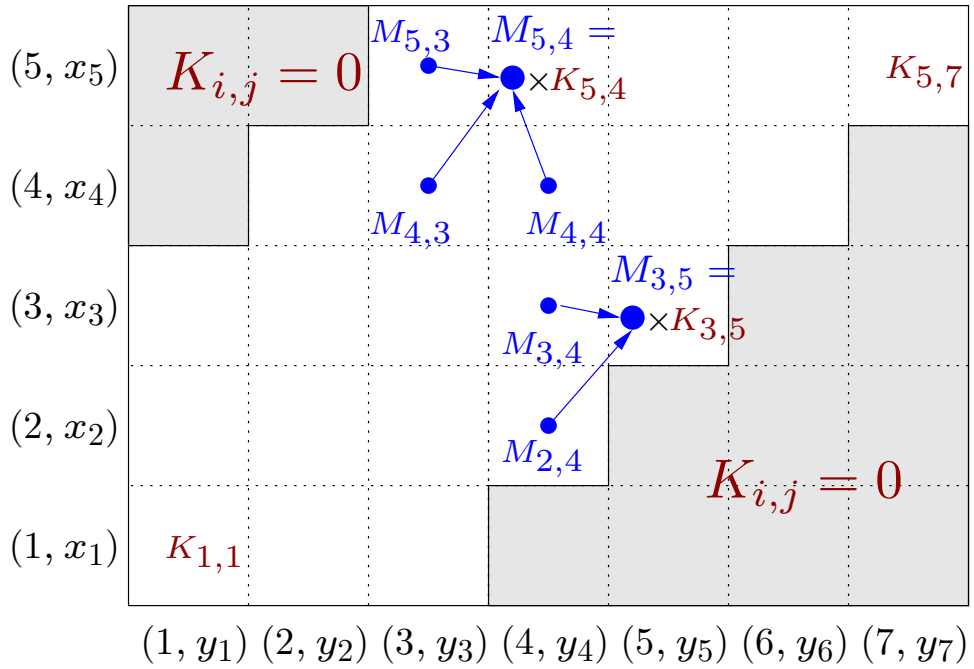
  Theorem 2. *Let κ be a kernel on $\mathcal{X} \times \mathcal{X}$ and ω a compactly supported Toeplitz kernel of order T. Then using $\frac{\omega\kappa}{1-\omega\kappa}$ as a local kernel, $k_{GA}(\mathbf{x}, \mathbf{y})$ can be computed with $O(T \min(n, m))$ operations. Furthermore, $k_{GA}(\mathbf{x}, \mathbf{y})$ is null when $|n - m| > T$.*

- Example: Triangular Kernel

$$\omega(i, j) = \left(1 - \frac{|i - j|}{T}\right)_+ .$$

3. Speeding up k_{GA}

Using a triangular kernel ω and a kernel κ, k_{GA} is **also sped up to** $O(T \min(n, m))$...



Here $K_{i,j}$ stands for $\left(\frac{\cdot}{1-\cdot}\right)(\omega \otimes \kappa((i, x_i), (j, y_j)))$

Experimental Results: Classifying Time Series

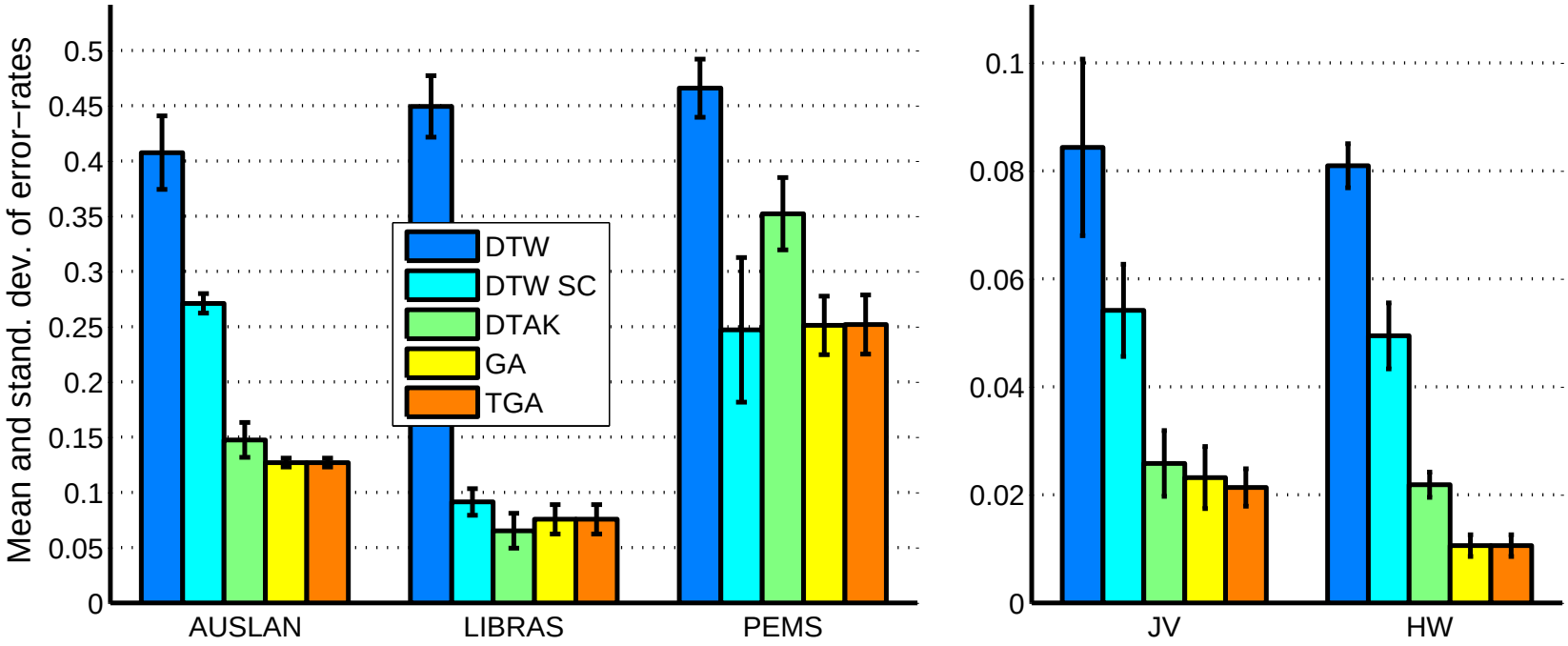
Benchmark Datasets (UCI repository) + **PEMS database** which we assembled

Database	d	n, $\mathrm{med}(n)$	**classes**	**# points**
Japanese Vowels	12	7-29, 15	9	640
Libras	2	45	15	945
Handwritten Characters	3	60-182, 122	20	2858
AUSLAN	22	45-136, 55	95	2465
PEMS	963	144	7	440

We consider the DTW kernel k_{DTW} and a few more...

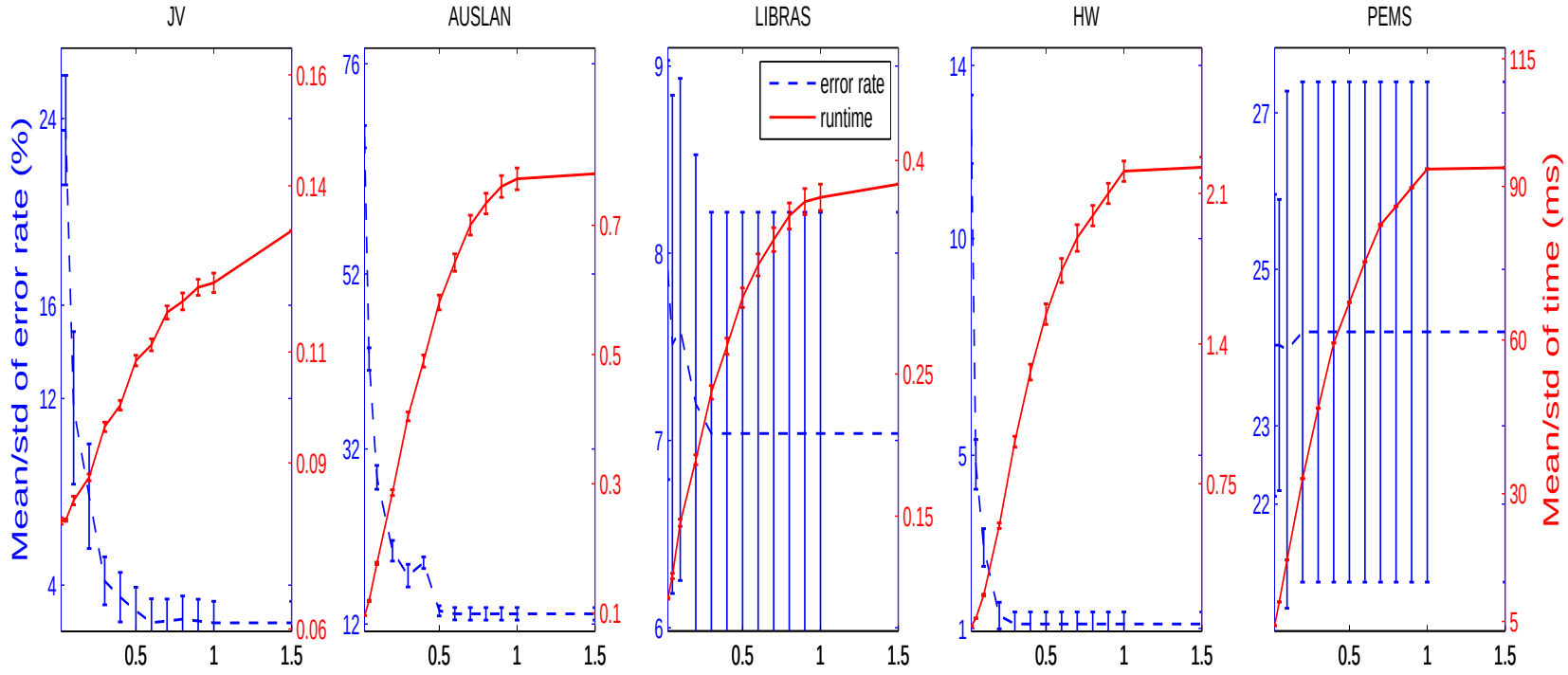
Kernel	Parameters	Parameter Values
k_{DTW}	t	$t \in \{0.2, 0.5, 1, 2, 5\} \cdot \mathrm{med}(\mathsf{DTW}(\mathbf{x}, \mathbf{x}))$
k_{SC}	t, T	$t \in \{0.2, 0.5, 1, 2, 5\} \cdot \mathrm{med}(\mathsf{DTW}_{\mathsf{SC}}(\mathbf{x}, \mathbf{y}))$, $T \in \{0.25, 0.5\} \cdot \mathrm{med}(\lvert\mathbf{x}\rvert)$
k_{DTAK}	t, σ	$t \in \{0.2, 0.5, 1, 2, 5\} \cdot \mathrm{med}(-\log k_{\mathsf{DTAK}}(\mathbf{x}, \mathbf{y}))$, $\sigma \in \{0.2, 0.5, 1, 2\} \cdot \mathrm{med}(\lVert x - y\rVert)$
k_{GA}	σ	$\sigma \in \{0.2, 0.5, 1, 2, 5\} \cdot \mathrm{med}(\lVert x - y\rVert) \cdot \sqrt{\mathrm{med}(\lvert\mathbf{x}\rvert)}$
k_{TGA}	σ, T	$\sigma \in \{0.2, 0.5, 1, 2, 5\} \cdot \mathrm{med}(\lVert x - y\rVert) \cdot \sqrt{\mathrm{med}(\lvert\mathbf{x}\rvert)}$, $T \in \{0.25, 0.5\} \cdot \mathrm{med}(\lvert\mathbf{x}\rvert)$

Experimental Results



Results averaged on 3-fold 3-repeats cross validations.
Parameters selected within training folds using 3-fold 2-repeats.

Experimental Results



Comparing the effect of T (as fraction of median length) on speed and classification performance.

Conclusion

Better not use DTW with a kernel machine ($e.g.$ SVM's), try $\boldsymbol{k}_{\mathsf{GA}}$ instead