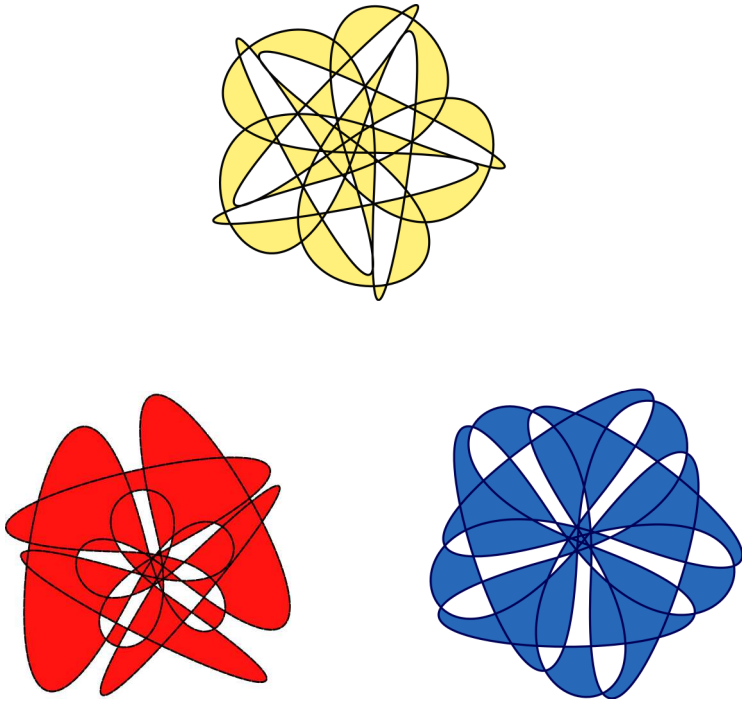
Introduction to Information Sciences

Machine Learning
Metrics and Kernels

Marco Cuturi - Kyoto University

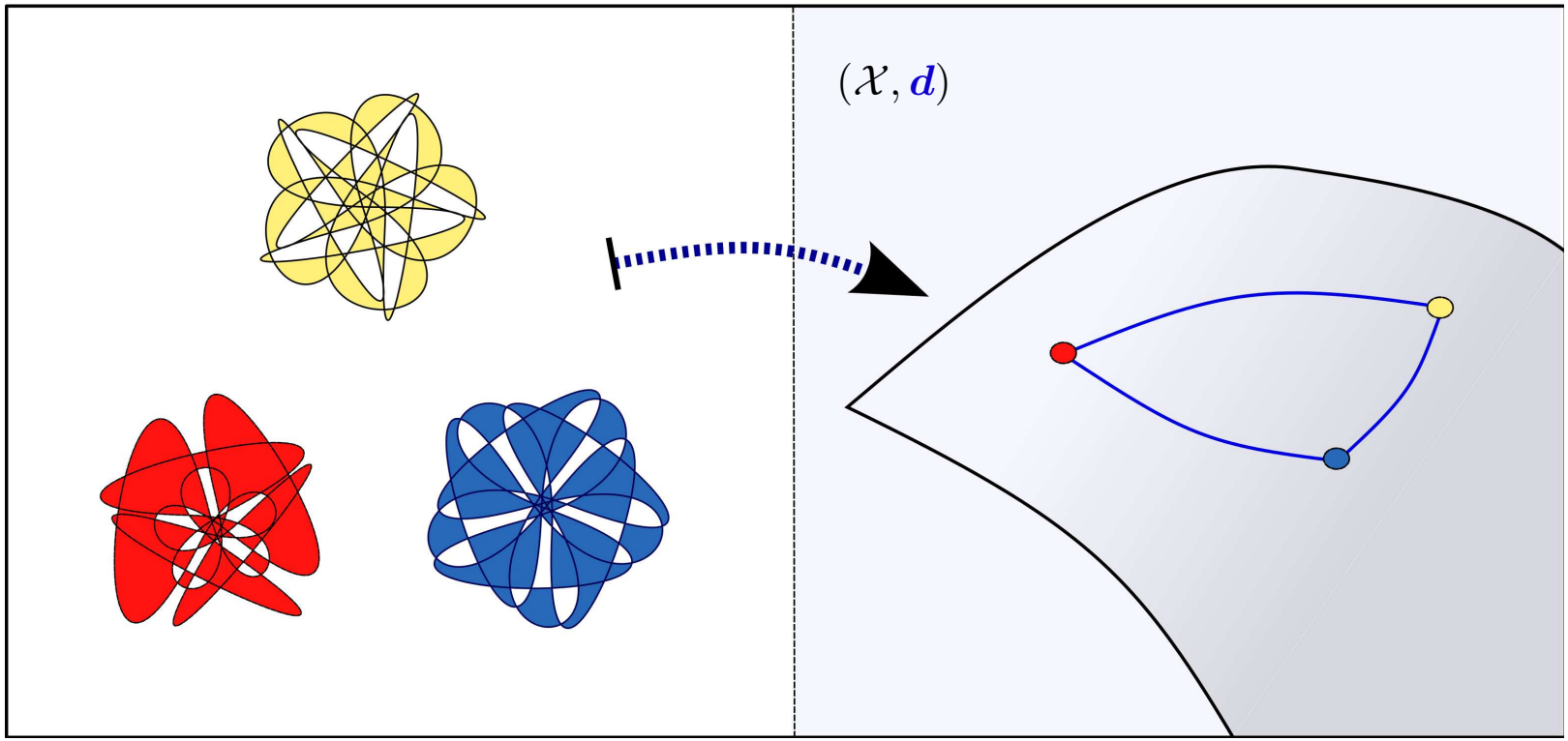
Address Complexity in Machine Learning



- Embed structures in metric spaces
- Embed structures in Hilbert spaces
- Create feature vectors: embed in $\mathbb{R}^d$.

$\mapsto$ metric Space

$$\boldsymbol{d}(x, y)$$

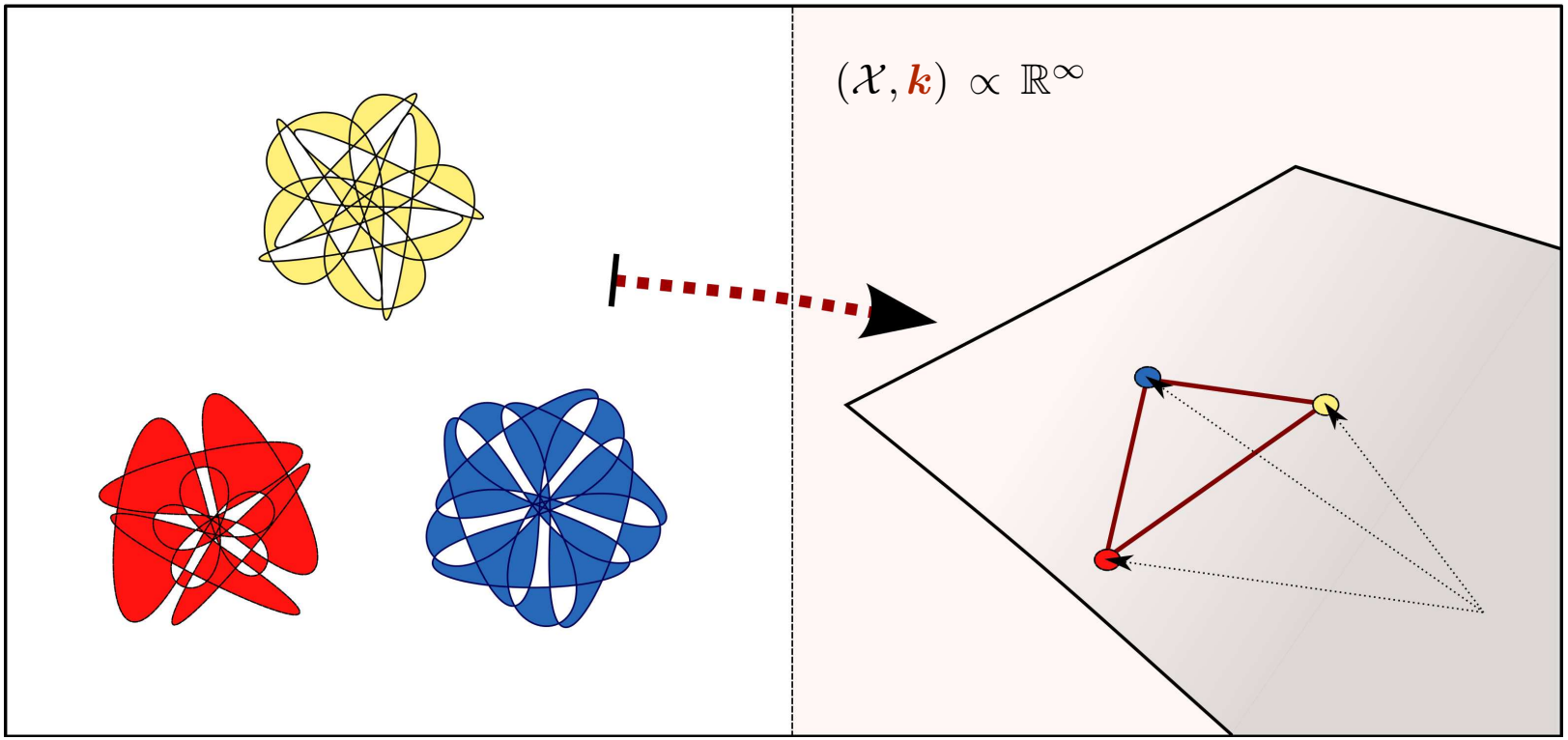


Retrieval & Clustering - Nearest Neighbor Methods

+ implicit map, flexibility / −− limited operations

$\mapsto$ Hilbert Space

$$\boldsymbol{k}(x,y) = \langle \Phi(x), \Phi(y) \rangle_{\mathcal{H}}, \quad \mathbf{d}(x,y) = \|\Phi(x) - \Phi(y)\|_{\mathcal{H}}$$

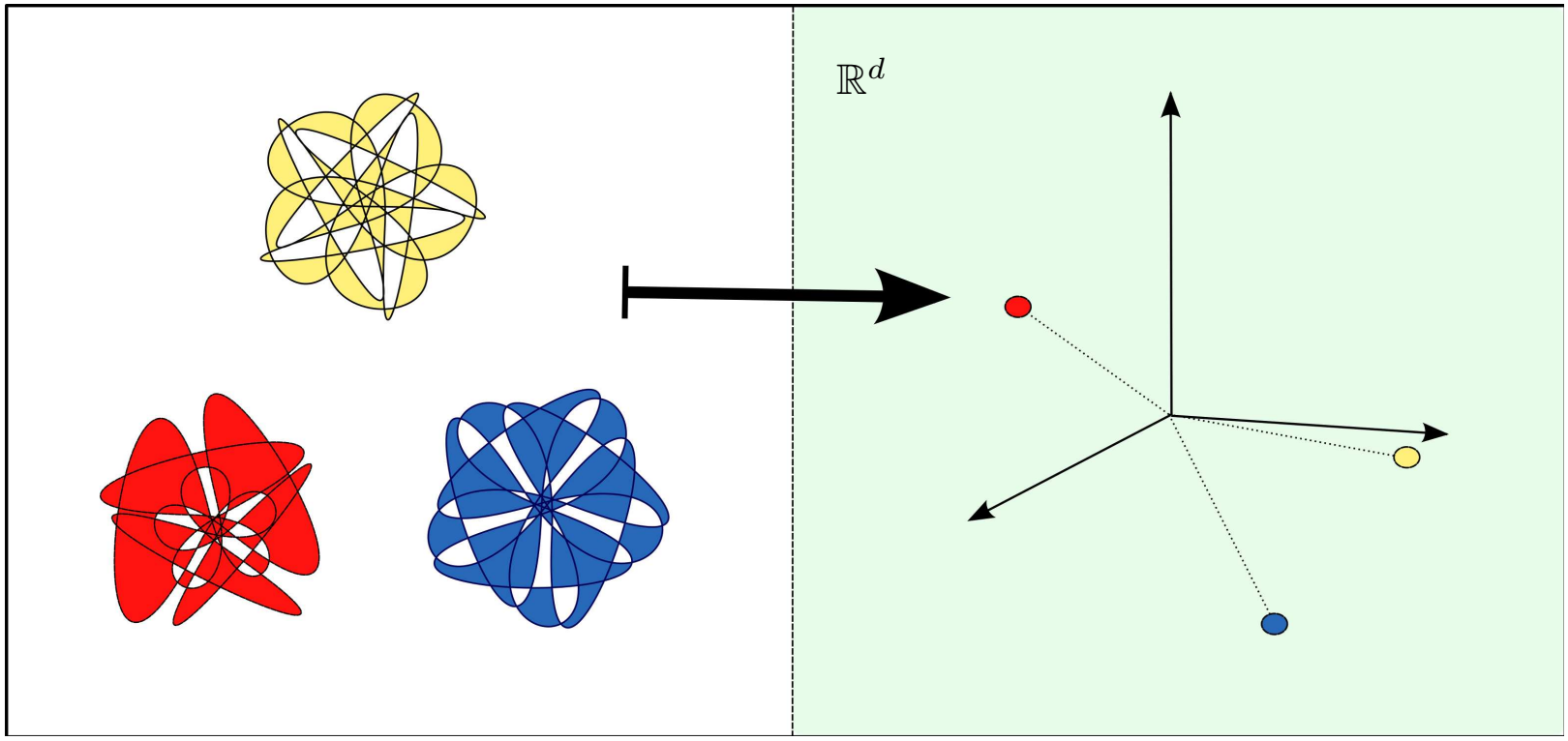


Kernel Methods, Support Vector Machines, Gaussian Processes ...

++ **kernel trick, convexity**/ – **limited operations**

$\mapsto$ Feature Vectors

$$\mathbf{x} = [x_1\ x_2\ x_3\ \cdots\ x_d]^T$$



Linear Models - High-dimensional statistics - Deep Learning ...

++ **fast, feature selection**/ − **explicit map**

Outline of the Next 2 Lectures

Talk about **distance** and **kernel** based methods

- Start with **distances**;
  - **why distances** in machine learning? example of k-nearest neighbors;
  - On **learning** a **good** distance for vectors with **Mahalanobis distances**
- Continue with **kernels**
  - define **positive definite kernels**
  - introduce **support vector machines** to illustrate

Distances

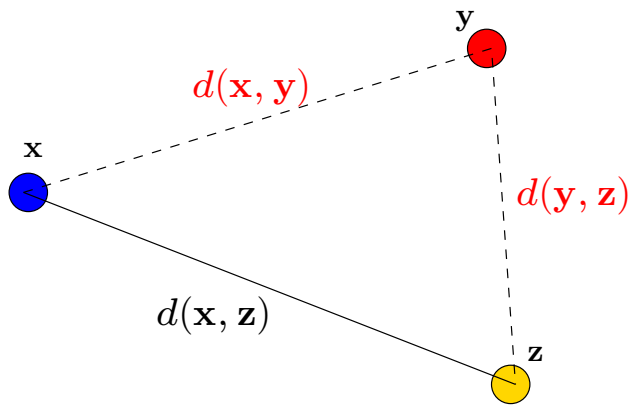
Distances

Definition: A **distance** defined on a set $\mathcal{X}$ is a function

$$\begin{array}{rccc} \boldsymbol{d}: & \mathcal{X} \times \mathcal{X} & \rightarrow & \mathbb{R}_+ \\ & (\mathbf{x}, \mathbf{y}) & \mapsto & d(\mathbf{x}, \mathbf{y}) \end{array}$$

such that $\forall\, \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{X}$,

- $\boldsymbol{d}(\mathbf{x}, \mathbf{y}) = \boldsymbol{d}(\mathbf{y}, \mathbf{x})$, **symmetry**
- $\boldsymbol{d}(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}$, **definiteness**
- $\boldsymbol{d}(\mathbf{x}, \mathbf{z}) \leq \boldsymbol{d}(\mathbf{x}, \mathbf{y}) + \boldsymbol{d}(\mathbf{y}, \mathbf{z})$, **triangle inequality**



Why Distances in Machine Learning?

- In machine learning problems, we consider **training sets** of points

$$T = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$$

and use these points to "learn" a way to handle **new** observations **x**.

For distance and kernel based algorithms,
learning from a training set T
actually means **use** T to check how similar **x** is
to the instances $\mathbf{x}_i$ in T
and make a decision based on that.

similar? for distance based tools $\rightarrow$ how small $d(\mathbf{x}, \mathbf{x}_i)$ is.

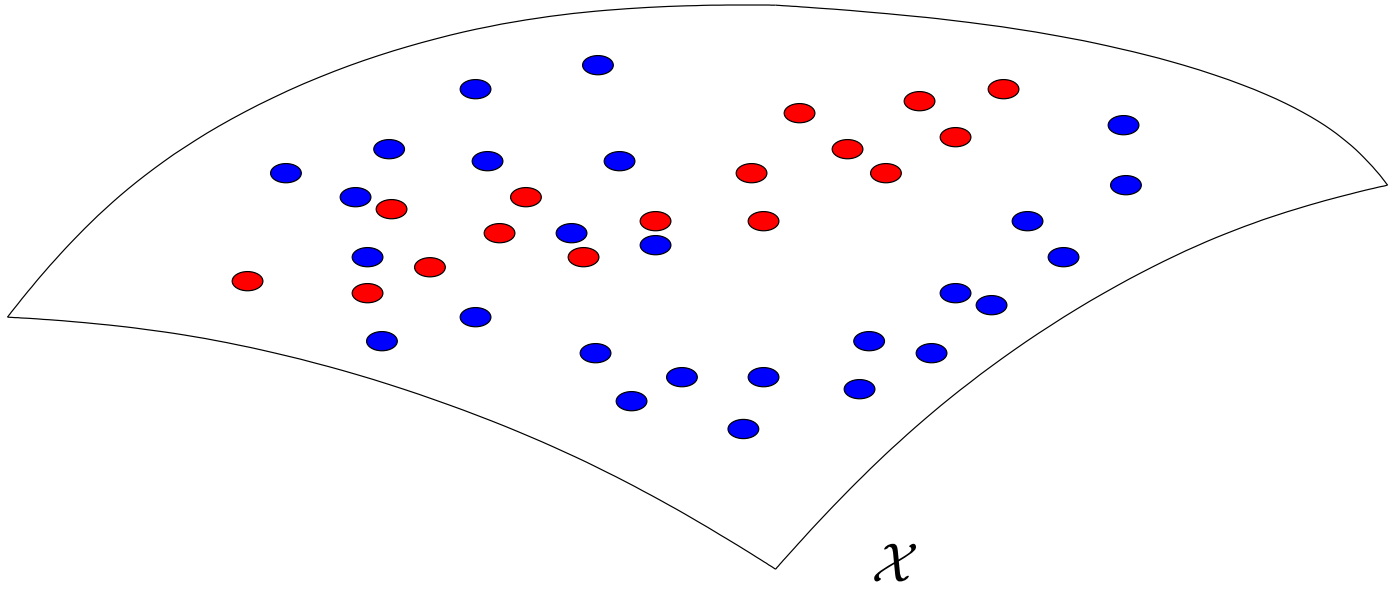
Example: k-nearest neighbors

A very elementary prediction tool

- Used in supervised tasks: given a point $\mathbf{x}$, guess its label y.
- All it takes to use **k-nearest neighbors** is
  - A date sample of labeled points $\{(\mathbf{x}_i, y_i), i = 1, \cdots, N\}$.
  - A **distance function** for two points $d(\mathbf{x}, \mathbf{x}')$.
  - a parameter k that defines the size of the neighborhood.

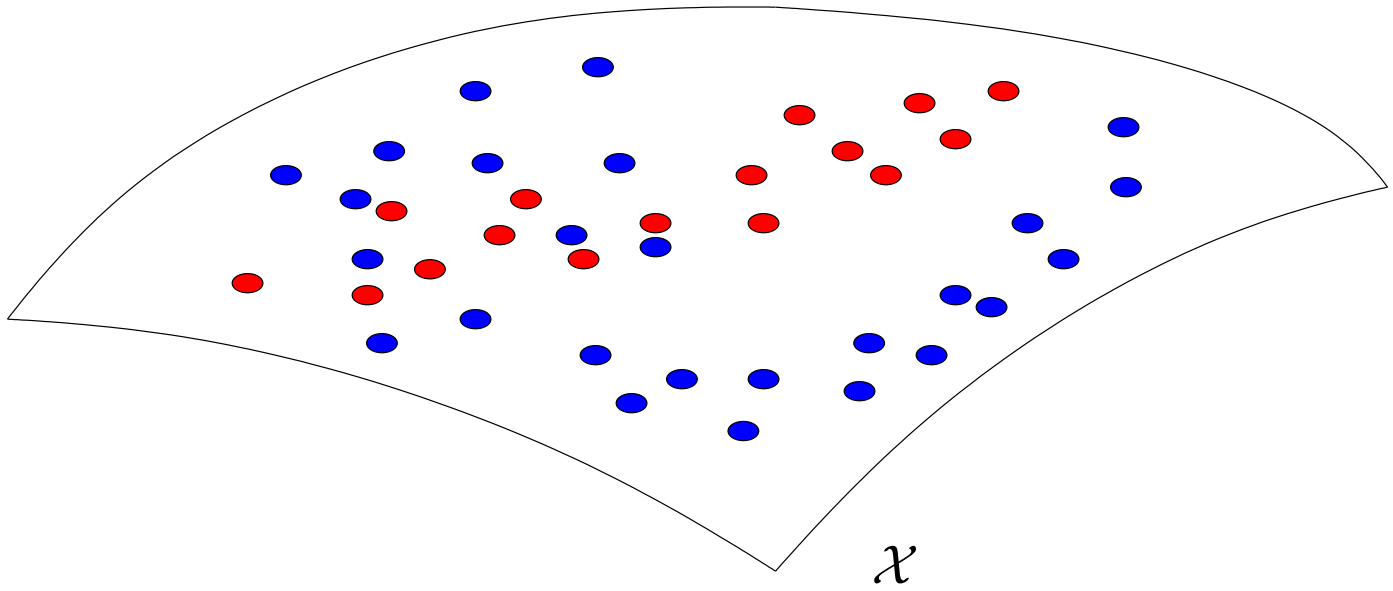
How does it work?

Example: k-nearest neighbors



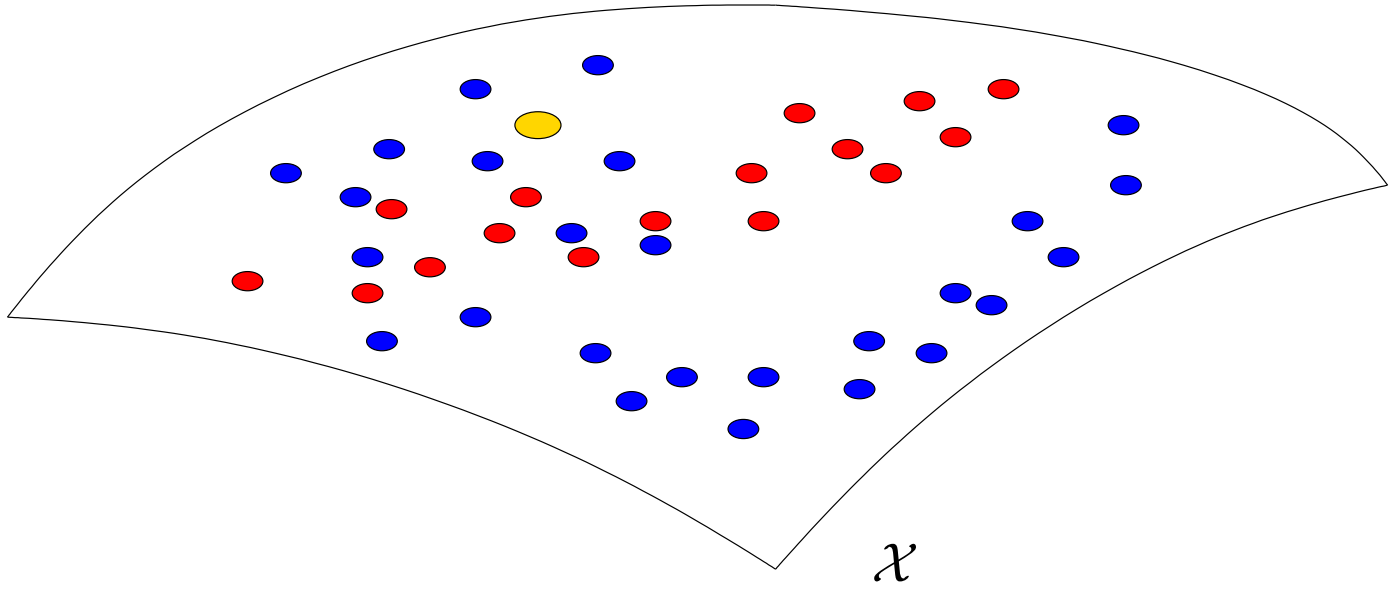
A database represented as a bunch of colored points

Example: k-nearest neighbors



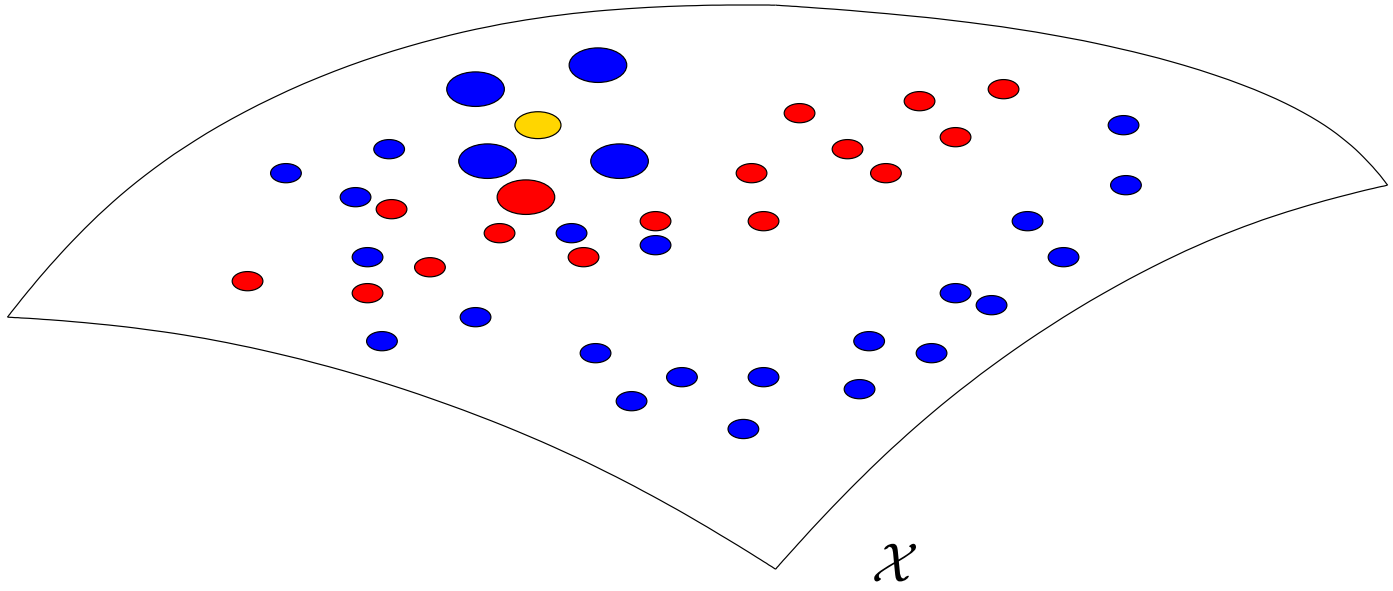
Points are split between **blue** and **red** points.

Example: k-nearest neighbors



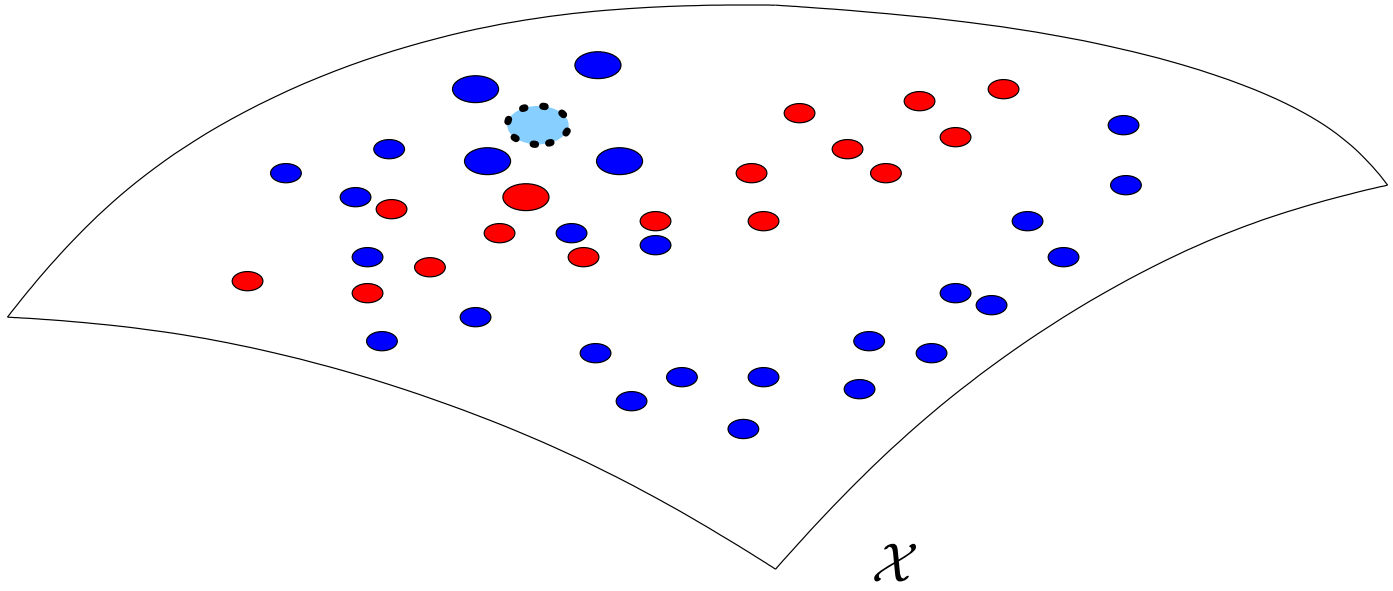
Here we have a new point to label... **blue** or **red**?

Example: k-nearest neighbors



Look at **5** closest neighbors... that is set $k = 5$.

Example: k-nearest neighbors



4 **blue** and 1 **red**: majority vote $\rightarrow$ **5-nearest neighbors**' guess is **blue**.

k nearest neighbors

- A computer program cannot "see" the dataset the way we just did.
- Neither do we when $d > 3$...
- A computer would only rely on
  - the dataset $\{(\mathbf{x}_i, y_i), i = 1, \cdots, N\}$.
  - the distance function d.
  - the parameter k.
- How would this work in practice?
- Suppose $N = 100$ and $k = 3$, labels are **blue** or **red**.

k nearest neighbors

- How would this work in practice?
  - get a new point **x**... **we want to guess its label**

k nearest neighbors

- How would this work in practice?
  - get a new point $\mathbf{x}$.
  - Compute a vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_1) \\ d(\mathbf{x}, \mathbf{x}_2) \\ d(\mathbf{x}, \mathbf{x}_3) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_{100}) \end{bmatrix}$.

k nearest neighbors

- How would this work in practice?
  - get a new point $\mathbf{x}$.
  - Compute a vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_1) \\ d(\mathbf{x}, \mathbf{x}_2) \\ d(\mathbf{x}, \mathbf{x}_3) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_{100}) \end{bmatrix} = \begin{bmatrix} 0.32 \\ 4.56 \\ 0.112 \\ \vdots \\ 2.892 \end{bmatrix}$.

k nearest neighbors

- How would this work in practice?
  - get a new point $\mathbf{x}$.
  - Compute a vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_1) \\ d(\mathbf{x}, \mathbf{x}_2) \\ d(\mathbf{x}, \mathbf{x}_3) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_{100}) \end{bmatrix} = \begin{bmatrix} 0.32 \\ 4.56 \\ 0.112 \\ \vdots \\ 2.892 \end{bmatrix}$.
  - **Sort (at the top)** this vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_3) \\ d(\mathbf{x}, \mathbf{x}_{86}) \\ d(\mathbf{x}, \mathbf{x}_{13}) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_2) \end{bmatrix} = \begin{bmatrix} 0.112 \\ 0.132 \\ 0.133 \\ \vdots \\ 4.56 \end{bmatrix}$.

k nearest neighbors

- How would this work in practice?
  - get a new point $\mathbf{x}$.
  - Compute a vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_1) \\ d(\mathbf{x}, \mathbf{x}_2) \\ d(\mathbf{x}, \mathbf{x}_3) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_{100}) \end{bmatrix} = \begin{bmatrix} 0.32 \\ 4.56 \\ 0.112 \\ \vdots \\ 2.892 \end{bmatrix}$.
  - **Sort (at the top)** this vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_3) \\ d(\mathbf{x}, \mathbf{x}_{86}) \\ d(\mathbf{x}, \mathbf{x}_{13}) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_2) \end{bmatrix} = \begin{bmatrix} 0.112 \\ 0.132 \\ 0.133 \\ \vdots \\ 4.56 \end{bmatrix}$.
  - **Select** the three closest neighbors $\mathbf{x}_3, \mathbf{x}_{86}, \mathbf{x}_{13}$.

k nearest neighbors

- How would this work in practice?
  - get a new point $\mathbf{x}$.
  - Compute a vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_1) \\ d(\mathbf{x}, \mathbf{x}_2) \\ d(\mathbf{x}, \mathbf{x}_3) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_{100}) \end{bmatrix} = \begin{bmatrix} 0.32 \\ 4.56 \\ 0.112 \\ \vdots \\ 2.892 \end{bmatrix}$.
  - **Sort (at the top)** this vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_3) \\ d(\mathbf{x}, \mathbf{x}_{86}) \\ d(\mathbf{x}, \mathbf{x}_{13}) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_2) \end{bmatrix} = \begin{bmatrix} 0.112 \\ 0.132 \\ 0.133 \\ \vdots \\ 4.56 \end{bmatrix}$.
  - **Select** the three closest neighbors $\mathbf{x}_3$, $\mathbf{x}_{86}$, $\mathbf{x}_{13}$ with their labels.

k nearest neighbors

- How would this work in practice?
  - get a new point $\mathbf{x}$.
  - Compute a vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_1) \\ d(\mathbf{x}, \mathbf{x}_2) \\ d(\mathbf{x}, \mathbf{x}_3) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_{100}) \end{bmatrix} = \begin{bmatrix} 0.32 \\ 4.56 \\ 0.112 \\ \vdots \\ 2.892 \end{bmatrix}$.
  - **Sort (at the top)** this vector of distances $\begin{bmatrix} d(\mathbf{x}, \mathbf{x}_3) \\ d(\mathbf{x}, \mathbf{x}_{86}) \\ d(\mathbf{x}, \mathbf{x}_{13}) \\ \vdots \\ d(\mathbf{x}, \mathbf{x}_2) \end{bmatrix} = \begin{bmatrix} 0.112 \\ 0.132 \\ 0.133 \\ \vdots \\ 4.56 \end{bmatrix}$.
  - **Select** the three closest neighbors $\mathbf{x}_3$, $\mathbf{x}_{86}$, $\mathbf{x}_{13}$ with their labels.

The 3-nearest neighbor algorithm outputs a red label for $\mathbf{x}$.

k nearest neighbors

3 blocks of k-nearest neighbor:

- **database** of **colored** points $\{ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \cdots, \mathbf{x}_{13}, \cdots, \mathbf{x}_{100} \}$
- neighborhood size $k = 3$
- a **distance function** d.

the dataset is fixed. k is a simple parameter.
the **distance** is the most challenging to define.

Can we **fine tune** this distance so that we get improved results?

Choosing a Distance for k-nearest neighbors

Choosing a Distance

For many applications,
both training points $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and observations $\mathbf{x}$
are vectors of **features**,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^d.$$

What distances can we use for vectors?

Choosing a Distance

- Euclidean/l_2 distance in $\mathbb{R}^d$

$$d_2(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}.$$

Choosing a Distance

- Euclidean/l_2 distance in $\mathbb{R}^d$

$$d_2(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2} = \sqrt{(\mathbf{x} - \mathbf{x}')^T (\mathbf{x} - \mathbf{x})}$$

- Manhattan/l_1 distance in $\mathbb{R}^d$

$$d_1(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} |x_i - x'_i|.$$

Choosing a Distance

- Euclidean/l_2 distance in $\mathbb{R}^d$

$$d_2(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}.$$

- Manhattan/l_1 distance in $\mathbb{R}^d$

$$d_1(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} |x_i - x'_i|.$$

- Infinite/l_∞ distance in $\mathbb{R}^d$

$$d_\infty(\mathbf{x}, \mathbf{x}') = \max_{i=1..d} |x_i - x'_i|.$$

Choosing a Distance

- Euclidean/l_2 distance in $\mathbb{R}^d$

$$d_2(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}.$$

- Manhattan/l_1 distance in $\mathbb{R}^d$

$$d_1(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} |x_i - x'_i|.$$

- Infinite/l_∞ distance in $\mathbb{R}^d$

$$d_\infty(\mathbf{x}, \mathbf{x}') = \max_{i=1..d} |x_i - x'_i|.$$

- the list of possibilities is **very long** [*Encyclopedia of Distances*, Deza & Deza, 2009]

Learning Distances

Rather than **choose**, **learn** using *data*.

Consider a
parameterized and **rich** enough **family of distances**
rather than a
long and predefined list of candidates

- always preferable if easy to implement and optimize...
- this is the goal of **metric learning**

Learning Distances: Starting Point

- The Euclidean distance,

$$d_2(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2} = \sqrt{(\mathbf{x} - \mathbf{x}')^T (\mathbf{x} - \mathbf{x}')}$$

  can be seen as a particular case of **Mahalanobis distances** (40's),

$$d^{\Omega}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \Omega (\mathbf{x} - \mathbf{x}')}.$$

  a family of distances parameterized by a **positive definite matrix** Ω in $\mathbb{R}^{d \times d}$.

- Indeed, $d_2 = d^I$.

Positive Definiteness

$$d^{\Omega}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \Omega (\mathbf{x} - \mathbf{x}')}.$$

a family of distances parameterized by a **positive definite matrix** Ω in $\mathbb{R}^{d \times d}$.

- Why require that Ω is symmetric positive definite?
  - if Ω is not positive definite $\rightarrow \exists \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}^T \Omega \mathbf{x} < 0$, then for any $\mathbf{y}$,

$$d^{\Omega}(\mathbf{y} + \mathbf{x}, \mathbf{y}) = \mathbf{x}^T \Omega \mathbf{x} < 0 \; !$$

- If Ω is positive **semi**definite and $\exists \mathbf{x} \neq 0 | \mathbf{x}^T \Omega \mathbf{x} = 0$, d^{Ω} is a **pseudo**-metric.
- This is usually ok for most algorithms.

Positive Definiteness

$$d^{\Omega}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \Omega (\mathbf{x} - \mathbf{x}')}.$$

a family of distances parameterized by a **positive definite matrix** Ω in $\mathbb{R}^{d \times d}$.

- Note also that if Ω is positive definite, then $\exists L$ such that

$$\Omega = L^T L \text{ (Cholesky)},$$

  then $d^{\Omega}(\mathbf{x}, \mathbf{x}') = d_2(L\mathbf{x}, L\mathbf{x}')$.

- Choosing a Mahalanobis distance $\Leftrightarrow$ defining an arbitrary linear map L.

Learning Mahalanobis Distances

- Imagine you would like to use Mahalanobis distances,

$$d^{\Omega}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \Omega (\mathbf{x} - \mathbf{x}')}.$$

question : **How to set Ω?**

Learning Mahalanobis Distances

- Imagine you would like to use Mahalanobis distances,

$$d^{\Omega}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \Omega (\mathbf{x} - \mathbf{x}')}.$$

question : **How to set Ω?**

- Up to 10 years ago, basically set $\Omega = \hat{\Sigma}^{-1}$, inverse of empirical variance.
- Computed using a dataset $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and

$$\hat{\Sigma} = \frac{1}{N-1} \sum_{i=1}^{N} \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^T,$$

  where $\tilde{\mathbf{x}}_i \stackrel{\text{def}}{=} \mathbf{x}_i - \bar{\mathbf{x}}$ and $\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$

- also known as *whitening* the data.

Learning Mahalanobis Distances

- Imagine you would like to use Mahalanobis distances,

$$d^{\Omega}(\mathbf{x}, \mathbf{x}') = \sqrt{(\mathbf{x} - \mathbf{x}')^T \Omega (\mathbf{x} - \mathbf{x}')}.$$

question : **How to set Ω?**

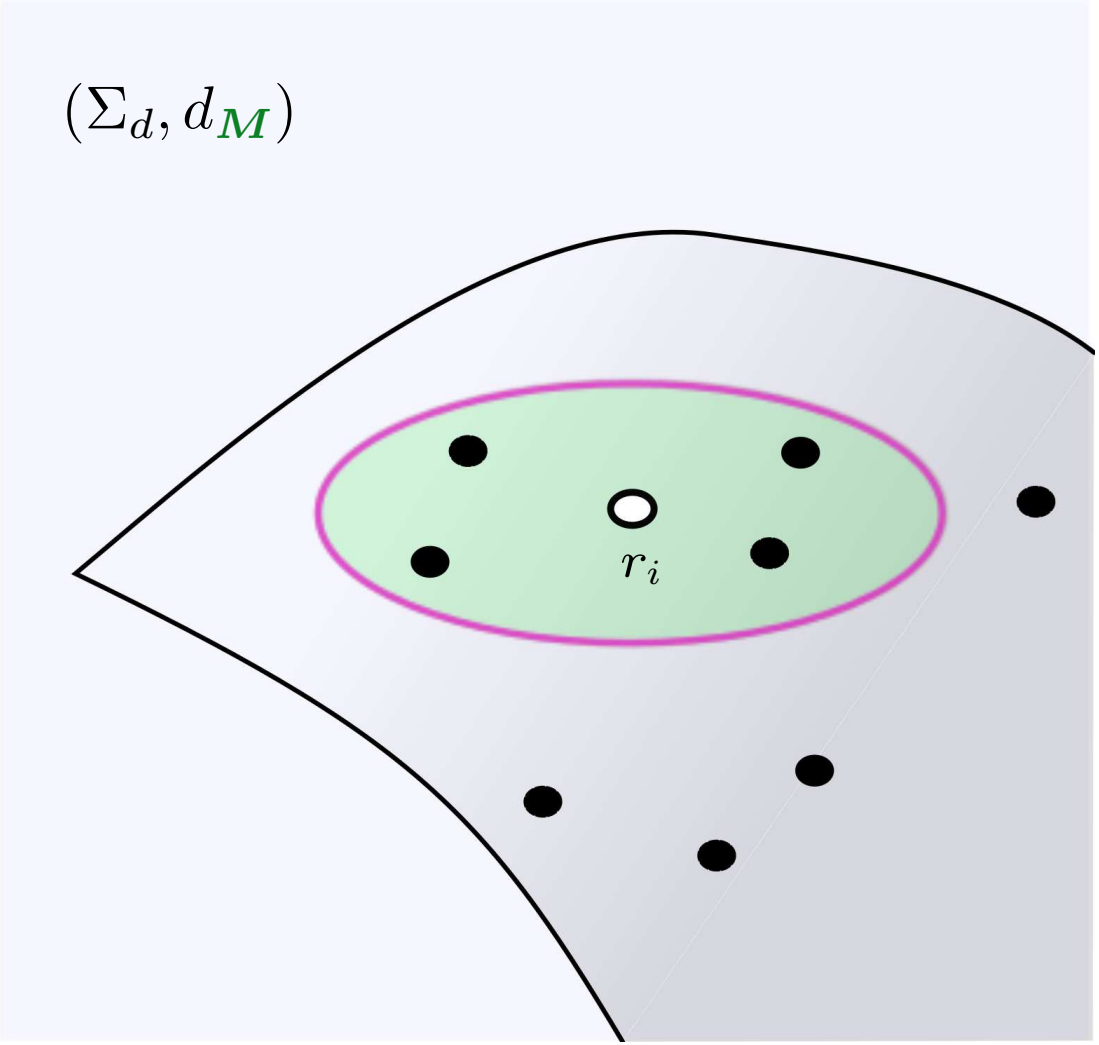
- **New answer in recent machine learning research** (>2003),
  - Distance Learning (Xing et al. 2003)
  - Pseudo-Metric Online Learning Algorithm (Shalev-Schwartz et al., 2004)
  - Large Margin Nearest Neighbor (Weinberger & Saul, 2006),
  - Information Theoric Metric Learning (Kulis et al., 2007),
  - *etc.*

Works well for *many* datasets and vectors...

Basic Idea Behind Metric Learning

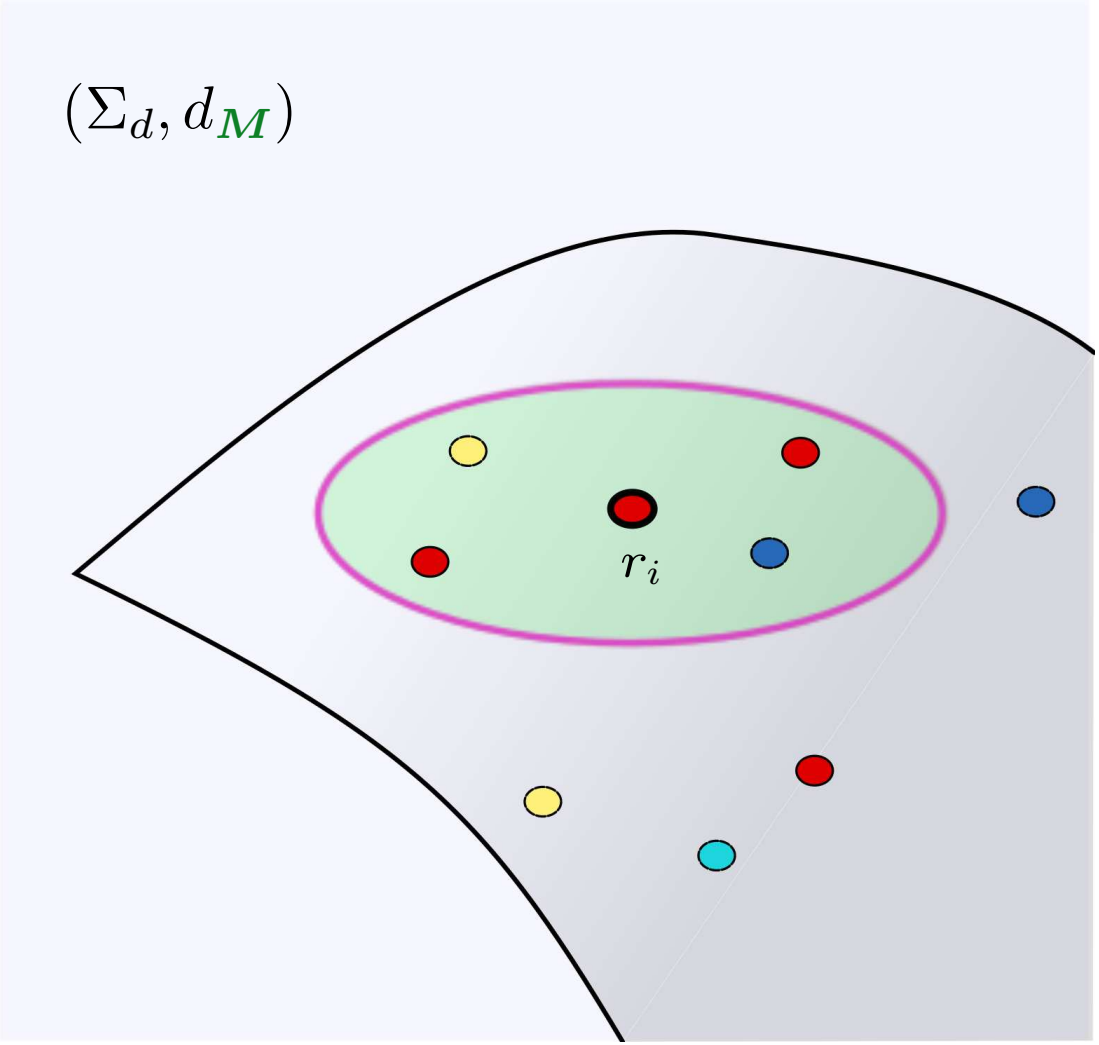
Learning a parameter M

Consider neighborhood of the i^{th} datum induced by $d_{\boldsymbol{M}}$



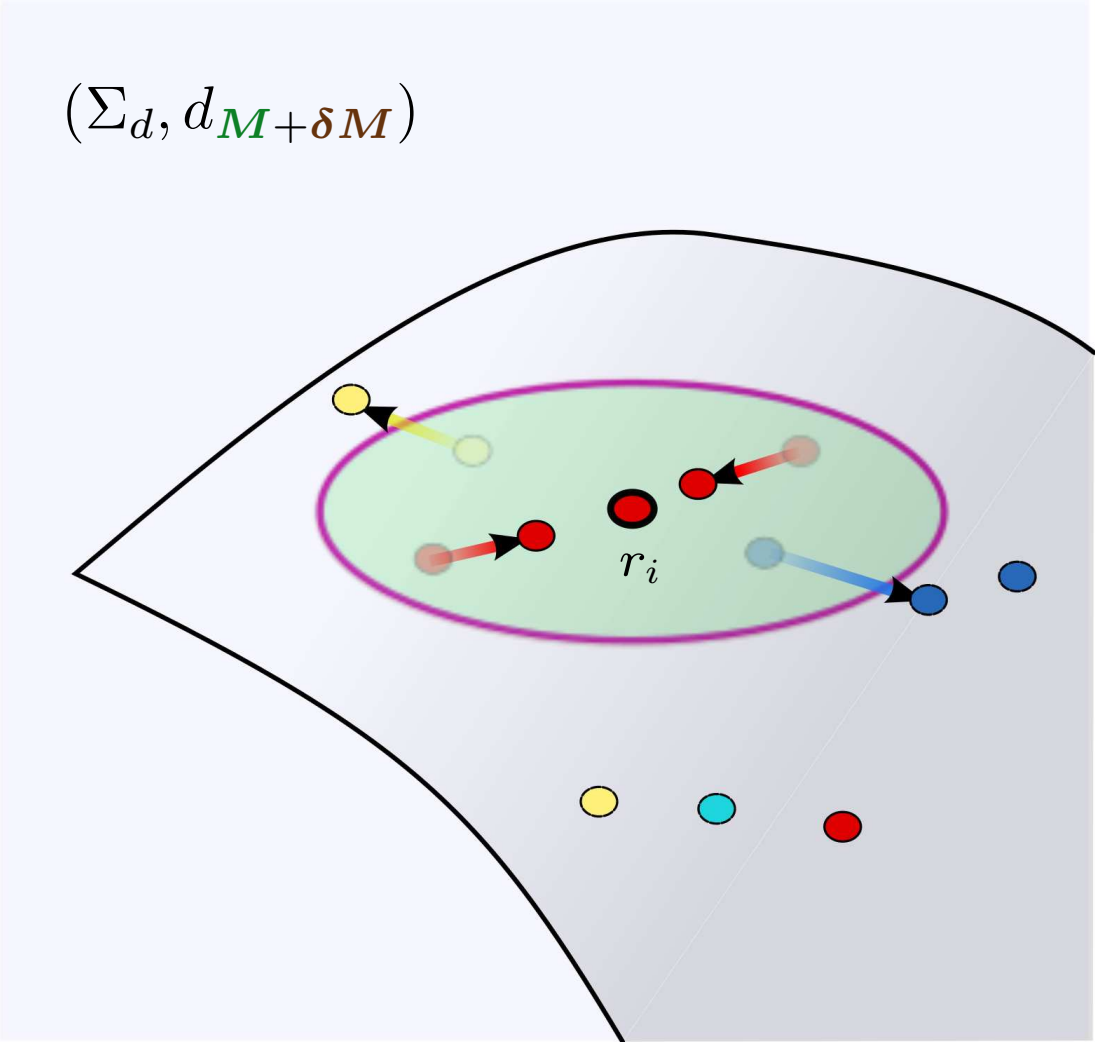
Learning a parameter M

Assume we have side-information, typically labels



Learning a parameter M

change M to $M + \delta M$ to improve local geometry



trick proposed in the metric learning literature: **[Xing et al. '03, Weinberger Saul '06]**

Mathematical Tools: Semidefinite Programming

Linear Programs

- $x \in \mathbb{R}^d, A \in \mathbb{R}^{d \times m}, \mathbf{b} \in \mathbb{R}^m$.
- Consider the **positive orthant** $\mathbb{R}^{\boldsymbol{d}}_+$

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T \mathbf{x} \\ \text{subject to} & A\mathbf{x}\{=, \leq, \geq\}\mathbf{b} \\ & \mathbf{x} \in \mathbb{R}^{\boldsymbol{d}}_+ \end{array}$$

- History: simplex method (Danzig,'47), interior point methods (80's)

Conic Linear Program

- $x \in \mathbb{R}^d, A \in \mathbb{R}^{d \times m}, \mathbf{b} \in \mathbb{R}^m$.
- Consider an arbitrary **cone** $\mathbf{K}$.

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T \mathbf{x} \\ \text{subject to} & A\mathbf{x}\{=, \leq, \geq\}\mathbf{b} \\ & \mathbf{x} \in \mathbf{K} \end{array}$$

- History: 90's. Interior point methods.
- When $\mathbf{K}$ is the cone of positive semidefinite matrices, **semidefinite program**.

In practice

Inputs: $\{ d^{\Omega}(\mathbf{x}_i, \mathbf{x}_j),\ \mathbf{x}_i, \mathbf{x}_j \in T\}$

In practice

Inputs: $\{ d^{\Omega}(\mathbf{x}_i, \mathbf{x}_j),\ \mathbf{x}_i, \mathbf{x}_j \in T\}$

- As a function of Ω, $d^{\Omega}(\mathbf{x}, \mathbf{x}')$ is **concave** w.r.t. Ω

$$d^{\Omega}(\mathbf{x}, \mathbf{x}') = \sqrt{\langle \Omega, (\mathbf{x} - \mathbf{x}')(\mathbf{x} - \mathbf{x}')^T \rangle}.$$

In practice

Inputs: $\{ d^{\Omega}(\mathbf{x}_i, \mathbf{x}_j),\ \mathbf{x}_i, \mathbf{x}_j \in T\}$

- As a function of Ω, $d^{\Omega}(\mathbf{x}, \mathbf{x}')$ is **concave** w.r.t. Ω

$$d^{\Omega}(\mathbf{x}, \mathbf{x}') = \sqrt{\langle \Omega, (\mathbf{x} - \mathbf{x}')(\mathbf{x} - \mathbf{x}')^T \rangle}.$$

- **squared** distance (**not the distance itself**) is linear w.r.t. Ω

$$d^{\Omega}(\mathbf{x}, \mathbf{x}')^2 = \langle \Omega, (\mathbf{x} - \mathbf{x}')(\mathbf{x} - \mathbf{x}')^T \rangle.$$

In practice

Inputs: $\{ d^{\Omega}(\mathbf{x}_i, \mathbf{x}_j),\ \mathbf{x}_i, \mathbf{x}_j \in T\}$

- **squared** distance (**not the distance itself**) is linear w.r.t. Ω

$$d^{\Omega}(\mathbf{x}, \mathbf{x}')^2 = \langle \Omega, (\mathbf{x} - \mathbf{x}')(\mathbf{x} - \mathbf{x}')^T \rangle.$$

- for two points $\mathbf{x}, \mathbf{y}$, constraints of the kind

$$d^{\Omega}(\mathbf{x}, \mathbf{y})^2 \geq \varepsilon \Leftrightarrow \langle \Omega, (\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T \rangle \geq \varepsilon$$

  while for three points $\mathbf{x}, \mathbf{y}, \mathbf{z}$,

$$d^{\Omega}(\mathbf{x}, \mathbf{z}) \leq d^{\Omega}(\mathbf{x}, \mathbf{y}) \Leftrightarrow \langle \Omega, (\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T - (\mathbf{x} - \mathbf{z})(\mathbf{x} - \mathbf{z})^T \rangle \geq 0$$

In practice

Inputs: $\{ d^{\Omega}(\mathbf{x}_i, \mathbf{x}_j),\ \mathbf{x}_i, \mathbf{x}_j \in T\}$

- **squared** distance (**not the distance itself**) is linear w.r.t. Ω

$$d^{\Omega}(\mathbf{x}, \mathbf{x}')^2 = \langle \Omega, (\mathbf{x} - \mathbf{x}')(\mathbf{x} - \mathbf{x}')^T \rangle.$$

- for two points $\mathbf{x}, \mathbf{y}$, constraints of the kind

$$d^{\Omega}(\mathbf{x}, \mathbf{y})^2 \geq \varepsilon \Leftrightarrow \langle \Omega, (\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T \rangle \geq \varepsilon$$

  while for three points $\mathbf{x}, \mathbf{y}, \mathbf{z}$,

$$d^{\Omega}(\mathbf{x}, \mathbf{z}) \leq d^{\Omega}(\mathbf{x}, \mathbf{y}) \Leftrightarrow \langle \Omega, (\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^T - (\mathbf{x} - \mathbf{z})(\mathbf{x} - \mathbf{z})^T \rangle \geq 0$$

- Falls into a SDP setting... as long as **squared** distances are always considered...
- Some implementations proposed in the literature use dedicated solvers.

Examples

Metric Learning (Xing,Ng,Jordan,Russel 2003)

- Elementary idea:
  - pull all pairs of similar points **together** ($\mathcal{S}$),
  - push all pairs of dissimilar points **apart** ($\mathcal{D}$)

$$\max_{\mathbf{\Omega} \succeq 0} \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S}} d^{\mathbf{\Omega}}(\mathbf{x}_i, \mathbf{x}_j)^2$$

$$\text{such that} \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{D}} d^{\mathbf{\Omega}}(\mathbf{x}_i, \mathbf{x}_j) \geq 1$$

- $d^{\mathbf{\Omega}}$ is concave, so the constraint is convex
- The objective is linear in $\mathbf{\Omega}$.
- If use $d^{\mathbf{\Omega}}(\cdot, \cdot)^2$ in the constraint, problem is degenerate.

Pseudo-Metric Online Learning Algorithm (POLA, 2004)

- Considers and online setting where pairs $\mathbf{x}_t, \mathbf{x}'_t$ that are similar ($y_t = 1$) or dissimilar ($y_t = -1$) are received sequentially at time t.
- A new Ω is selected at each iteration t depending on previous observations.
- Objective: minimize, up to T,

$$L = \sum_{t=1}^{T} l_t(\boldsymbol{\Omega_t}, \boldsymbol{b_t})$$

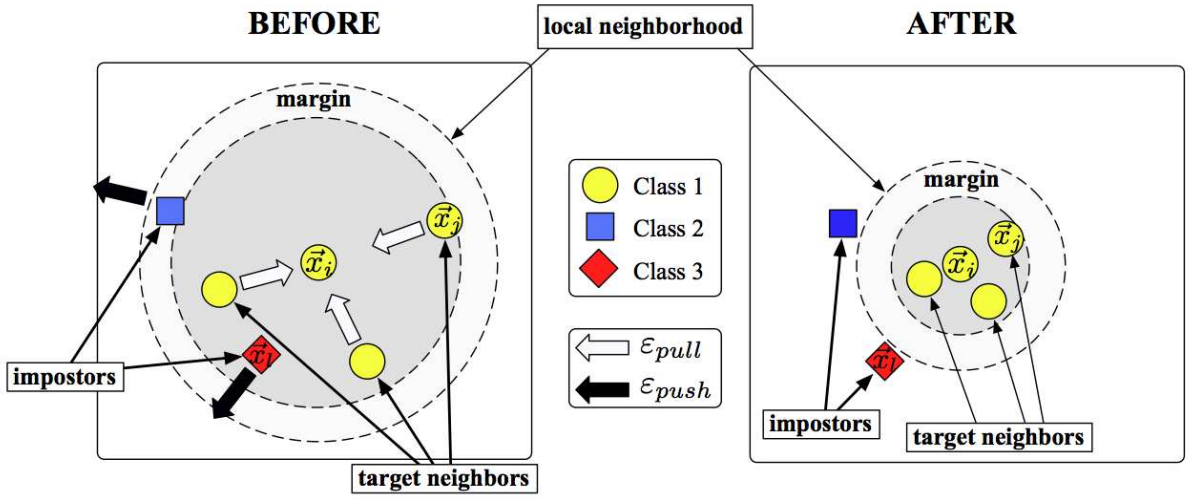
where

$$l_t(\boldsymbol{\Omega}, \boldsymbol{b}) \stackrel{\text{def}}{=} \max\left(0, y_t(d^{\boldsymbol{\Omega}}(\mathbf{x}_t, \mathbf{x}'_t)^2 - \boldsymbol{b}) + 1\right).$$

- To do so, use a simple updating step of Ω_t:
  - Let $\Omega_t \leftarrow \Omega_{t-1} - y_t \alpha_t (\mathbf{x}_t - \mathbf{x}'_t)(\mathbf{x}_t - \mathbf{x}'_t)^T$
  - Project on cone of positive definite matrices by thresholding

Large Margin Nearest Neighbor (LMNN, 2006)

- Original idea



- LMNN solves the following SDP:

$$\min_{\mathbf{\Omega}} \quad (1-\mu)\sum_{i,j\rightsquigarrow i}(\mathbf{x}_i-\mathbf{x}_j)^T\mathbf{\Omega}(\mathbf{x}_i-\mathbf{x}_j)+\mu\sum_{i,j\rightsquigarrow i,l}(1-y_{il})\xi_{ijl}$$

$$\text{such that}(\mathbf{x}_i-\mathbf{x}_j)^T\mathbf{\Omega}(\mathbf{x}_i-\mathbf{x}_j)-(\mathbf{x}_i-\mathbf{x}_l)^T\mathbf{\Omega}(\mathbf{x}_i-\mathbf{x}_l)\geq 1-\xi_{ijl}$$

$$\xi_{ijl}\geq 0$$

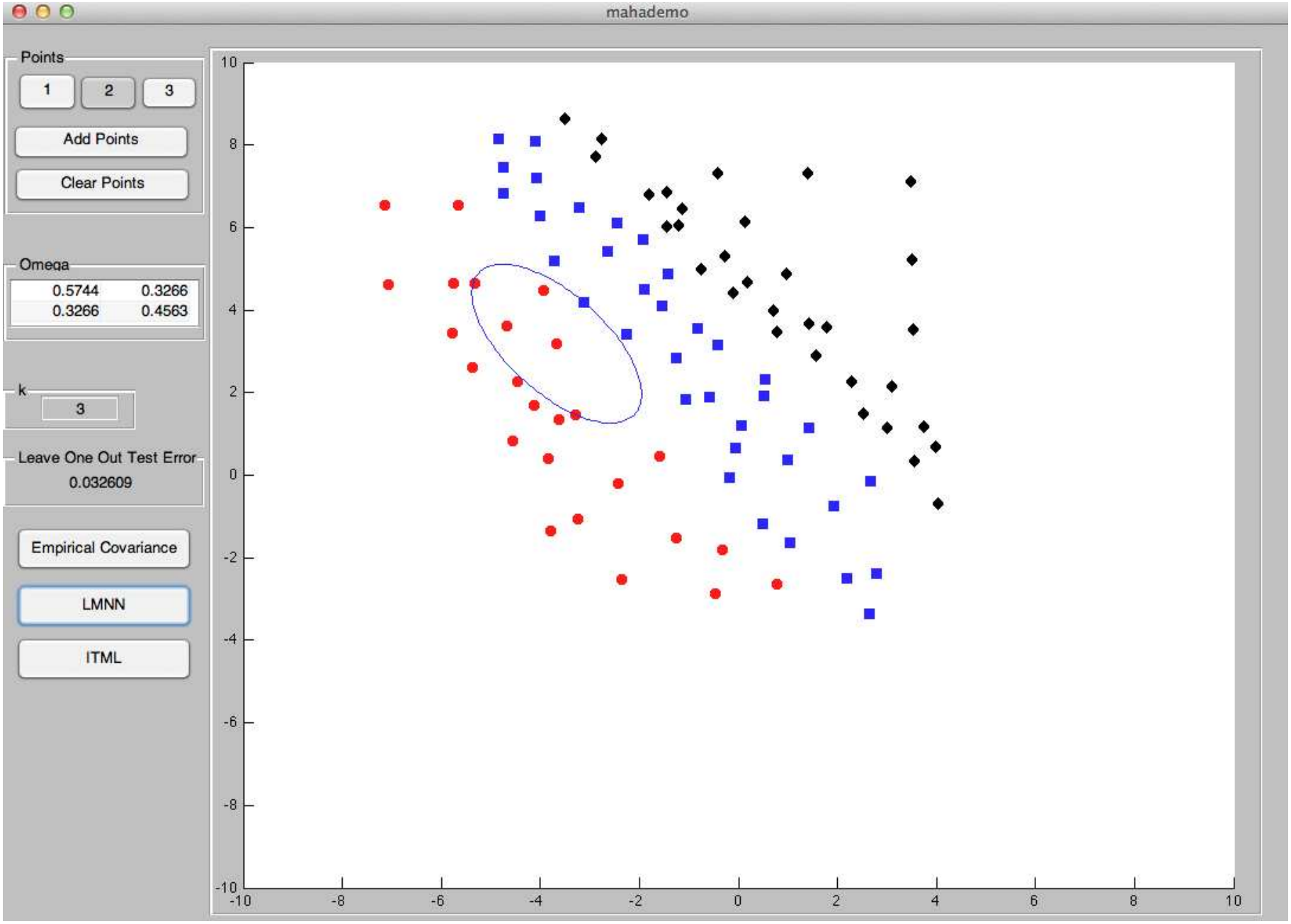
Information Theoric Metric Learning (ITML, 2007)

- Suppose we have a **prior candidate** for $\mathbf{\Omega}$ (e.g.$\hat{\Sigma}^{-1}$)
- Call this prior candidate $\mathbf{\Omega_0}$.
- ITML proposes to find an $\mathbf{\Omega}$ that is not too far from $\mathbf{\Omega_0}$ while still splitting similar and dissimilar points apart:

$$\min_{\mathbf{\Omega}} \quad \mathcal{L}(\mathbf{\Omega}, \mathbf{\Omega_0})$$
$$\text{such that for } (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S}, d^{\mathbf{\Omega}}(\mathbf{x}_i, \mathbf{x}_j)^2 \leq u$$
$$\text{such that for } (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{D}, d^{\mathbf{\Omega}}(\mathbf{x}_i, \mathbf{x}_j) \geq l$$

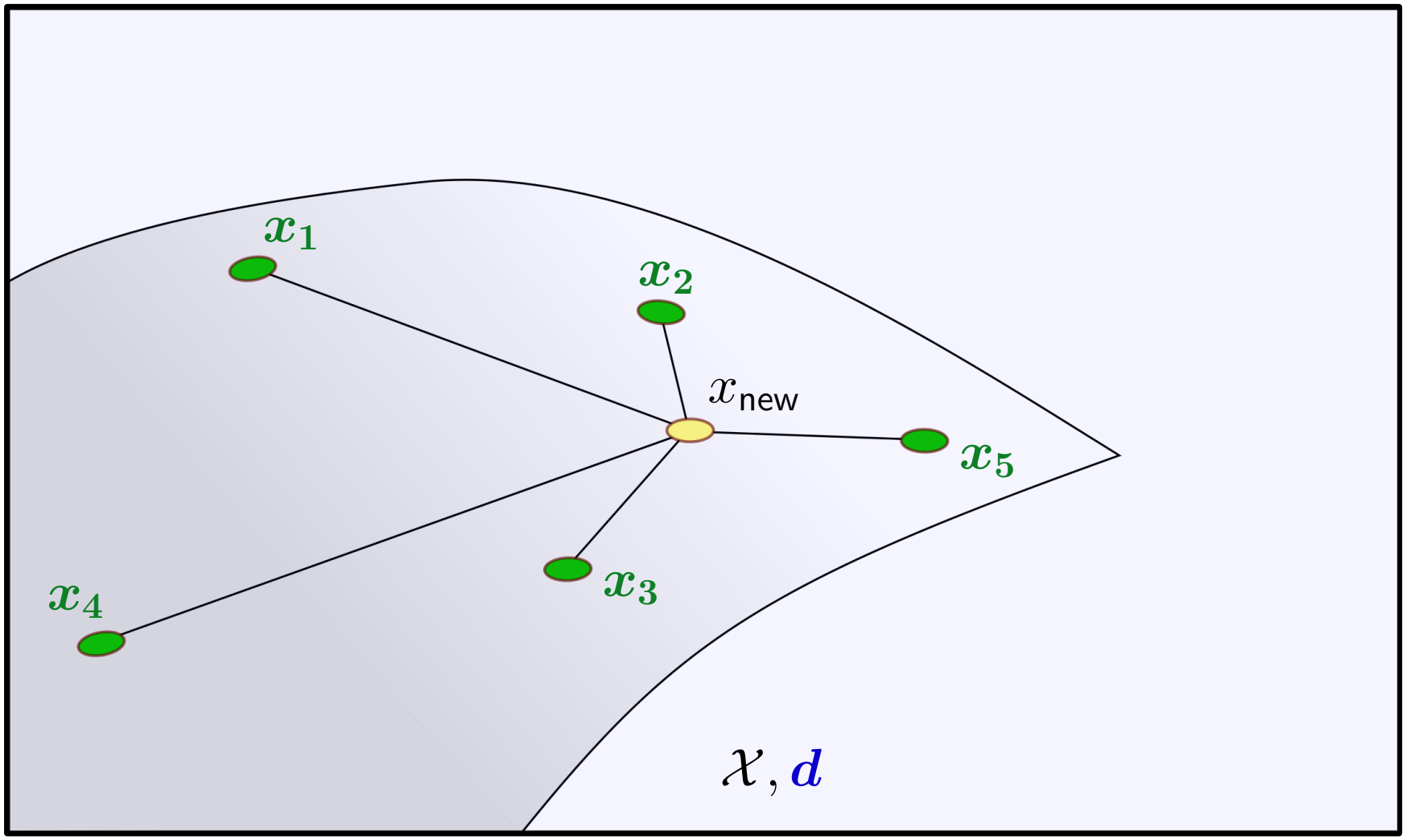
where $\mathcal{L}(A, B) \stackrel{\text{def}}{=} \text{tr}(AB^{-1}) - \log\det(AB^{-1})$ is a matrix-divergence

Illustration



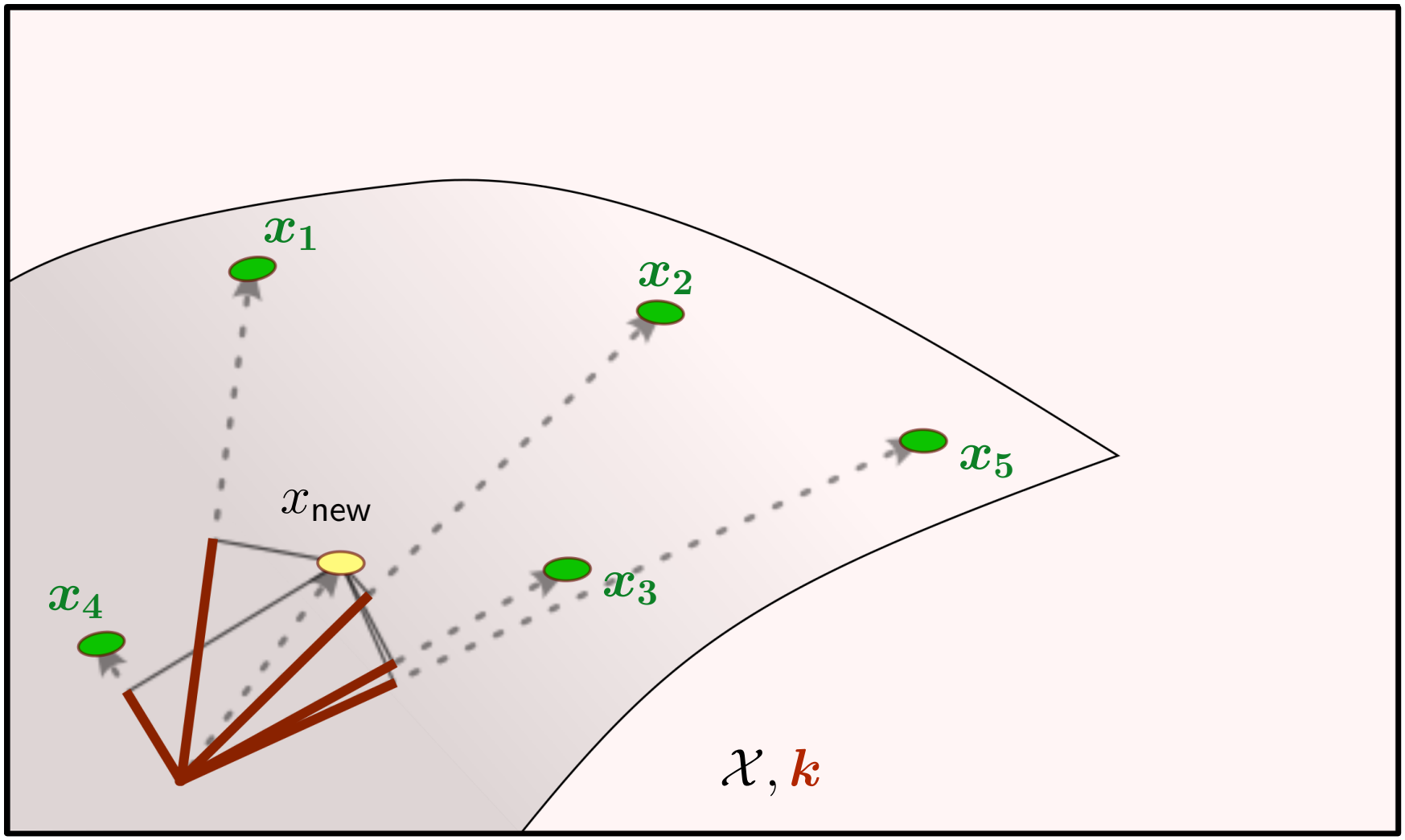
Kernels

While Distance Based Algorithms...



... rely **exclusively** on all distances $\{d(x_{\text{new}}, x_i), i = 1, \cdots, 5\}$

Kernel Based Algorithms...



... rely exclusively on **dot-products** $k(x_{\text{new}}, x_i), \quad i = 1, \cdots, 5$

Positive Definite Kernels

A bivariate function defined on a set $\mathcal{X}$

$$\begin{array}{rccc} \boldsymbol{k}: & \mathcal{X}\times\mathcal{X} & \to & \mathbb{R}_+ \\ & (\mathbf{x},\mathbf{y}) & \mapsto & \boldsymbol{k}(\mathbf{x},\mathbf{y}) \end{array}$$

is a **positive definite kernel** if $\forall\, \mathbf{x},\mathbf{y}\in\mathcal{X}$,

- $\boldsymbol{k}(\mathbf{x},\mathbf{y}) = \boldsymbol{k}(\mathbf{y},\mathbf{x}), \to$ *symmetry*
- and $\forall\, n\in\mathbb{N}, \{\mathbf{x}_1,\cdots,\mathbf{x}_n\}\in\mathcal{X}^n, c\in\mathbb{R}^n$,

$$\sum_{i=1}^{n} c_i\, c_j\, \boldsymbol{k}(\mathbf{x}_i,\mathbf{x}_j) \geq 0, \to \textit{positive definiteness}$$

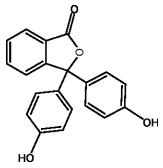
$$\text{equivalently}\quad K = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & \cdots & k(\mathbf{x}_1,\mathbf{x}_i) & \cdots & k(\mathbf{x}_1,\mathbf{x}_n) \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ k(\mathbf{x}_i,\mathbf{x}_1) & \cdots & k(\mathbf{x}_i,\mathbf{x}_i) & \cdots & k(\mathbf{x}_2,\mathbf{x}_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n,\mathbf{x}_1) & \cdots & k(\mathbf{x}_n,\mathbf{x}_i) & \cdots & k(\mathbf{x}_n,\mathbf{x}_n) \end{bmatrix} \succeq 0$$

is positive semidefinite (has a nonnegative spectrum).

In the context of these lectures...

- A kernel k is a function

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

- which compares two objects of a space $\mathcal{X}$, $e.g.$...
  - strings, texts and sequences, 
  - images, audio and video feeds,  
  - graphs, interaction networks and 3D structures 
- whatever actually... time-series of graphs of images? graphs of texts?...

What can we do with a kernel?

The setting

- Pretty simple setting: a set of objects $\mathbf{x}_1, \cdots, \mathbf{x}_n$ of $\mathcal{X}$
- **Sometimes** additional information on these objects
  - labels $\mathbf{y}_i \in \{-1, 1\}$ or $\{1, \cdots, \#(\text{classes})\}$,
  - scalar values $\mathbf{y}_i \in \mathbb{R}$,
  - associated object $\mathbf{y}_i \in \mathcal{Y}$
- A kernel $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$.

A few intuitions on the possibilities of kernel methods

Important concepts and perspectives

- The functional perspective: represent **points as functions**.
- **Nonlinearity** : linear combination of kernel evaluations.
- Summary of a sample through its **kernel matrix**.

Represent any point in $\mathcal{X}$ as a function

For every **x**, the map
$\mathbf{x} \longrightarrow k(\mathbf{x}, \cdot)$
associates to **x** a function $k(\mathbf{x}, \cdot)$ from $\mathcal{X}$ to $\mathbb{R}$.

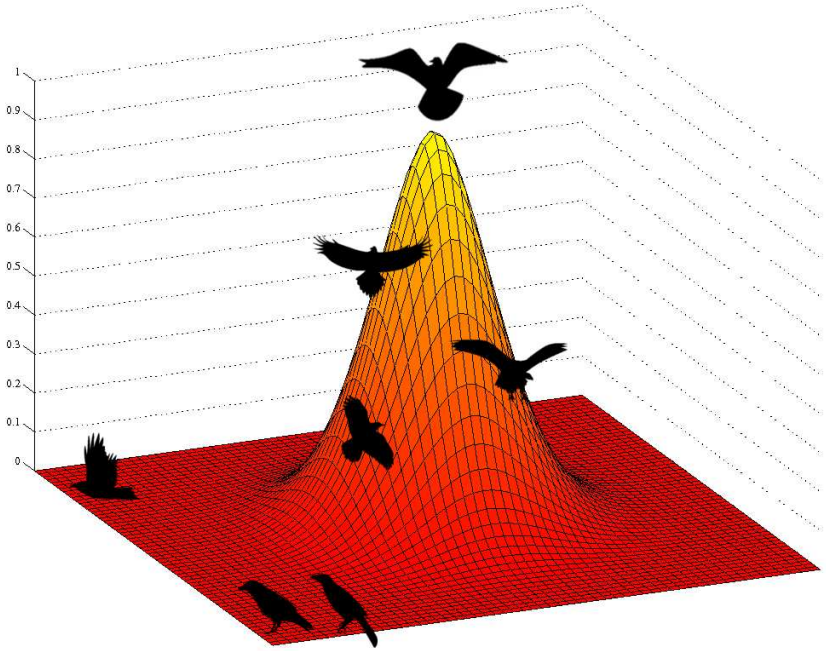
- Suppose we have a kernel k on bird images

- Suppose for instance

$$k\left(\text{  }, \text{  } \right) = .32$$

Represent any point in $\mathcal{X}$ as a function

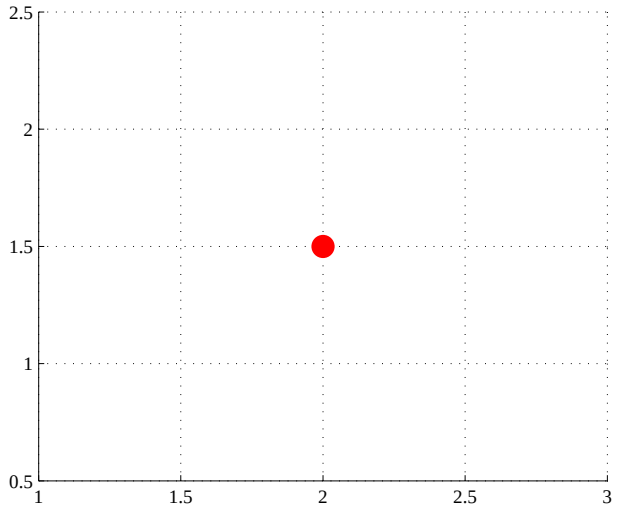
- We examine one image in particular: 
- With kernels, we get a **representation** of that bird as a real-valued function, defined on the space of birds, represented here as $\mathbb{R}^2$ for simplicity.



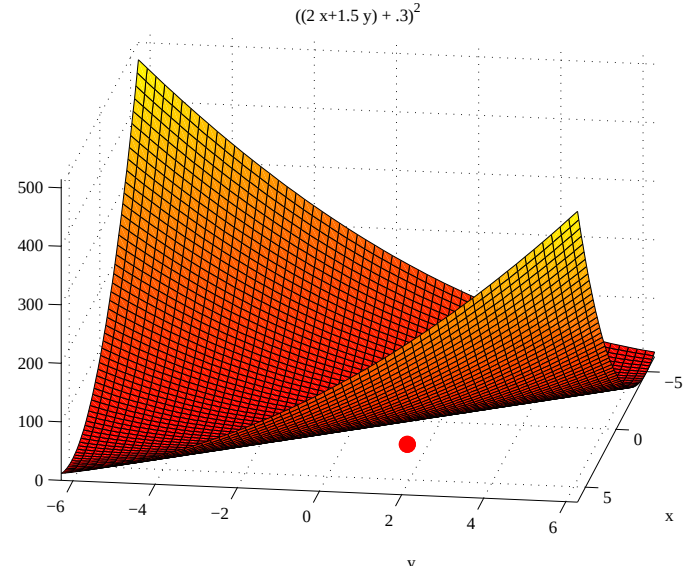
schematic plot of $k\,($  $, \cdot\,)$.

Represent any point in $\mathcal{X}$ as a function

- If the bird example was confusing...
- $k\left(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} x' \\ y' \end{bmatrix}\right) = \left(\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} + .3\right)^2$
- From a point in $\mathbb{R}^2$ to a function defined over $\mathbb{R}^2$.



$\rightarrow$



- We assume implicitly that the **functional representation** will be more useful than the **original representation**.

Decision functions as linear combination of kernel evaluations

- Linear decisions functions are a major tool in statistics, that is functions

$$f(\mathbf{x}) = \beta^T \mathbf{x} + \beta_0.$$

- Implicitly, a point $\mathbf{x}$ is processed depending on its characteristics x_i,

$$f(\mathbf{x}) = \sum_{i=1}^{d} \boldsymbol{\beta_i} x_i + \boldsymbol{\beta_0}.$$

  the free parameters are scalars $\boldsymbol{\beta_0}, \boldsymbol{\beta_1}, \cdots, \boldsymbol{\beta_d}$.

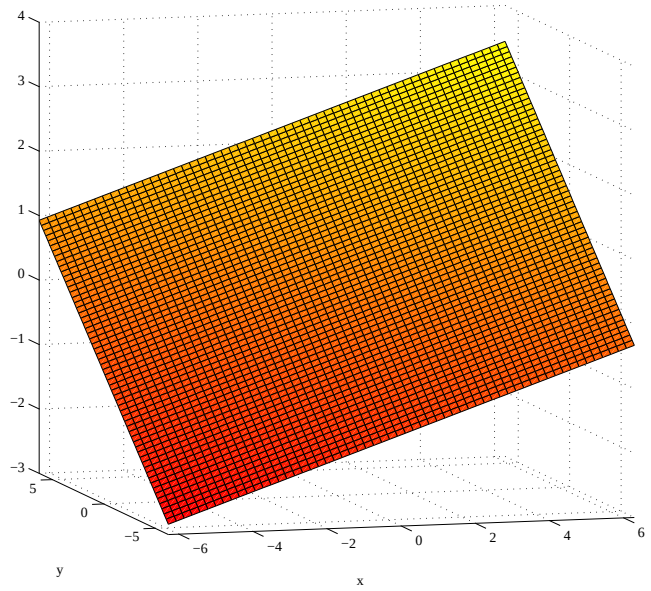
- Kernel methods yield candidate decision functions

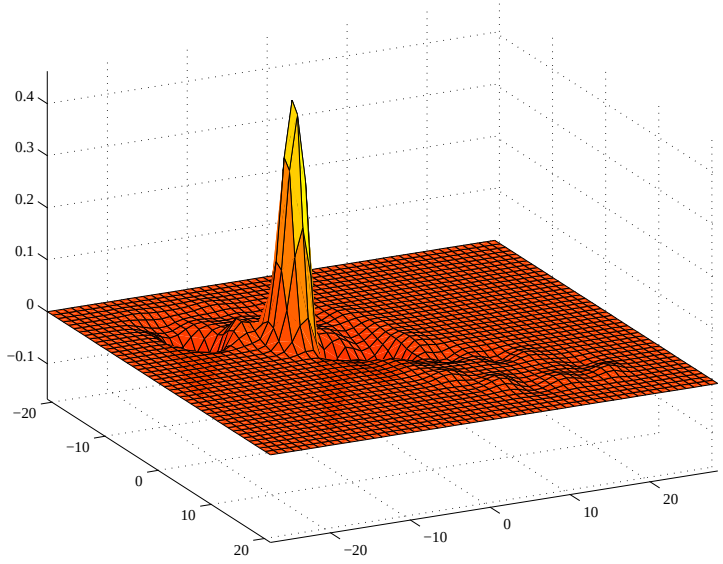
$$f(\mathbf{x}) = \sum_{j=1}^{\boldsymbol{n}} \boldsymbol{\alpha_j} k(\mathbf{x}_j, \mathbf{x}) + \boldsymbol{\alpha_0}.$$

  the free parameters are scalars $\boldsymbol{\alpha_0}, \boldsymbol{\alpha_1}, \cdots, \boldsymbol{\alpha_n}$.

Decision functions as linear combination of kernel evaluations

- linear decision surface / linear expansion of **kernel surfaces** (here $k_G(\mathbf{x}_i, \cdot)$)

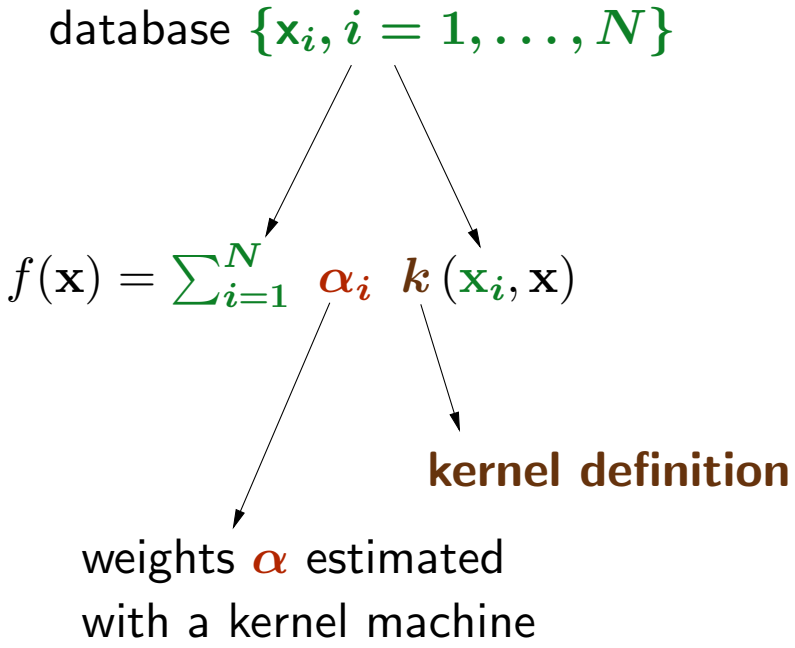




- Kernel methods are considered **non-linear** tools.
- Yet not completely "nonlinear" $\rightarrow$ only one-layer of nonlinearity.

kernel methods use the data as a functional base to define decision functions

Decision functions as linear combination of kernel evaluations



- f is any predictive function of interest of a new point $\mathbf{x}$.
- Weights α are **optimized** with a kernel machine ($e.g.$ support vector machine)

intuitively, kernel methods provide decisions based on how *similar* a point x is to each instance of the training set

The Gram matrix perspective

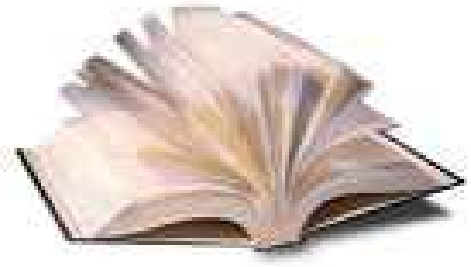
- Imagine a little task: you have read 100 novels so far.

- You would like to know whether you will enjoy reading a **new** novel.
- A few options:
  - read the book...
  - have friends read it for you, read reviews.
  - try to guess, based on the novels you read, if you will like it

The Gram matrix perspective

Two distinct approaches

- Define what **features** can characterize a book.
  - Map each book in the library onto vectors



$$\longrightarrow \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

typically the x_i's can describe...

- ▷ # pages, language, year 1st published, country,
- ▷ coordinates of the main action, keyword counts,
- ▷ author's prizes, popularity, booksellers ranking

- Challenge: find a decision function using 100 ratings and features.

The Gram matrix perspective

- Define what makes **two novels similar**,
  - Define a kernel k which quantifies novel similarities.
  - Map the library onto a Gram matrix

$$\longrightarrow \quad K = \begin{bmatrix} k(b_1, b_1) & k(b_1, b_2) & \cdots & k(b_1, b_{100}) \\ k(b_2, b_1) & k(b_2, b_2) & \cdots & k(b_2, b_{100}) \\ \vdots & \vdots & \ddots & \vdots \\ k(b_n, b_1) & k(b_n, b_2) & \cdots & k(b_{100}, b_{100}) \end{bmatrix}$$

- Challenge: find a decision function that takes this 100×100 matrix as an input.

The Gram matrix perspective

Given a new novel,

- with the **features approach**, the prediction can be rephrased as **what are the features of this new book**? what **features** have I found in the past that were good indicators of my taste?

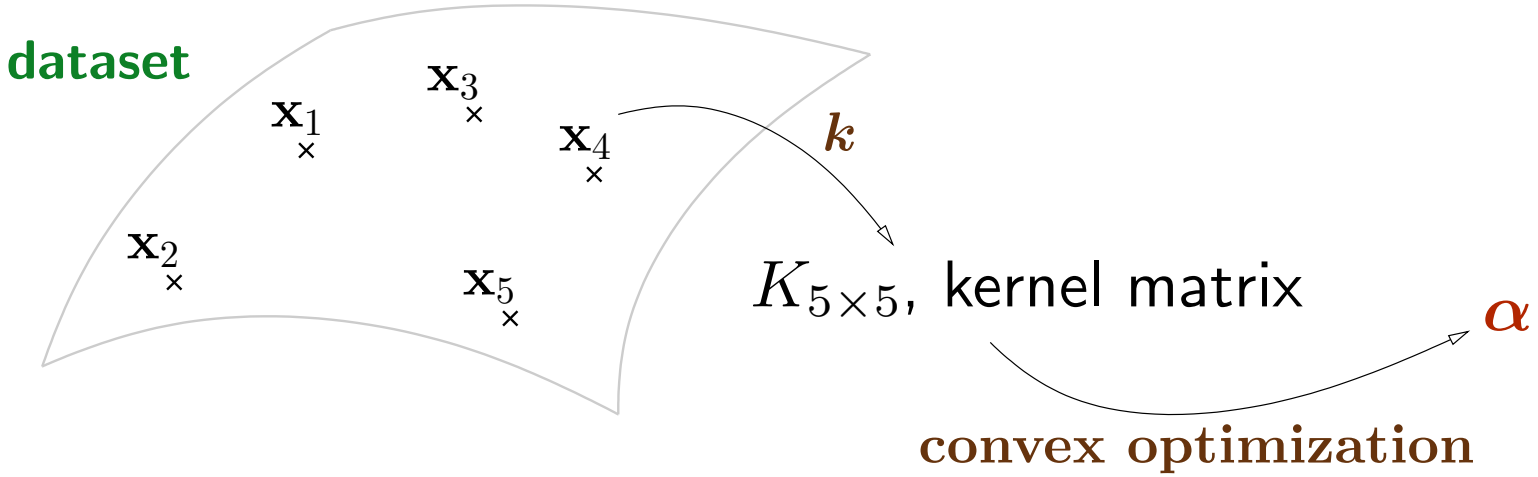
- with the **kernel approach**, the prediction is rephrased as **which novels this book is similar or dissimilar to?** what **pool of books** did I find the most influentials to define my tastes accurately?

kernel methods **only use kernel similarities**, do not consider features.

Features can help define similarities, but **never considered elsewhere**.

The Gram matrix perspective

in kernel methods, clear separation between the kernel...



and **Convex optimization** (thanks to psdness of K, more later) to output the α's.

Example of a Kernel Machine: Classification with the Support Vector Machine

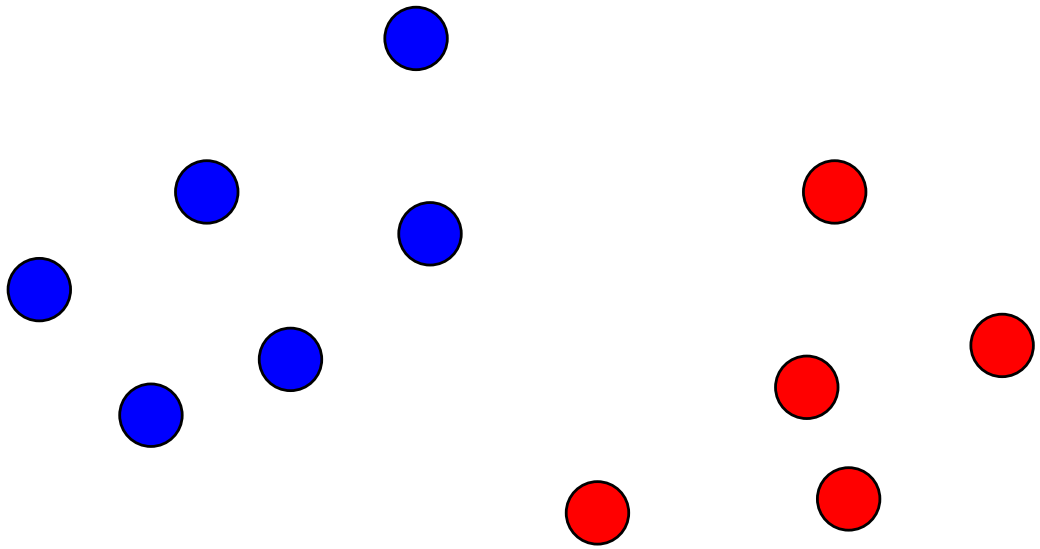
Data

- **Data** : vectors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_N$.
- Ideally, to infer a "yes/no" rule, we need the **correct answer** for each vector.
- We consider thus a set of **pairs of (vector,bit)**

$$\text{"training set"} = \left\{ \left(\mathbf{x}_i = \begin{bmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_d^i \end{bmatrix} \in \mathbb{R}^d, \ \mathbf{y}_i \in \{0,1\} \right)_{i=1..N} \right\}$$

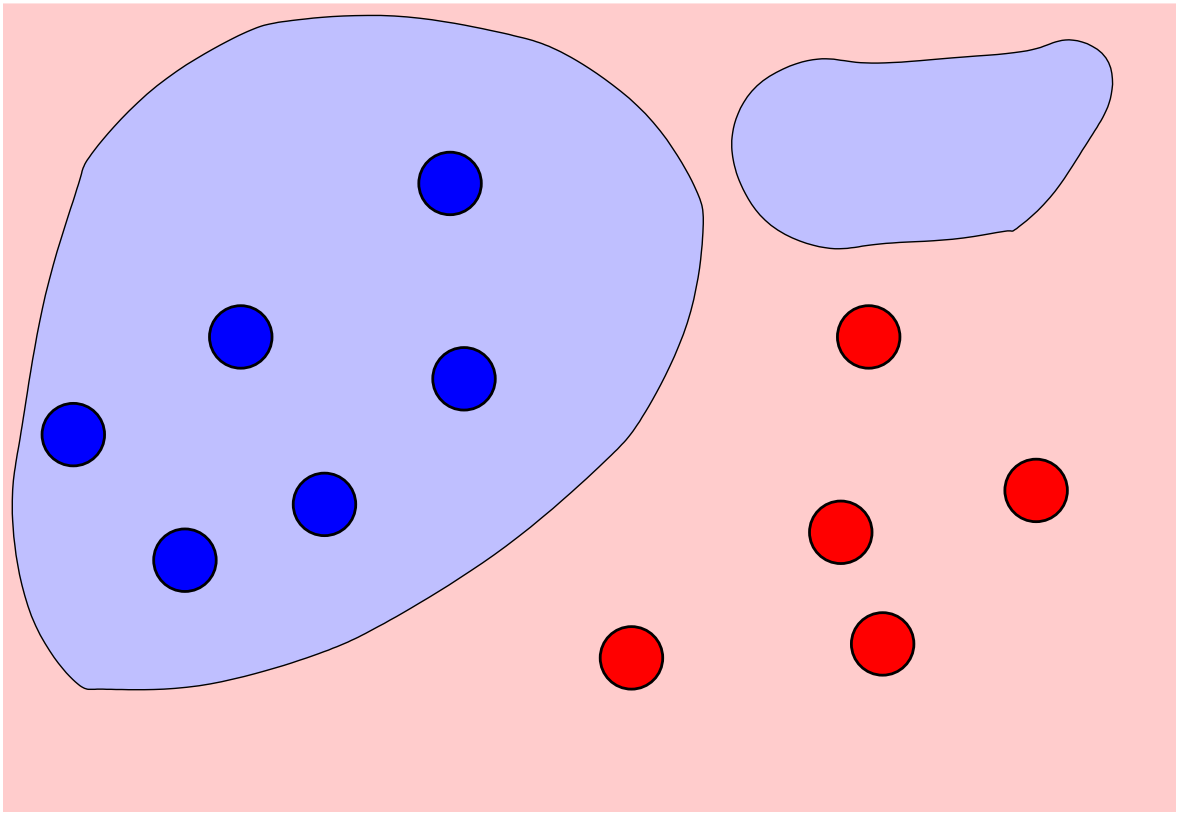
- For illustration purposes **only** we will consider **vectors in the plane**, $d = 2$.
- Points are easier to represent in 2 dimensions than in 20.000...
- The ideas for $d \gg 3$ are **exactly the same**.

Binary Classification Separation Surfaces for Vectors



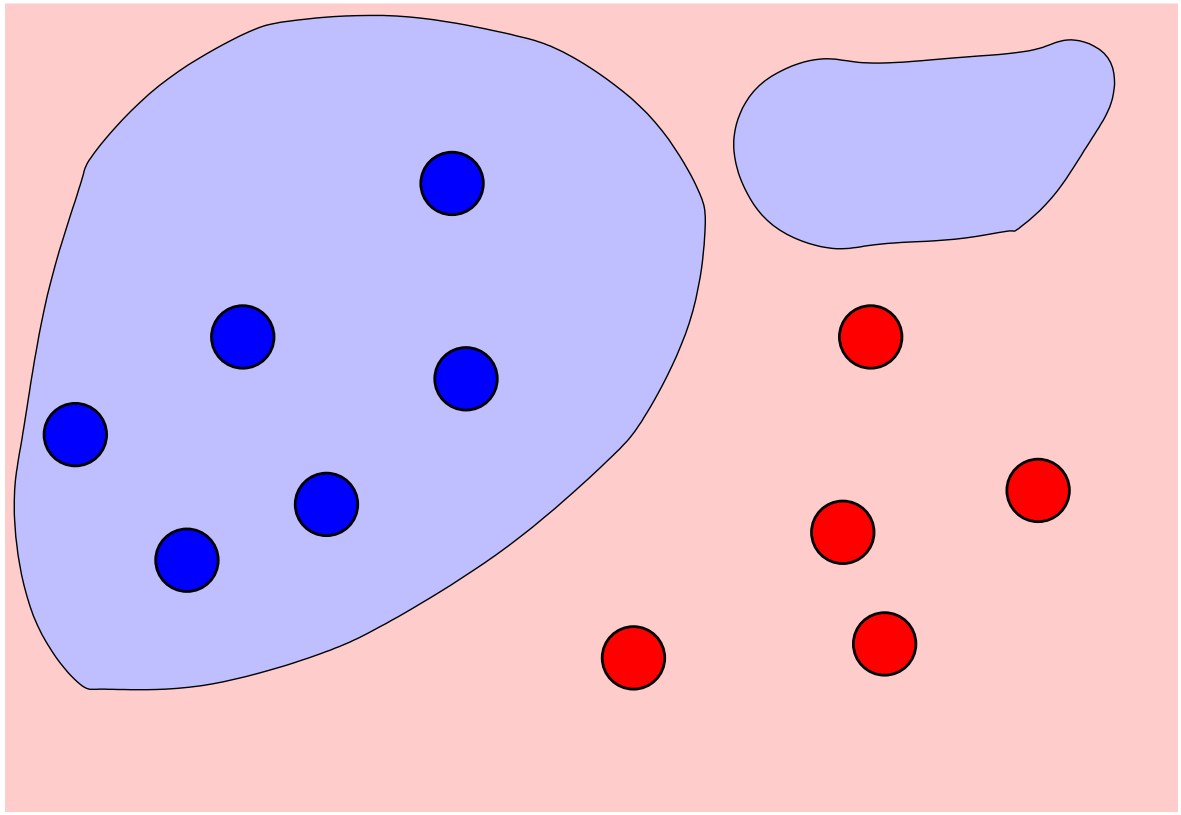
What is a classification rule?

Binary Classification Separation Surfaces for Vectors



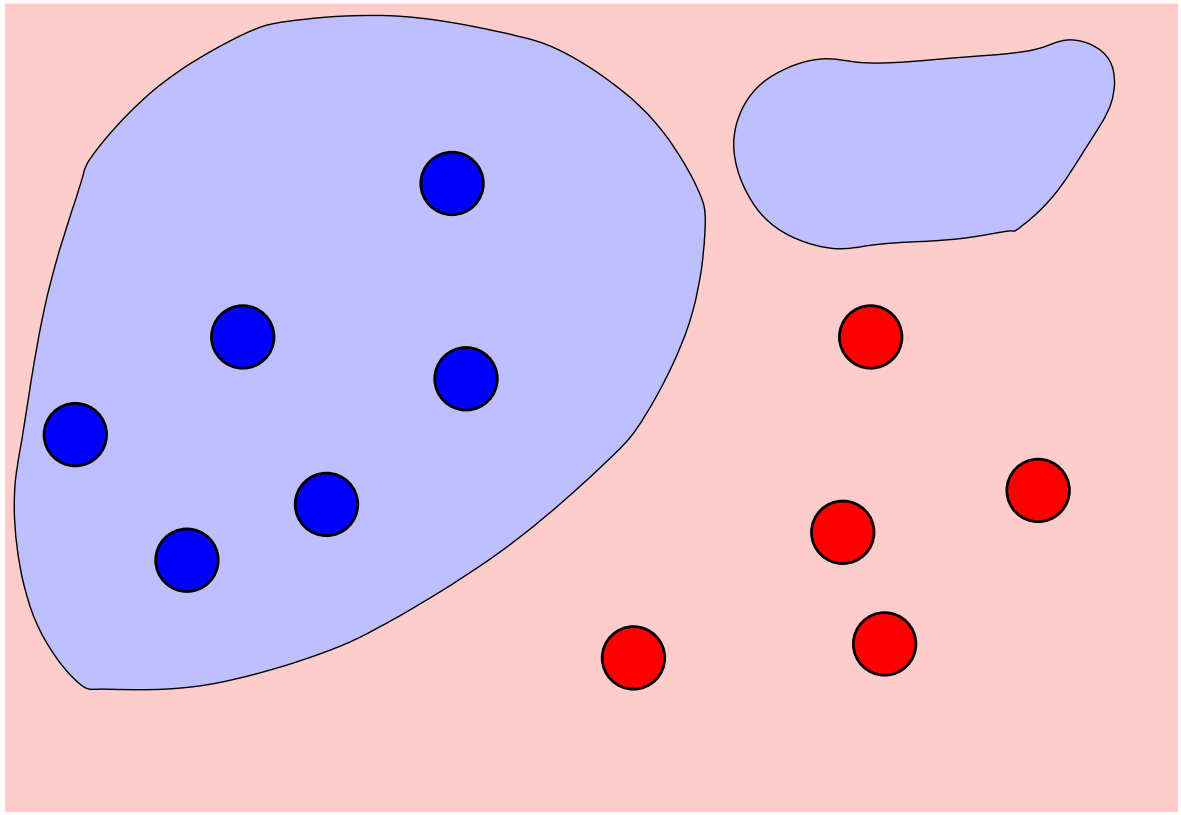
Classification rule = a partition of $\mathbb{R}^d$ into two sets

Binary Classification Separation Surfaces for Vectors



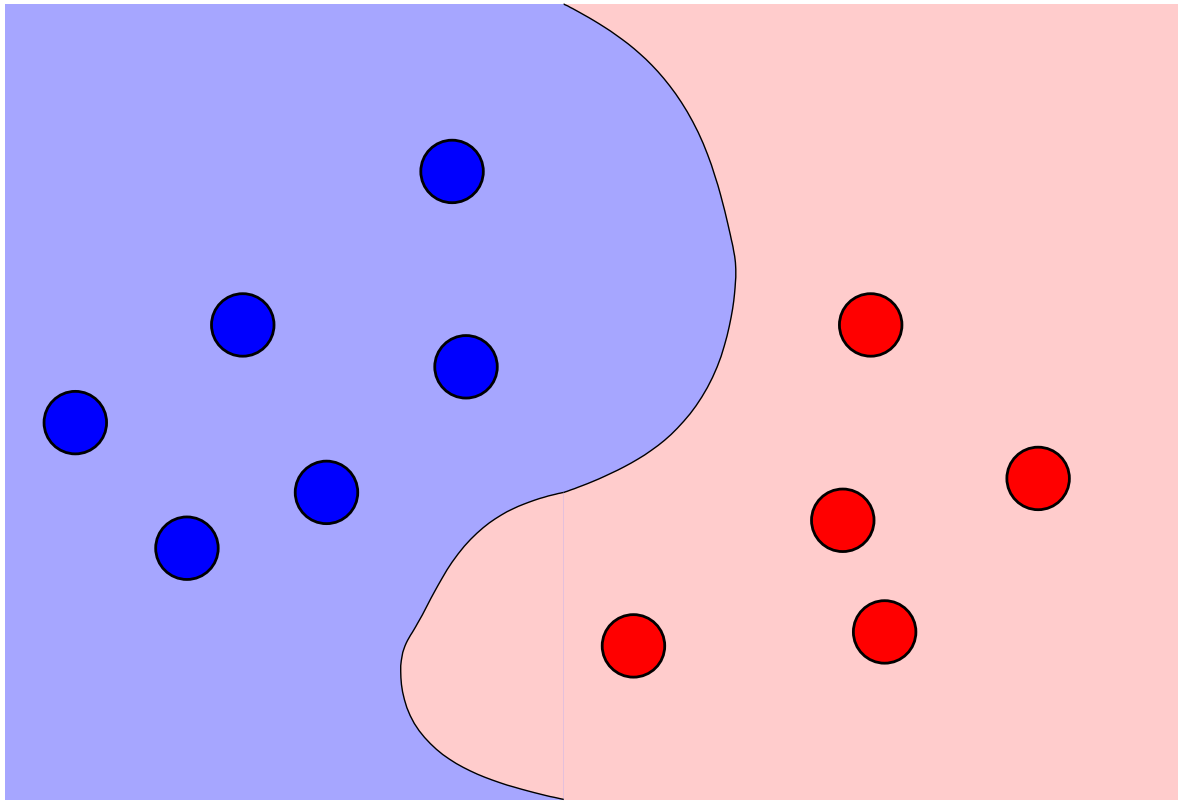
This partition is usually interpreted as the level set of function on $\mathbb{R}^d$

Binary Classification Separation Surfaces for Vectors



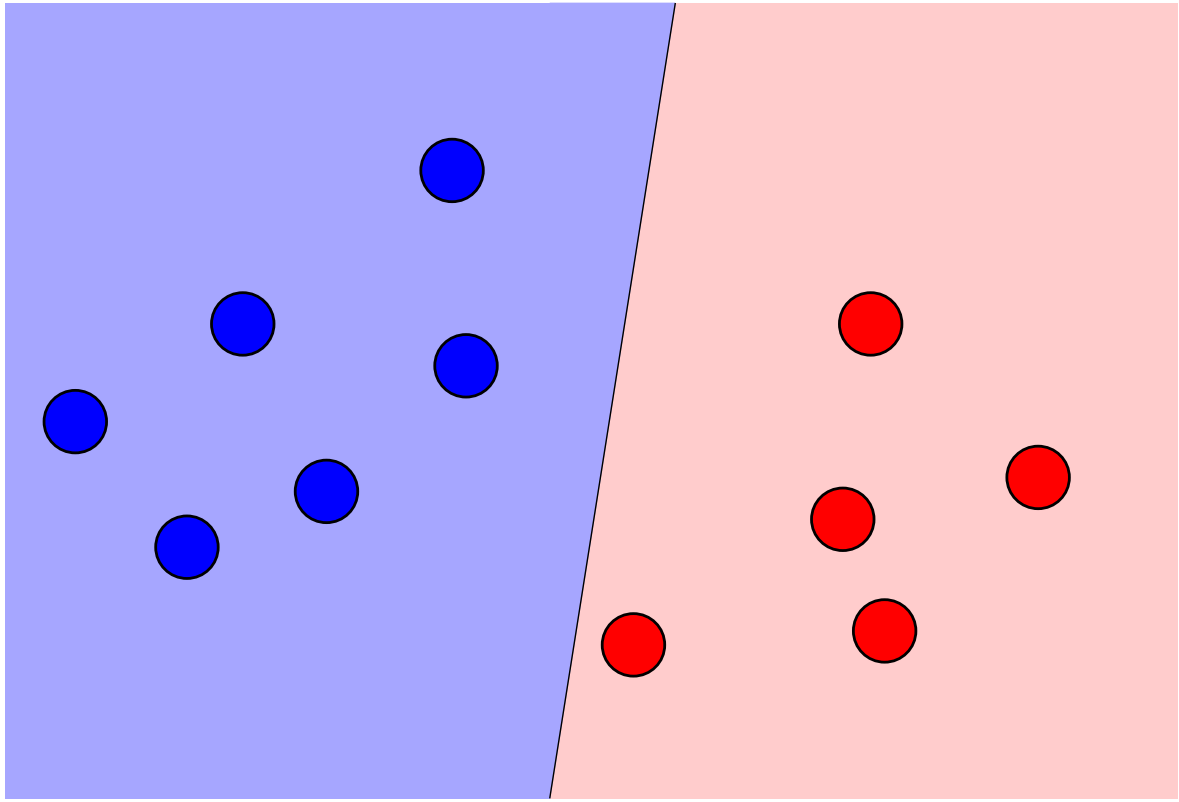
Typically, $\{\mathbf{x} \in \mathbb{R}^d | \boldsymbol{f}(\mathbf{x}) > 0\}$ and $\{\mathbf{x} \in \mathbb{R}^d | \boldsymbol{f}(\mathbf{x}) \leq 0\}$

Classification Separation Surfaces for Vectors



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

Classification Separation Surfaces for Vectors



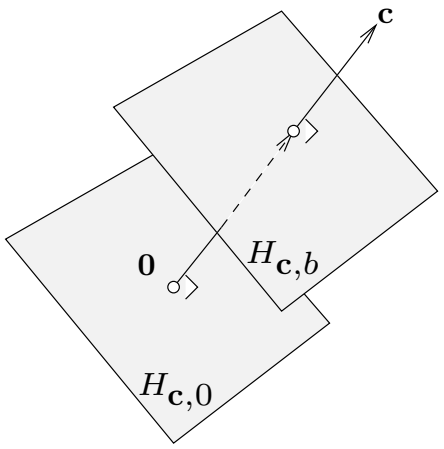
Even more **simple**: using **straight lines** and halfspaces.

Linear Classifiers

- **Straight lines** (hyperplanes when $d > 2$) are **the simplest type** of classifiers.
- A hyperplane $H_{\mathbf{c},b}$ is a set in $\mathbb{R}^d$ defined by
  - a normal vector $\mathbf{c} \in \mathbb{R}^d$
  - a constant $b \in \mathbb{R}$. as

$$H_{\mathbf{c},b} = \{\mathbf{x} \in \mathbb{R}^d \,|\, \mathbf{c}^T\mathbf{x} = b\}$$

- Letting b vary we can "slide" the hyperplane across $\mathbb{R}^d$

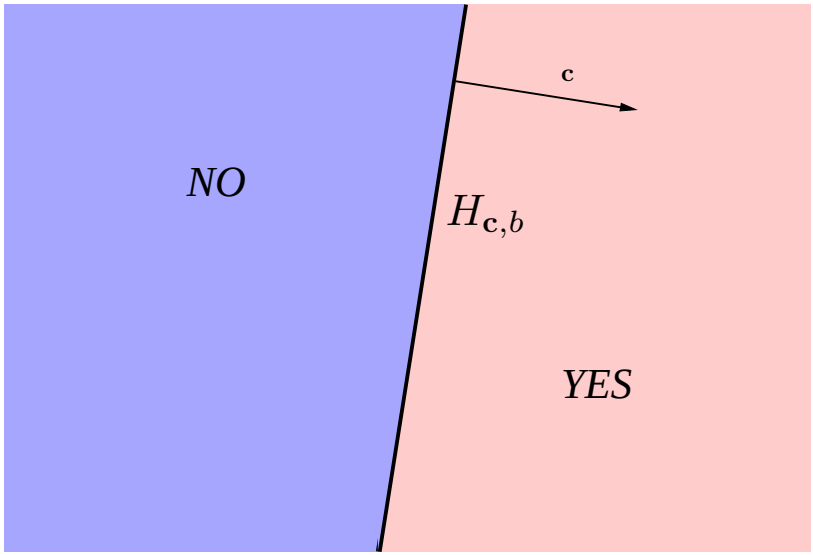


Linear Classifiers

- Exactly like lines in the plane, hypersurfaces **divide** $\mathbb{R}^d$ into **two** halfspaces,

$$\left\{\mathbf{x} \in \mathbb{R}^d \,|\, \mathbf{c}^T\mathbf{x} < b\right\} \cup \left\{\mathbf{x} \in \mathbb{R}^d \,|\, \mathbf{c}^T\mathbf{x} \geq b\right\} = \mathbb{R}^d$$

- Linear classifiers attribute the "yes" and "no" answers given arbitrary $\mathbf{c}$ and b.



- Assuming we only look at halfspaces for the decision surface...
  ...*how to* ***choose*** *the* ***"best"*** $(\mathbf{c}^\star, b^\star)$ *given a* ***training sample****?*

Linear Classifiers

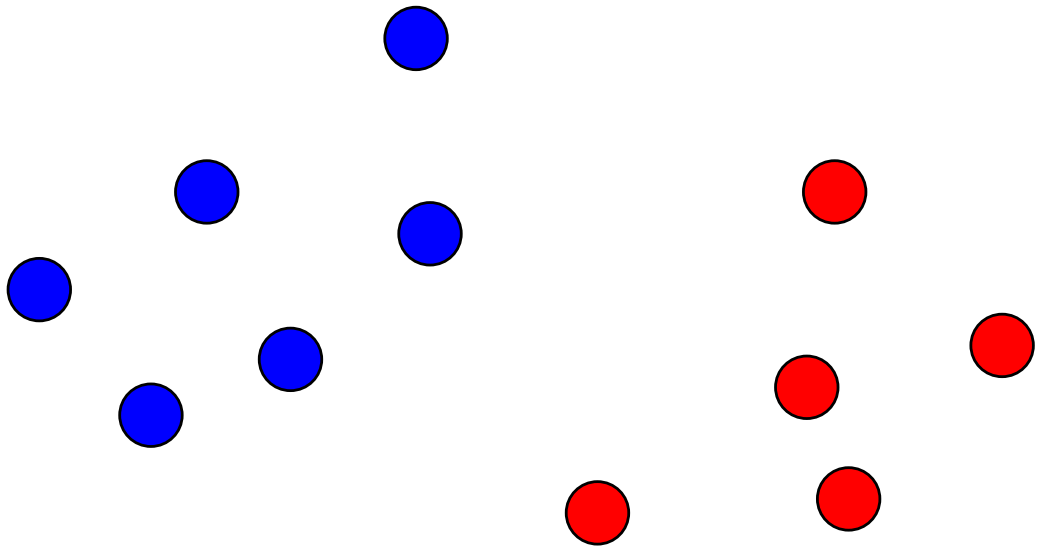
- This specific question,

$$\text{“training set”} \ \left\{ \left(\mathbf{x}_i \in \mathbb{R}^d, \ \mathbf{y}_i \in \{0,1\} \right)_{i=1..N} \right\} \overset{????}{\Longrightarrow} \text{“best”} \, \mathbf{c}^\star, \, b^\star$$

  has different answers. Depends on the meaning of “best” **?**:

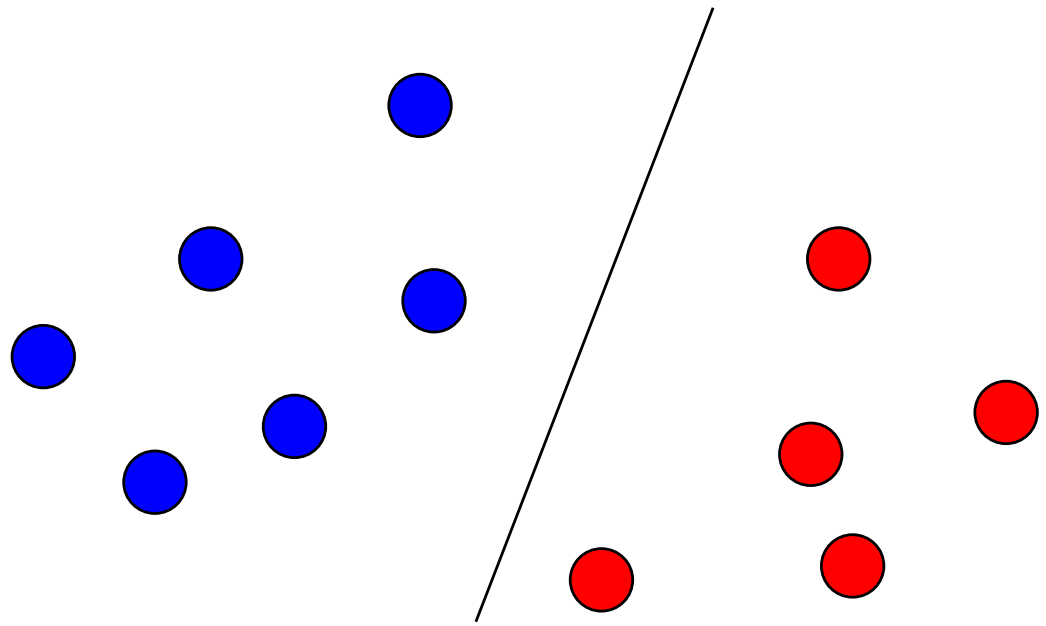
- **Linear Discriminant Analysis** (or Fisher's Linear Discriminant);
- **Logistic regression** maximum likelihood estimation;
- **Perceptron**, a one-layer neural network;
- **Support Vector Machine**, the result of a convex program
- *etc.*

Classification Separation Surfaces for Vectors



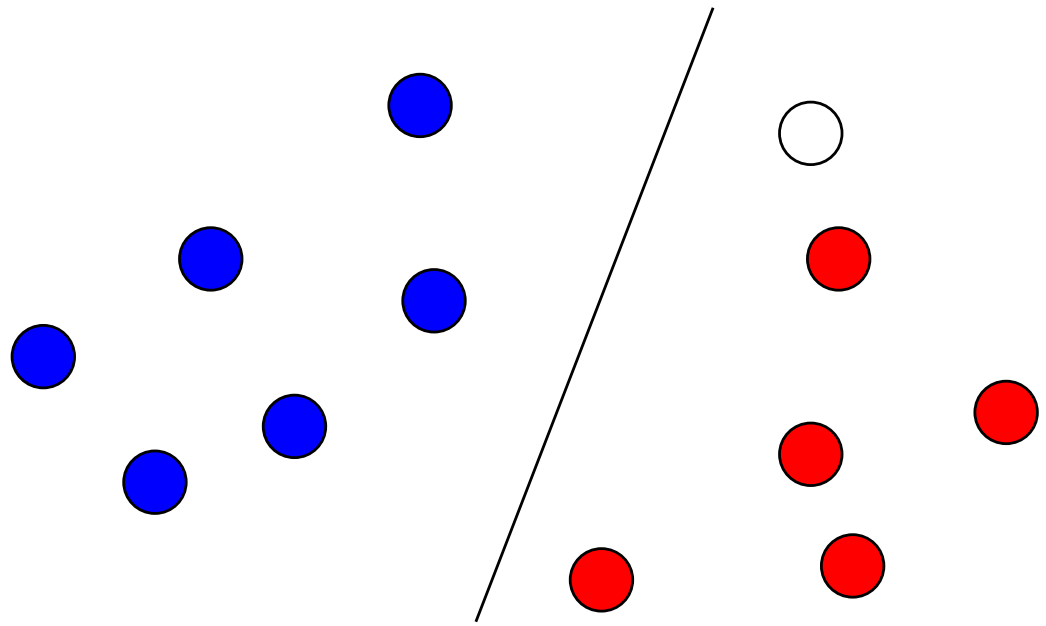
Given two sets of points...

Classification Separation Surfaces for Vectors



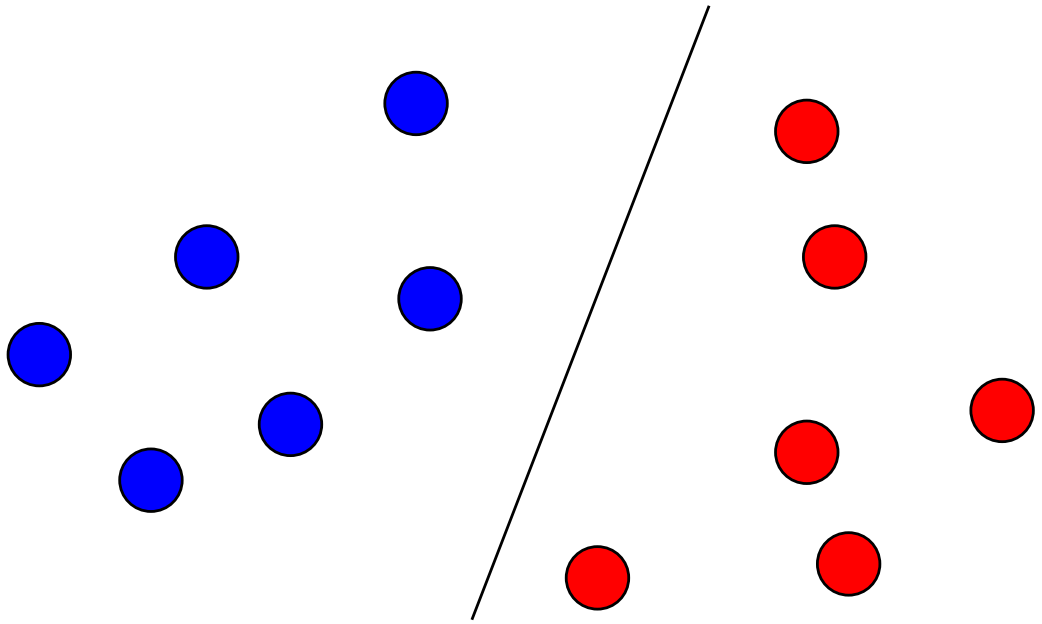
It is sometimes possible to separate them perfectly

Classification Separation Surfaces for Vectors

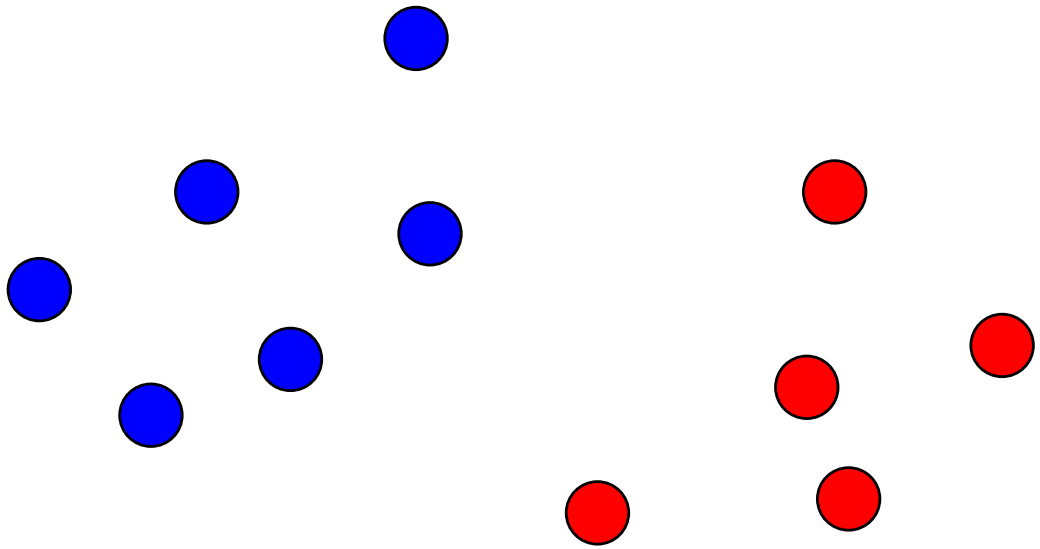


Each choice might look equivalently good on the training set,
but it will have obvious impact on new points

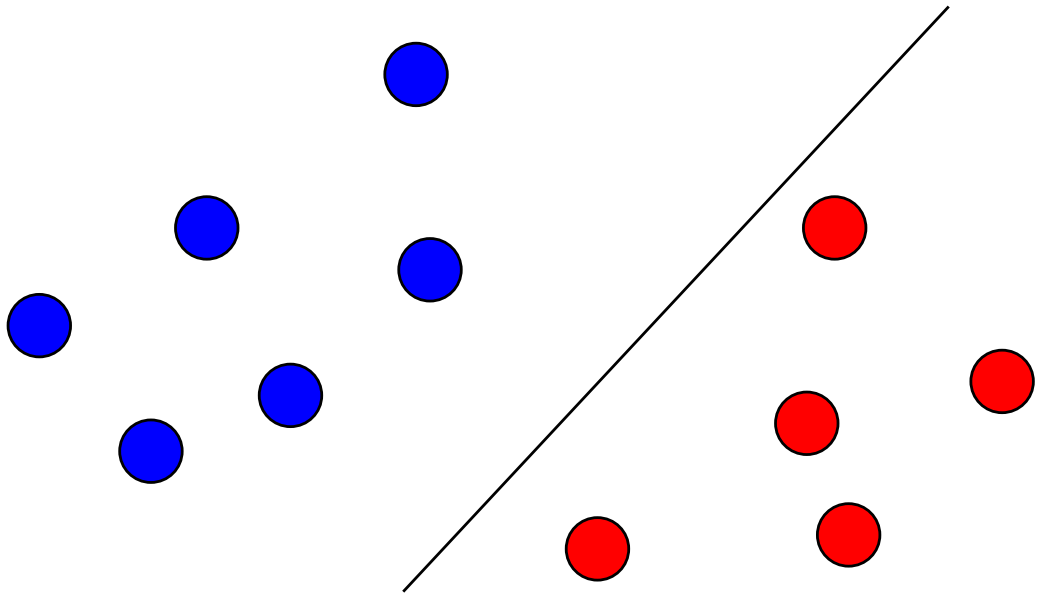
Classification Separation Surfaces for Vectors



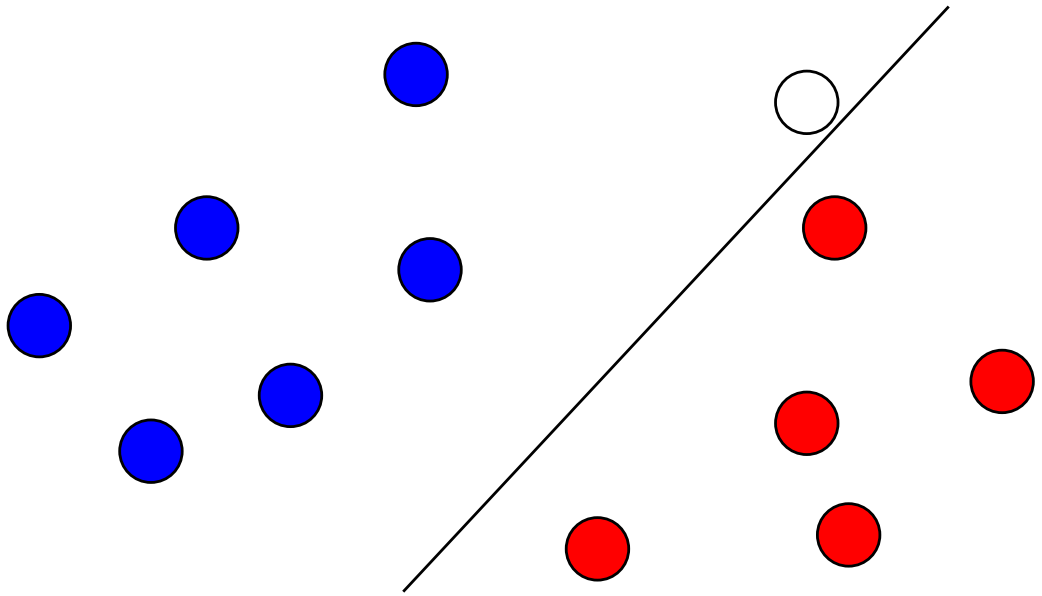
Linear classifier, some degrees of freedom



Linear classifier, some degrees of freedom

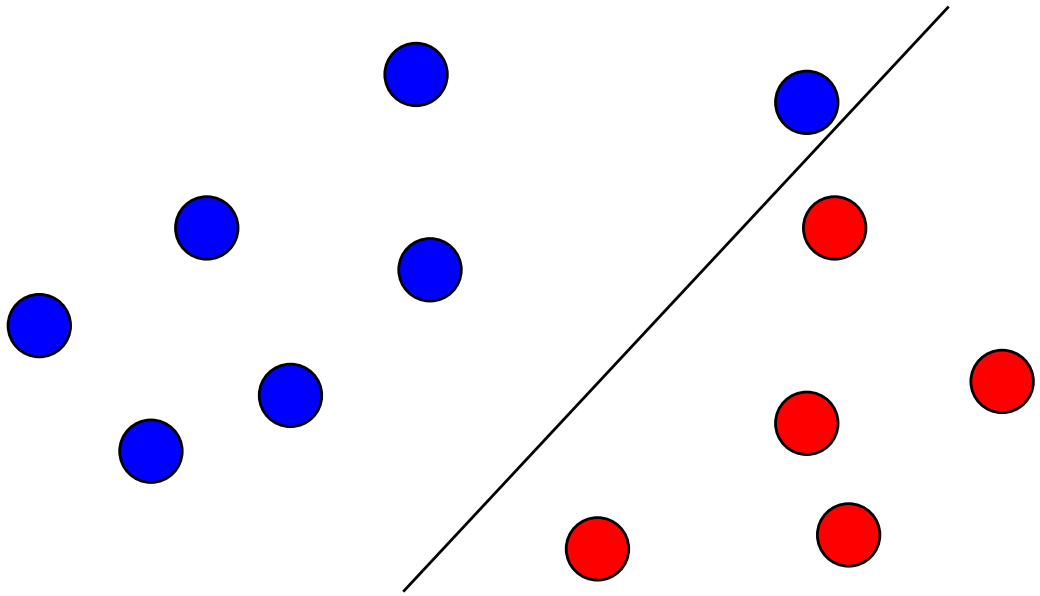


Linear classifier, some degrees of freedom

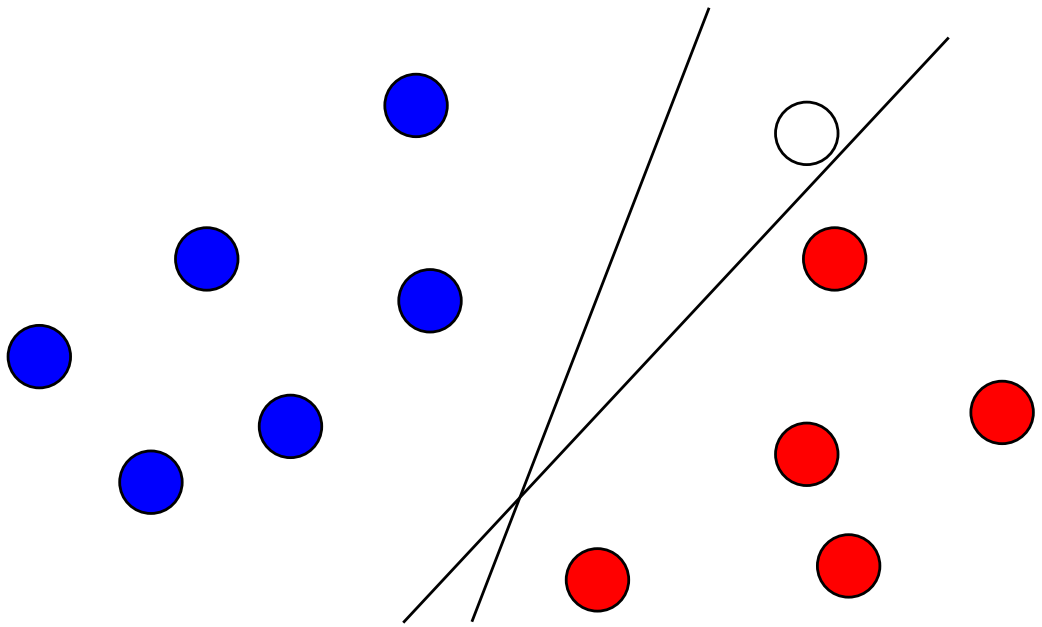


Specially close to the border of the classifier

Linear classifier, some degrees of freedom

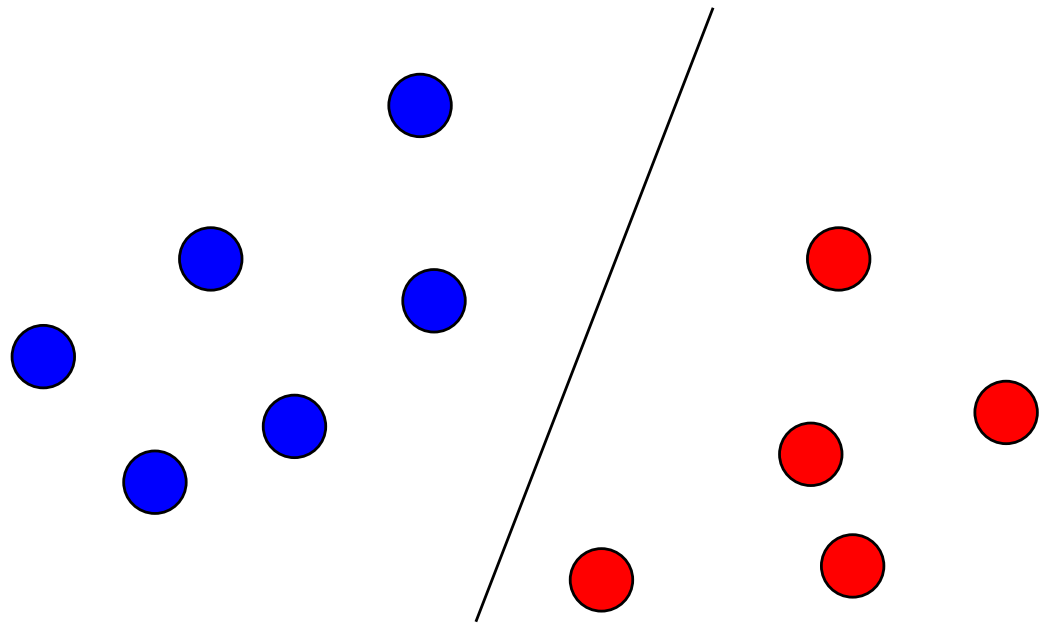


Linear classifier, some degrees of freedom

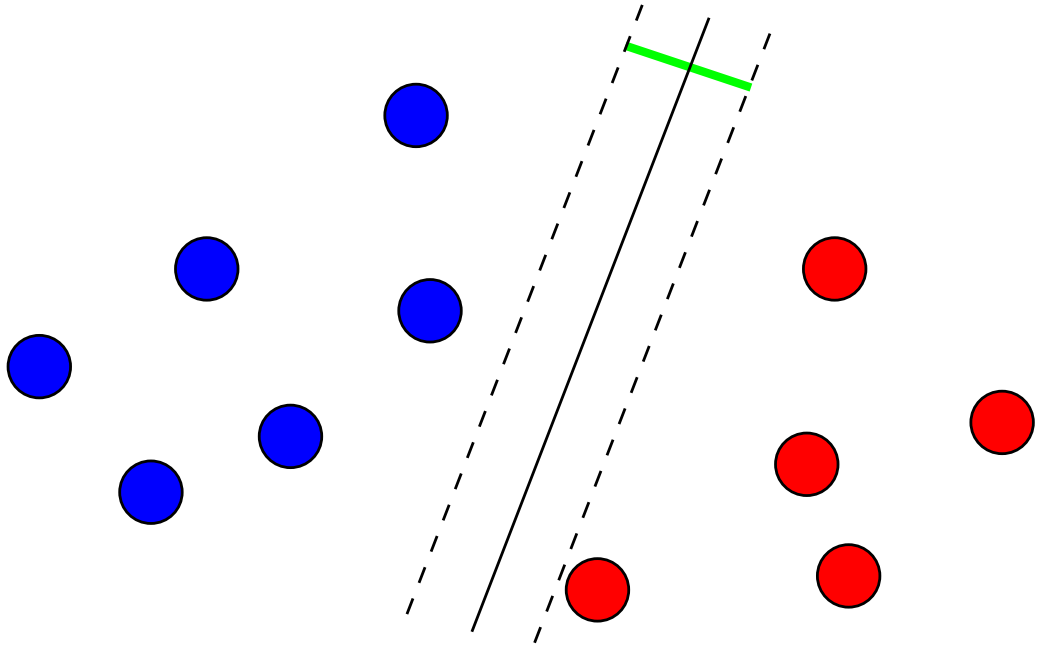


For each different technique, different results, different performance.

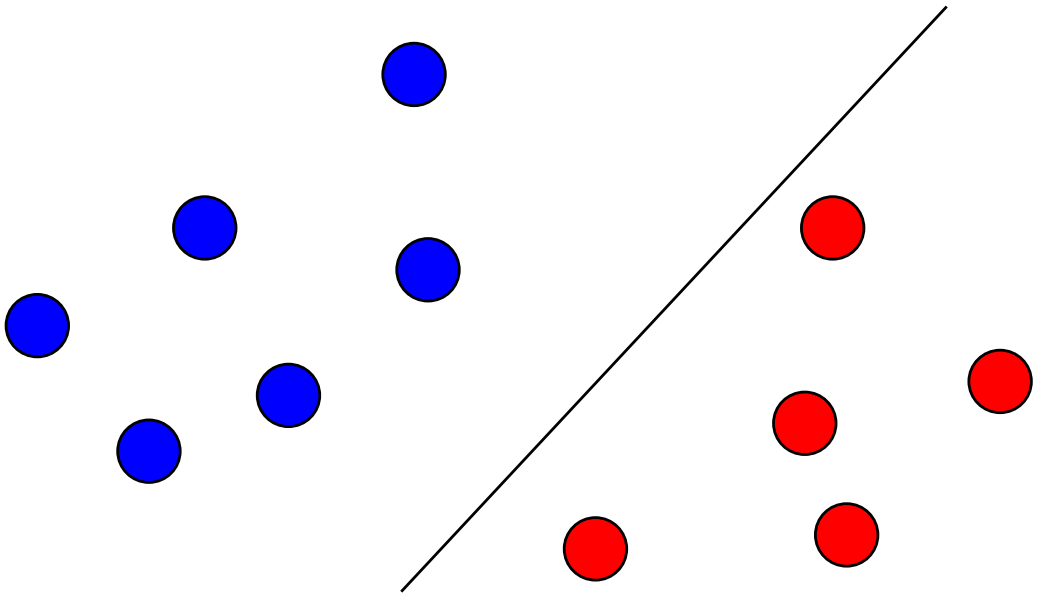
A criterion to select a linear classifier: the margin ?



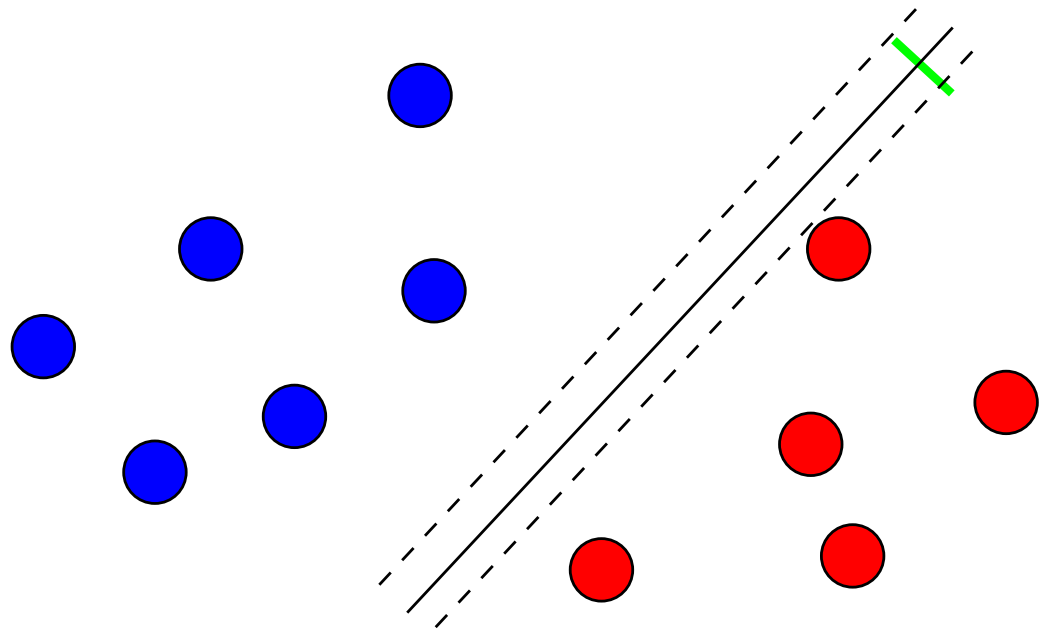
A criterion to select a linear classifier: the margin ?



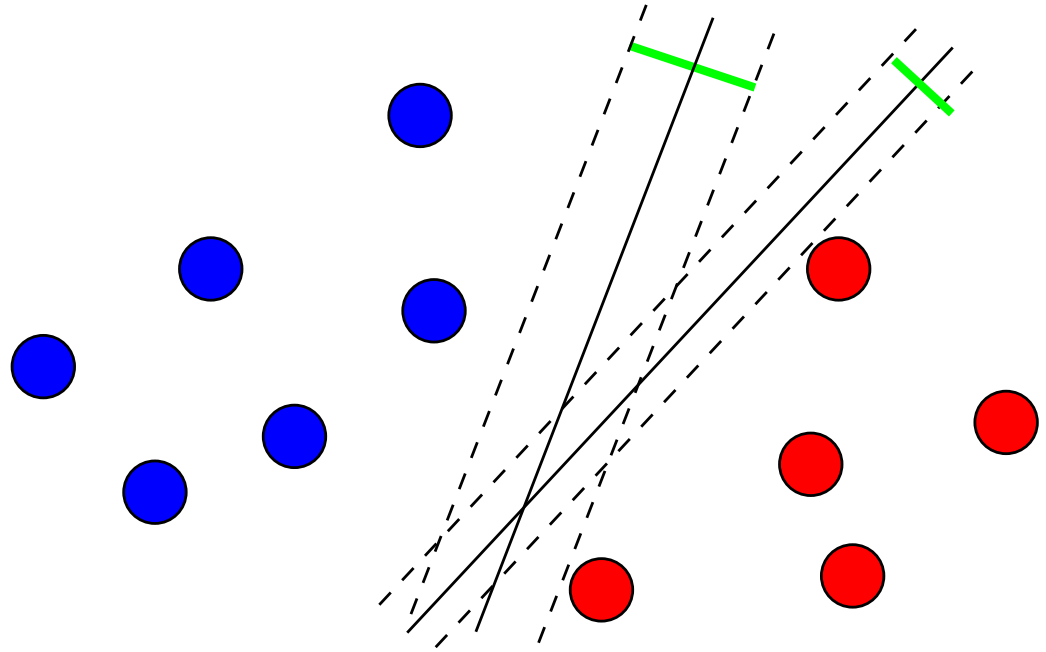
A criterion to select a linear classifier: the margin ?



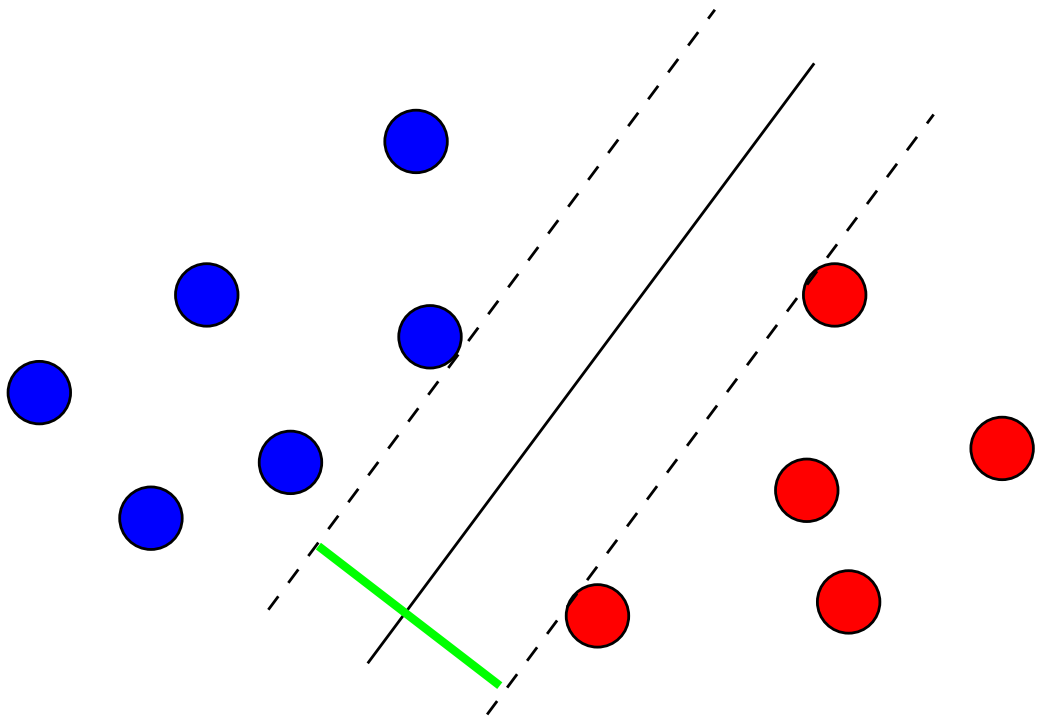
A criterion to select a linear classifier: the margin ?



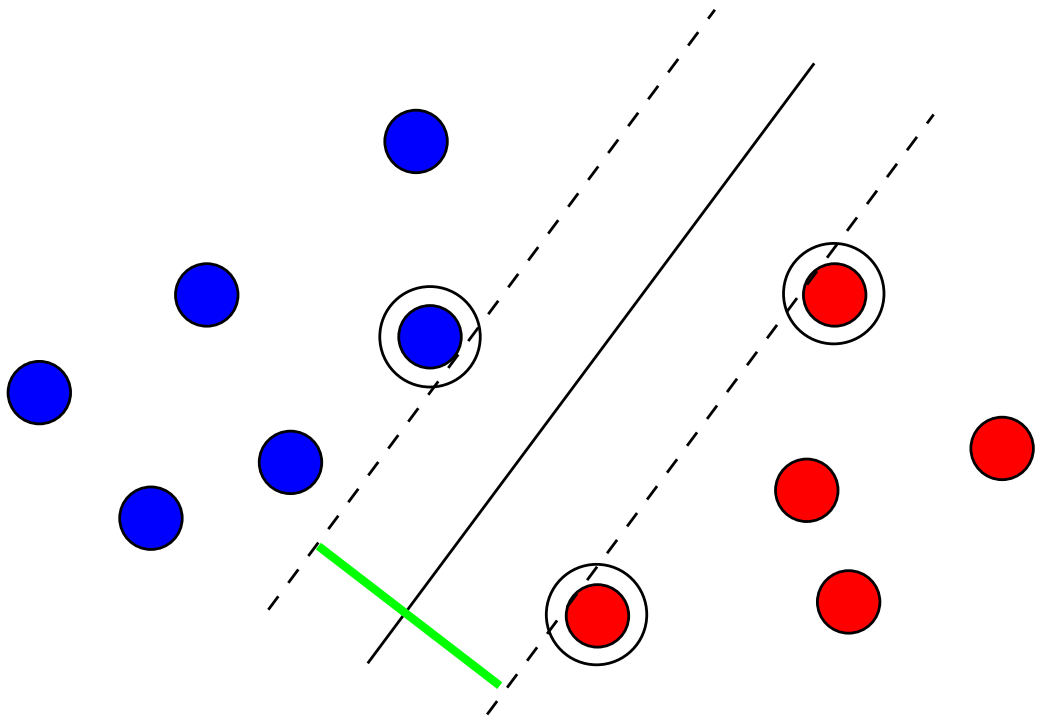
A criterion to select a linear classifier: the margin ?



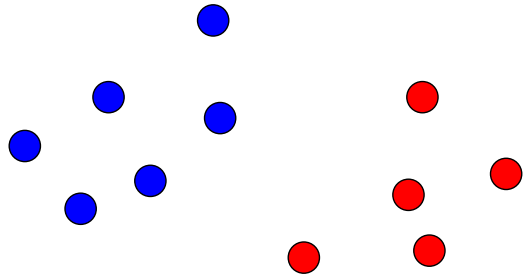
Largest Margin Linear Classifier ?



Support Vectors with Large Margin



In equations



- The **training set** is a finite set of n data/class pairs:

$$\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\},$$

  where $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{y}_i \in \{-1, 1\}$.

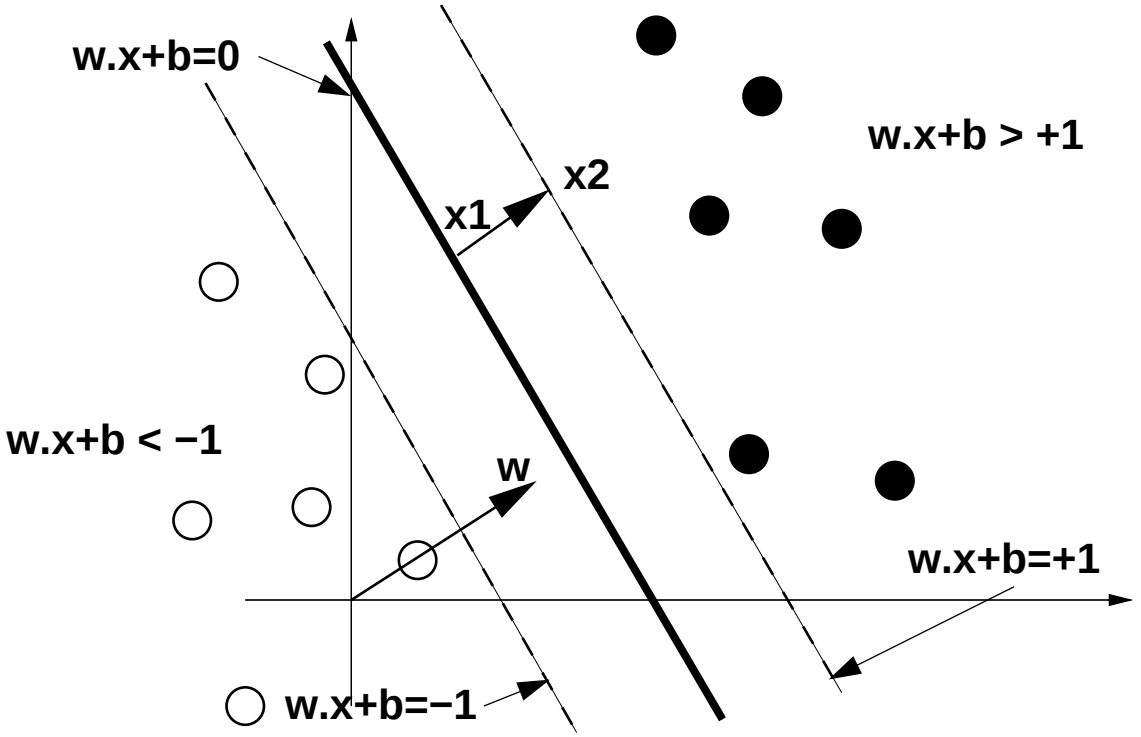
- We assume (for the moment) that the data are **linearly separable**, i.e., that there exists $(\mathbf{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ such that:

$$\begin{cases} \mathbf{w}^T \mathbf{x}_i + b > 0 & \text{if } \mathbf{y}_i = 1\,, \\ \mathbf{w}^T \mathbf{x}_i + b < 0 & \text{if } \mathbf{y}_i = -1\,. \end{cases}$$

How to find the largest separating hyperplane?

For the linear classifier $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ consider the *interstice* defined by the hyperplanes

- $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = +1$
- $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = -1$



The margin is $2/||\mathbf{w}||$

- Indeed, the points $\mathbf{x}_1$ and $\mathbf{x}_2$ satisfy:

$$\begin{cases} \mathbf{w}^T\mathbf{x}_1 + b = 0, \\ \mathbf{w}^T\mathbf{x}_2 + b = 1. \end{cases}$$

- By subtracting we get $\mathbf{w}^T(\mathbf{x}_2 - \mathbf{x}_1) = 1$, and therefore:

$$\gamma = 2||\mathbf{x}_2 - \mathbf{x}_1|| = \frac{2}{||\mathbf{w}||}.$$

  where γ is the margin.

All training points should be on the appropriate side

- For positive examples ($y_i = 1$) this means:

$$\mathbf{w}^T \mathbf{x}_i + b \geq 1$$

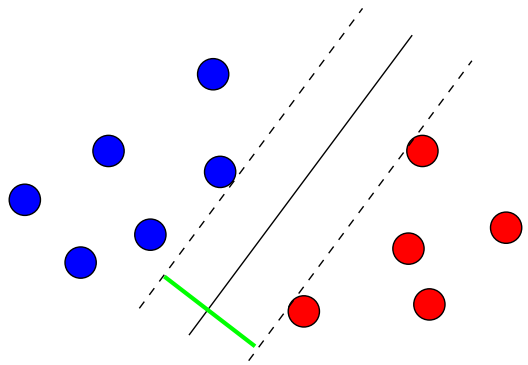
- For negative examples ($y_i = -1$) this means:

$$\mathbf{w}^T \mathbf{x}_i + b \leq -1$$

- in both cases:

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

Finding the optimal hyperplane



- Finding the optimal hyperplane is equivalent to finding $(\mathbf{w}, b)$ which minimize:

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

under the constraints:

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

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

Lagrangian

- In order to minimize:

$$\frac{1}{2}||\mathbf{w}||^2$$

  under the constraints:

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

- introduce **one dual variable α_i for each constraint**,
- one constraint for **each training point**.
- the **Lagrangian** is, for $\alpha \succeq 0$ (that is for each $\alpha_i \geq 0$)

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i \left(y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) - 1\right).$$

The Lagrange dual function

$$g(\alpha) = \inf_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) - 1 \right) \right\}$$

the saddle point conditions give us that at the minimum in $\mathbf{w}$ and b

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{y}_i \mathbf{x}_i, \quad (\text{ derivating w.r.t } \mathbf{w}) \quad (*)$$

$$0 = \sum_{i=1}^{n} \alpha_i \mathbf{y}_i, \quad (\text{derivating w.r.t } b) \qquad (**)$$

substituting $(*)$ in g, and using $(**)$ as a constraint, get the dual function $g(\alpha)$.

- To solve the dual problem, **maximize** g w.r.t. α.
- **Strong duality holds** : primal and dual problems have the **same optimum**.
- KKT gives us $\alpha_i(\mathbf{y}_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) - 1) = 0$,
  ...*hence*, either $\boldsymbol{\alpha_i = 0}$ or $\mathbf{y_i} \left(\mathbf{w}^T \mathbf{x_i} + b \right) = 1$.
- $\alpha_i \neq 0$ **only** for points on the support hyperplanes $\{(\mathbf{x}, \mathbf{y}) | \, \mathbf{y}_i(\mathbf{w}^T \mathbf{x}_i + b) = 1\}$.

Dual optimum

The dual problem is thus

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

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

Recovering the optimal hyperplane

- With α^*, we recover $(\mathbf{w}^T, b^*)$ corresponding to the **optimal hyperplane**.
- $\mathbf{w}^T$ is given by $\mathbf{w}^T = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i^T$,
- b^* is given by the conditions on the support vectors $\alpha_i > 0$, $\mathbf{y}_i(\mathbf{w}^T \mathbf{x}_i + b) = 1$,

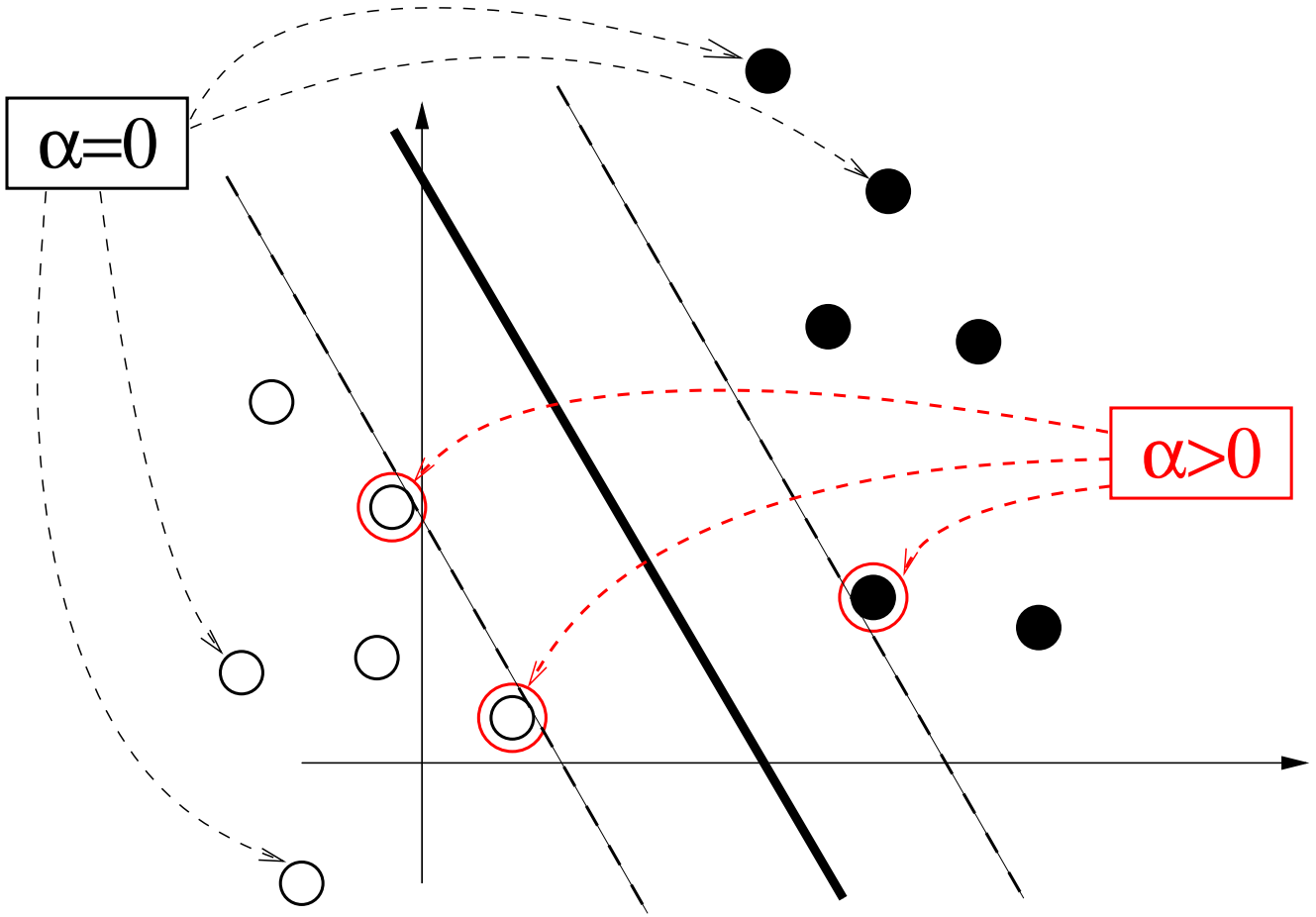
$$b^* = -\frac{1}{2}\left(\min_{\mathbf{y}_i=1, \alpha_i>0} (\mathbf{w}^T \mathbf{x}_i) + \max_{\mathbf{y}_i=-1, \alpha_i>0} (\mathbf{w}^T \mathbf{x}_i) \right)$$

- the **decision function** is therefore:

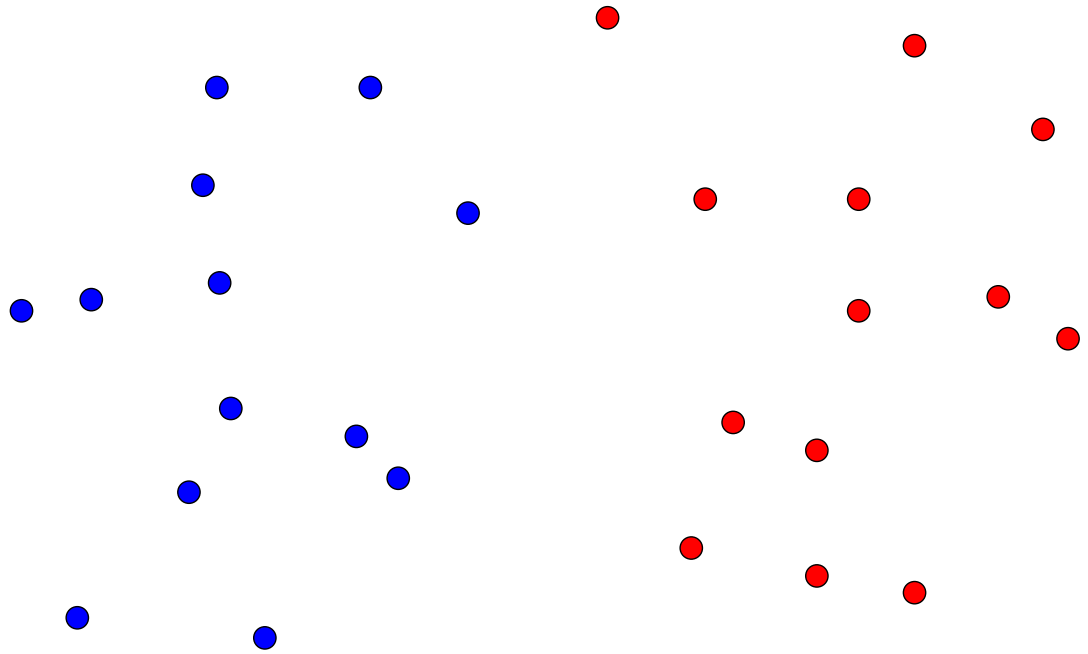
$$\begin{aligned} f^*(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + b^* \\ &= \left(\sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i^T \right) \mathbf{x} + b^*. \end{aligned}$$

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

Interpretation: support vectors

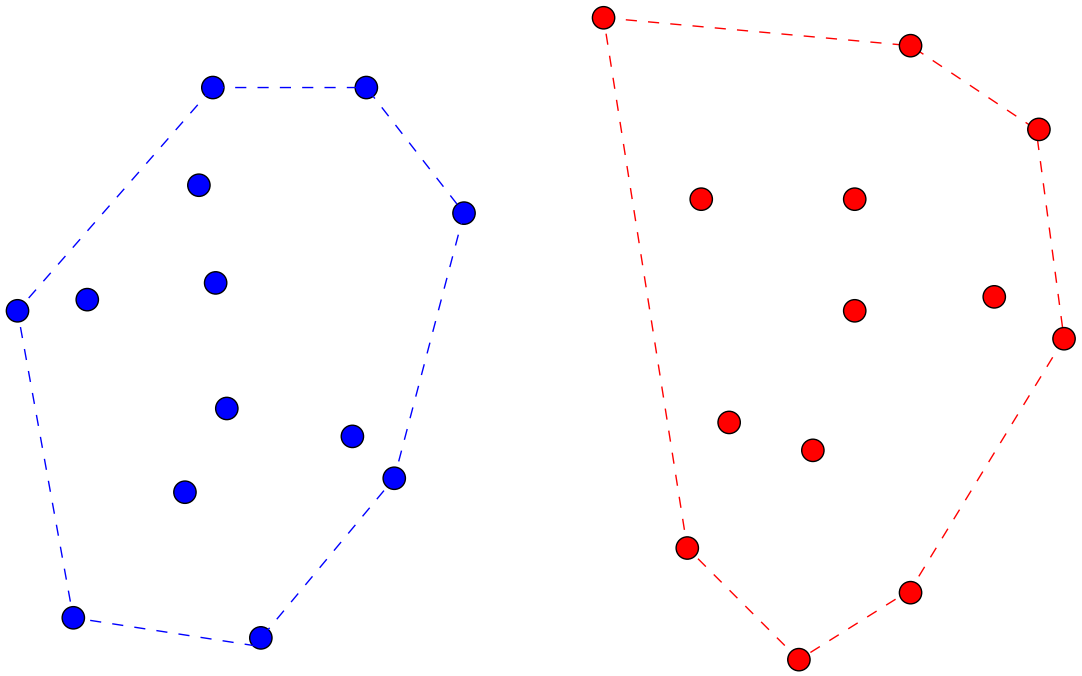


Another interpretation: Convex Hulls



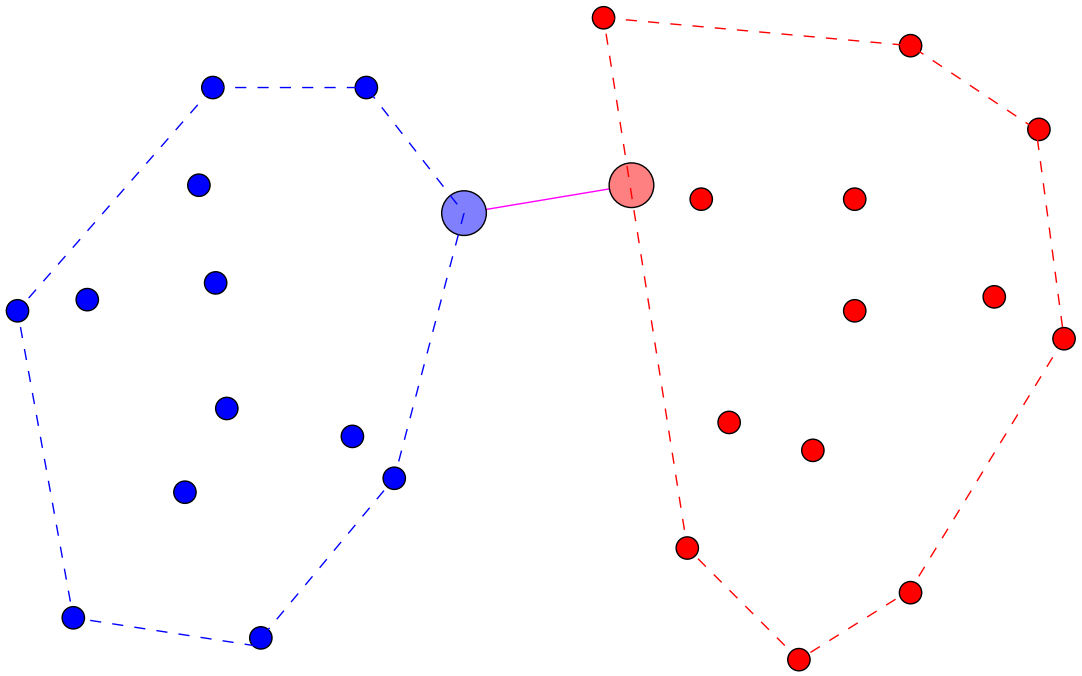
go back to 2 sets of points that are linearly separable

Another interpretation: Convex Hulls



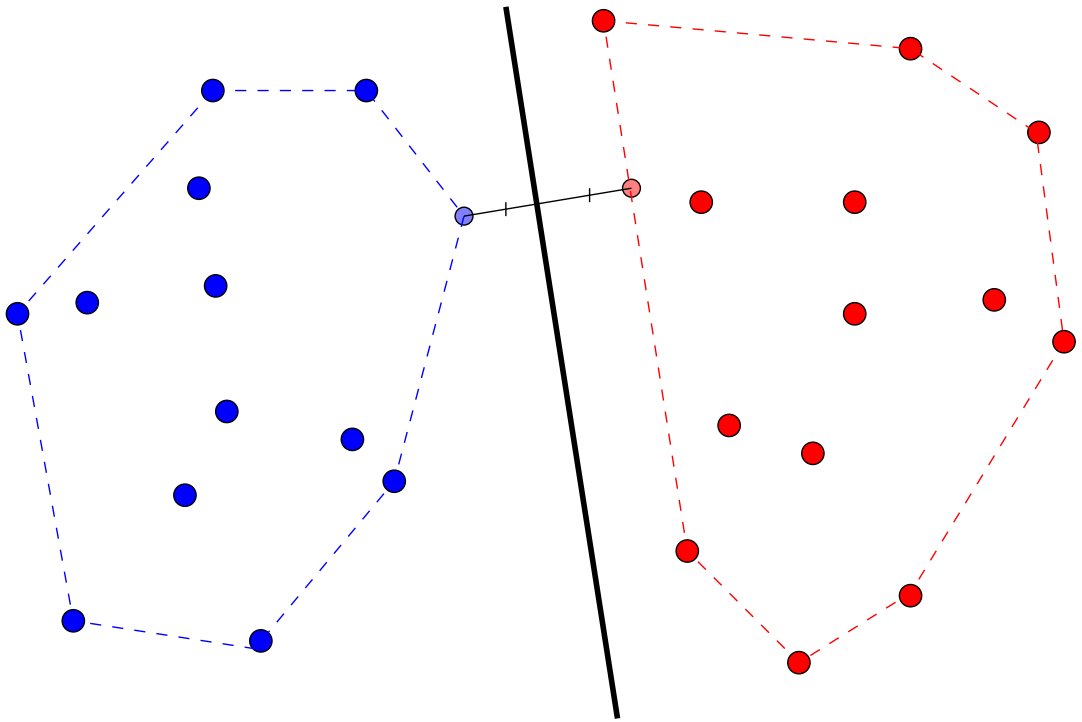
Linearly separable = convex hulls do not intersect

Another interpretation: Convex Hulls



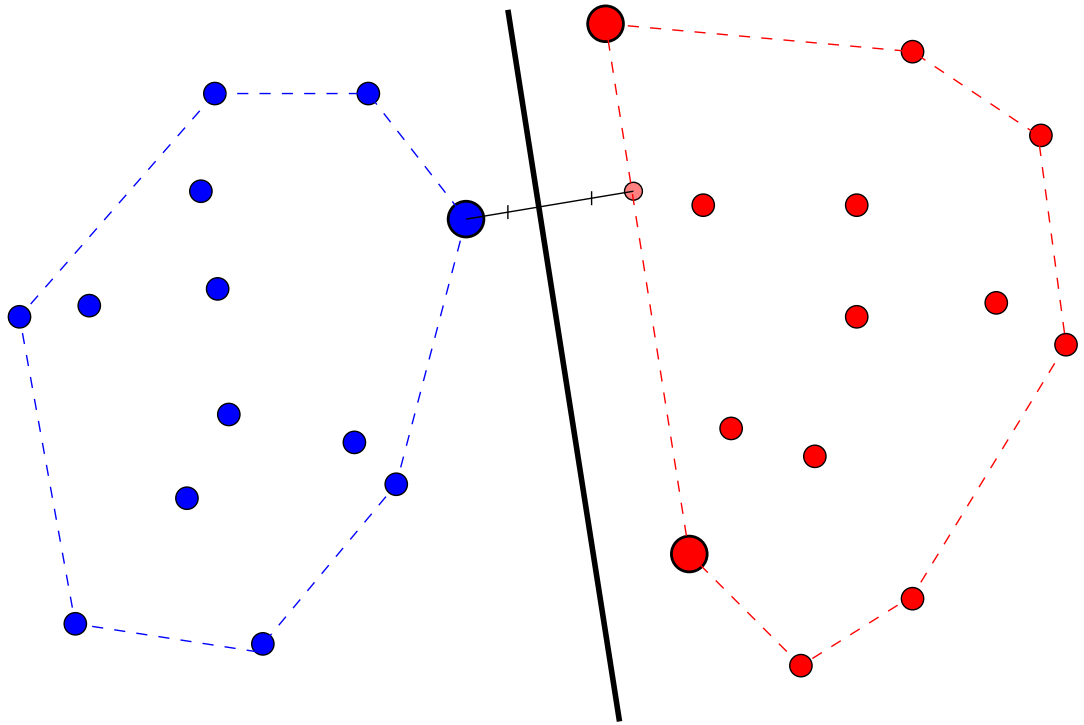
Find two closest points, one in each convex hull

Another interpretation: Convex Hulls



The SVM = bisection of that segment

Another interpretation: Convex Hulls

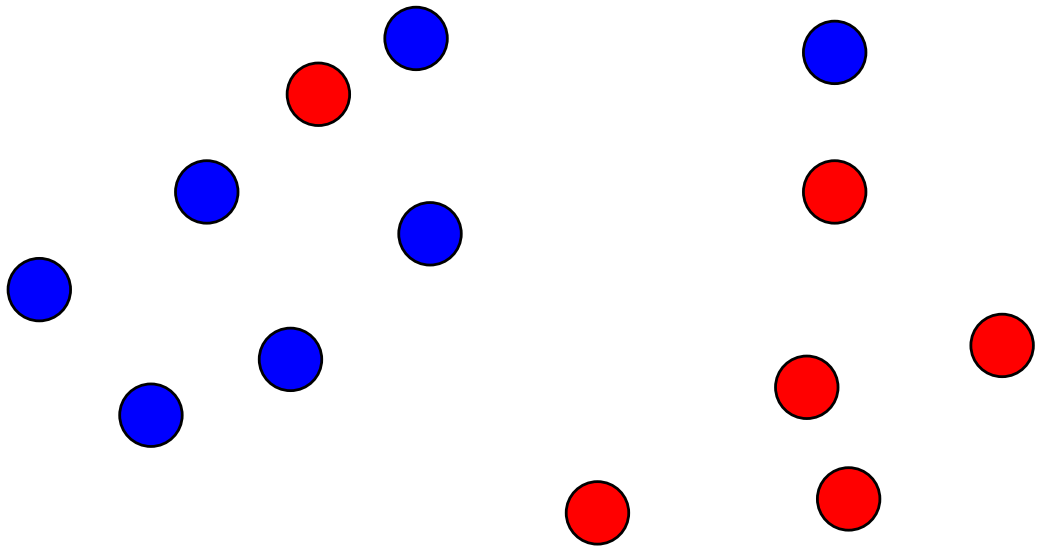


support vectors = extreme points of the faces on which the two points lie

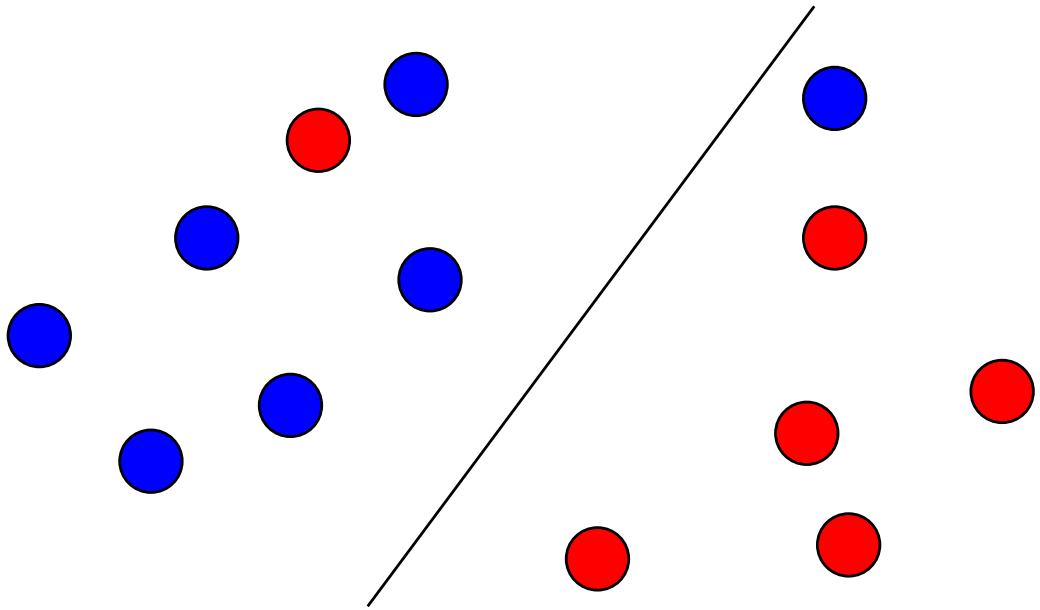
The non-linearly separable case

(when convex hulls intersect)

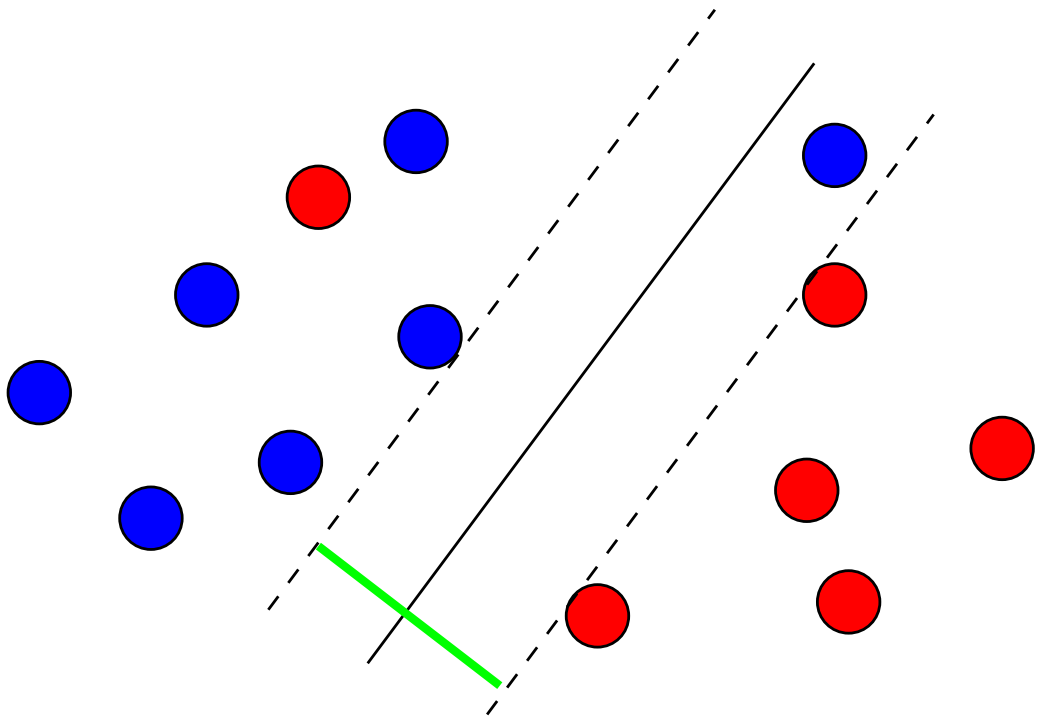
What happens when the data is not linearly separable?



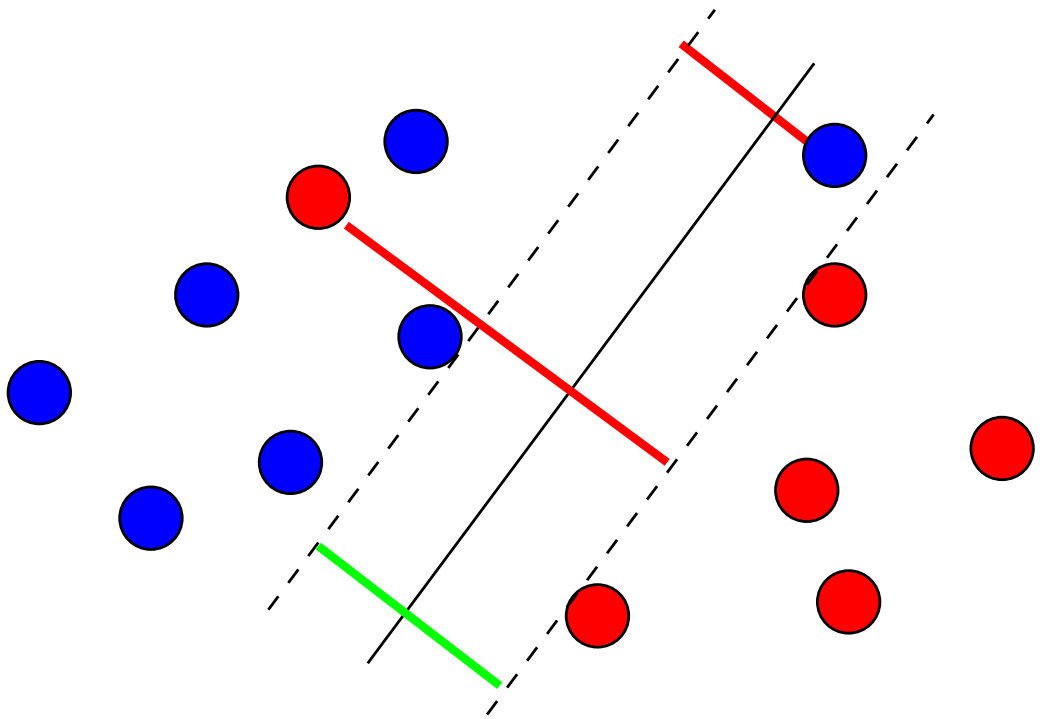
What happens when the data is not linearly separable?



What happens when the data is not linearly separable?



What happens when the data is not linearly separable?

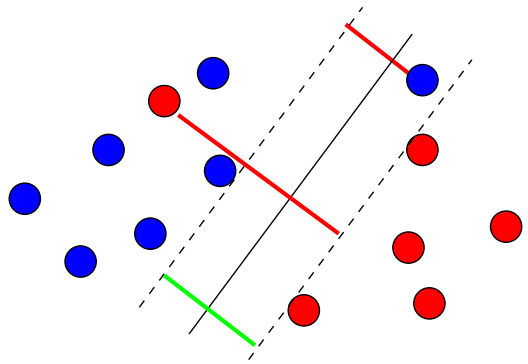


Soft-margin SVM ?

- 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



Soft-margin SVM formulation ?

- The **margin** of a labeled point $(\mathbf{x}, \mathbf{y})$ is

$$\text{margin}(\mathbf{x}, \mathbf{y}) = \mathbf{y}\left(\mathbf{w}^T\mathbf{x} + b\right)$$

- The **error** is
  - 0 if $\text{margin}(\mathbf{x}, \mathbf{y}) > 1$,
  - $1 - \text{margin}(\mathbf{x}, \mathbf{y})$ otherwise.

- The soft margin SVM solves:

$$\min_{\mathbf{w},b}\{\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max\{0, 1 - \mathbf{y}_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\}$$

- $c(u, y) = \max\{0, 1 - yu\}$ is known as the **hinge loss**.

- $c(\mathbf{w}^T\mathbf{x}_i + b, \mathbf{y}_i)$ associates a mistake cost to the decision $\mathbf{w}, b$ for example $\mathbf{x}_i$.

Dual formulation of soft-margin SVM

- The soft margin SVM program

$$\min_{\mathbf{w},b}\{\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max\{0, 1 - \mathbf{y}_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\}$$

  can be rewritten as

$$\begin{array}{ll}\text{minimize} & \|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{such that} & \mathbf{y}_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i\end{array}$$

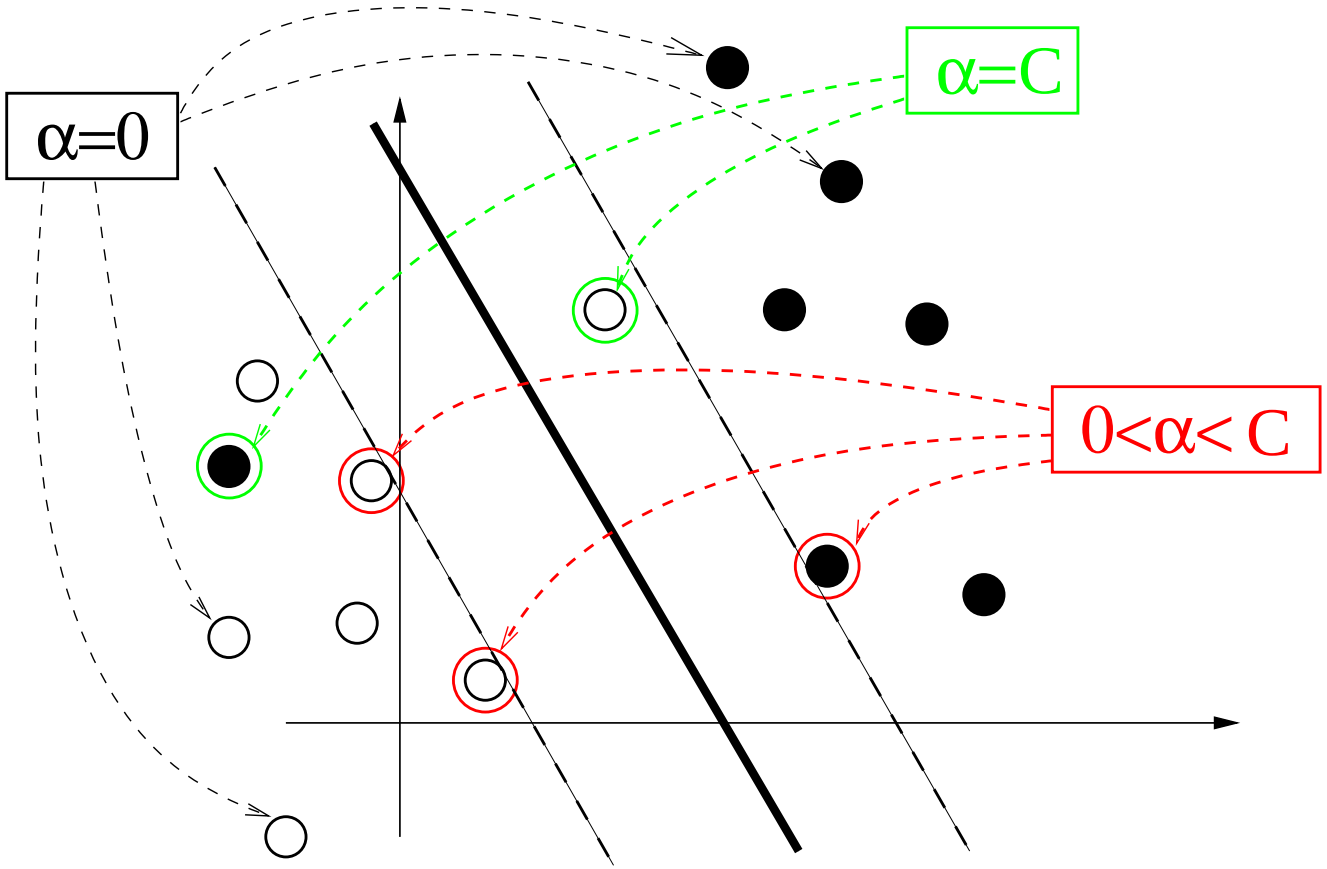
- In that case the dual function

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

  which is finite under the constraints:

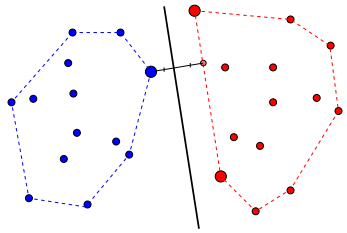
$$\begin{cases}0 \leq \alpha_i \leq \boldsymbol{C}, & \text{for } i = 1, \ldots, n \\ \sum_{i=1}^{n}\alpha_i\mathbf{y}_i = 0.\end{cases}$$

Interpretation: bounded and unbounded support vectors

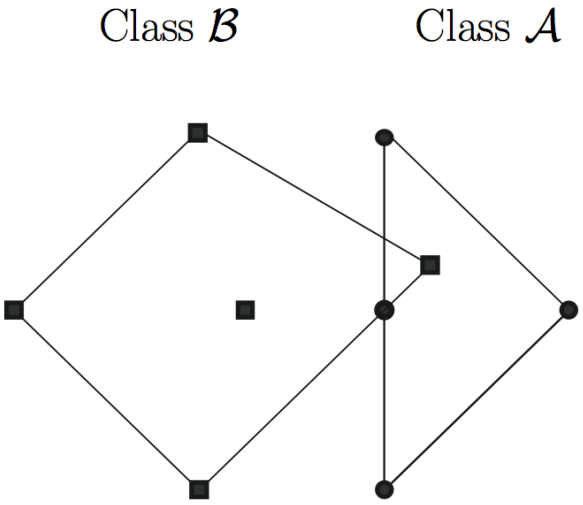


What about the convex hull analogy?

- Remember the separable case



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



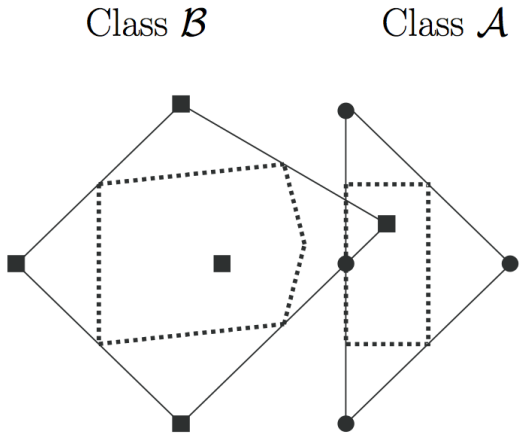
What about the convex hull analogy?

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.

Kernels

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

- **Reminder**: the dual problem is to maximize

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

under the constraints:

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

- The **decision function** becomes:

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

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 $\mathbf{y}$:

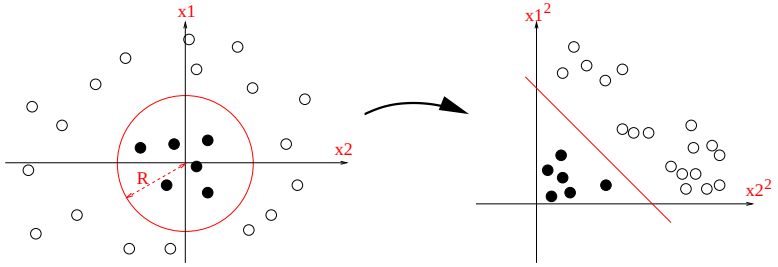
$$\begin{array}{ll} \text{maximize} & g(\alpha) = \alpha^T \mathbf{1} - \frac{1}{2}\alpha^T(\boldsymbol{K} \odot \mathbf{y}\mathbf{y}^T)\alpha \\ \text{such that} & 0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n \\ & \sum_{i=1}^n \alpha_i \mathbf{y}_i = 0. \end{array}$$

- $\boldsymbol{K}$'s **positive definite** $\Leftrightarrow$ **problem has an unique optimum**
- the decision function is $f(\cdot) = \sum_{i=1}^n \alpha_i \boldsymbol{k}(\mathbf{x}_i, \cdot) + b$.

Kernel example: polynomial kernel

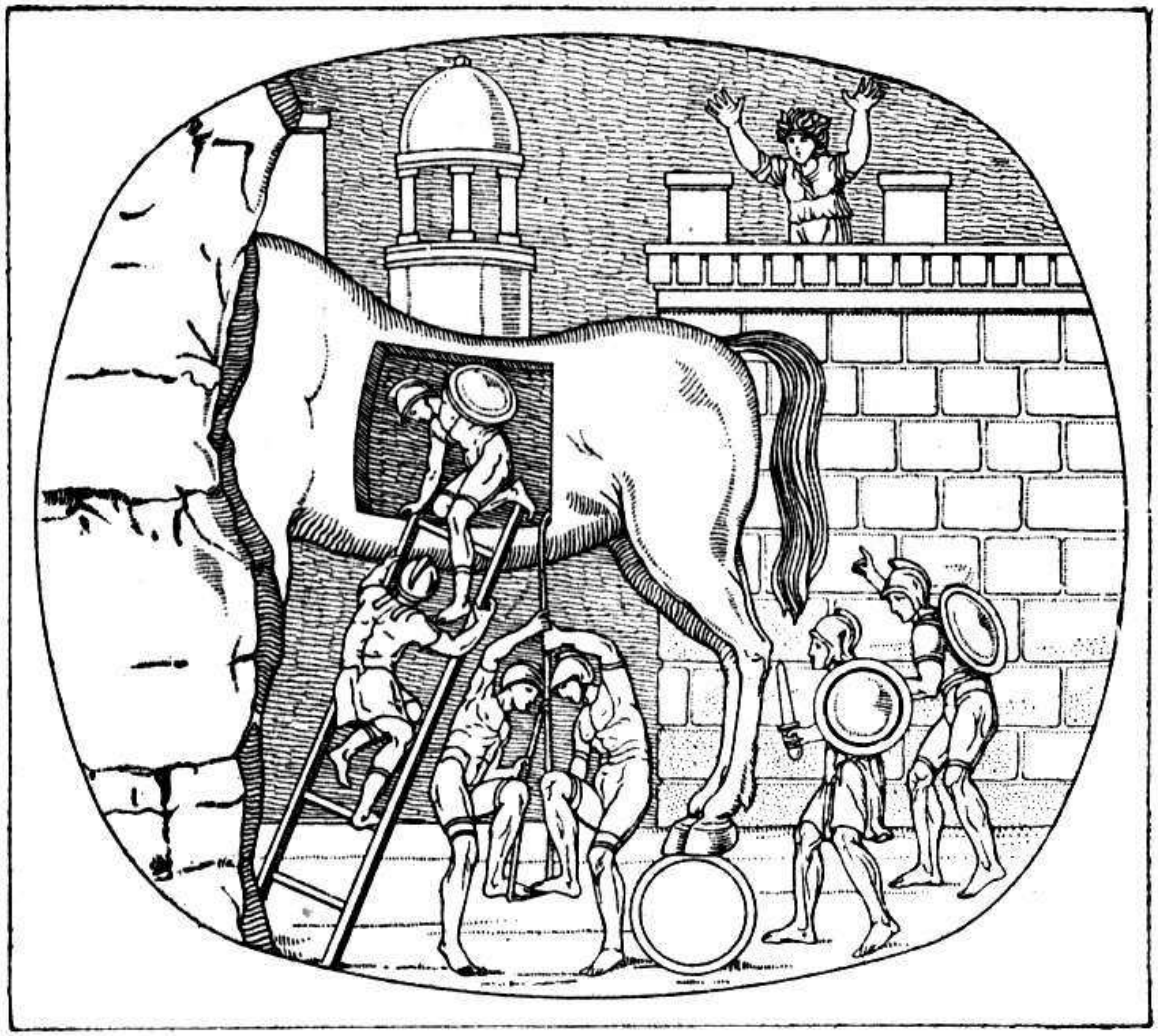
- For $\mathbf{x} = (x_1, x_2)^\top \in \mathbb{R}^2$, let $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$:

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



Kernels are Trojan Horses onto Linear Models

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



Designing Kernels

- As with distances, one can design kernels on any kind of object
  - time-series, strings, graphs, trees
  - images, video, audio, text
  - combination of the objects above!
- very large literature! too vast to discuss today.