4th Virtual Training Workshop in Bioinformatics

Classification in Bioinformatics: the SVM & Kernels

mcuturi@i.kyoto-u.ac.jp

Summary

- Objectives of these lectures:
  - Familiarize yourself with the general concept of **classification**
  - Learn how to **classify complex biological structures** with a **SVM**

- Outline of this short course:

1. Introduce **linear classifiers** in general...
2. ...the **support vector machine** in particular...
3. ...and its extension as a **kernel algorithm**.
4. we conclude by presenting a few **kernels for biological structures**.

to go beyond these lectures: a few pointers at the end of this document

The starting point: data

- The data-revolution: **more data, new datatypes, new databases**
  - All hard-drives of the planet (2009) $\approx$ 487 billion Gb $\approx$ 72Gb/person!
  - In 1993, internet traffic = 100 Tb/**year**. In 2008, 160 Tb/**second**
- Some of that data is scientifically relevant...
  - The LHC particle accelerator will produce 15 Mil Gb per year
  - Bioinformatics databases: see lectures by *Michael Gromiha* (CBRC)
- Key question: can we use this data to make **scientific discoveries**? how?

> **Statistical inference**: learn from **previously seen data** to make decisions **about new data.**

Statistical Inference

- **Statistical**: useful to study random systems...
  - Mutations, environmental changes $etc.\longrightarrow$ **life is random**!
- **Inference**: learn rules using observations assuming some "stationarity".

"**yes/no**" rules = "**binary classification**"

  - given a protein sequence, does it belong to the functional class `ABC`?
  - given a patient's genome, is it safe/effective to give him medecine `XYZ`?
  - given a patient's genome, is (s)he at risk of developing Parkinson's disease?
  - given gene expression data of a tumor cell, is it benign/malign?

"**binary classification**" $\Rightarrow$ **simple predictions** for **well-understood problems**

Statistical Inference

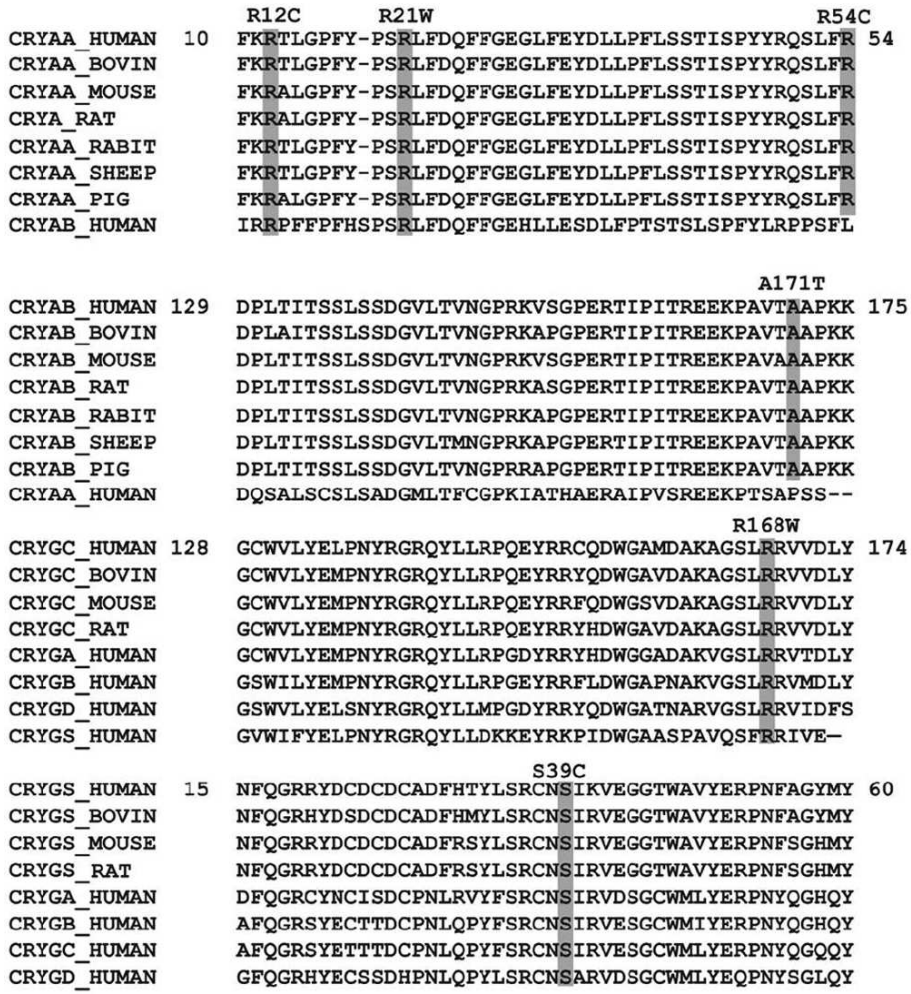
We will not discuss...

- *Multiclass classification*: provide one among **many (≥ 3)** possible answers;
- *Clustering*: create the taxonomy (classes) and the answers at the same time;
- *Regression*: provide **real-valued answers** (in $\mathbb{R}$ or $\mathbb{R}^d$);
- *Structured Output Regression*: the answer is a whole new object ($e.g.$predict a whole 3D structure)

... but it useful to understand *binary* classification
to understand the problems above.

Statistical Inference on Biological Datatypes

- What do we have in bioinformatics databases?
  - very long sequences (proteins, DNA)

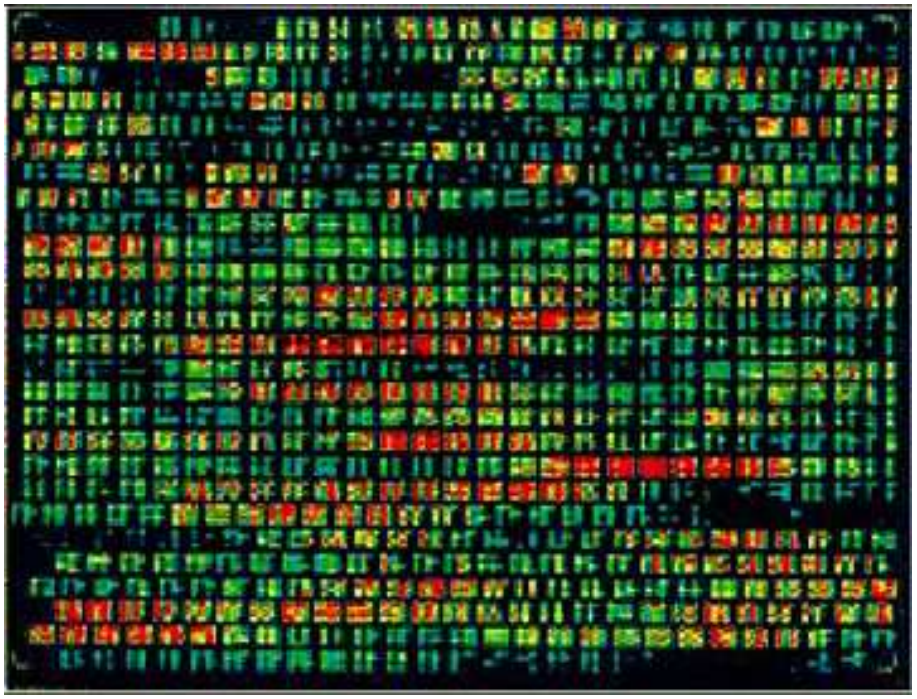


Statistical Inference on Biological Datatypes

- very complex 3D structures (protein folds, molecules)

Statistical Inference on Biological Datatypes

- high-definition images corrupted by noise (DNA-chips)



...*etc.*

Yet.... Vectors??

- Yet... the natural datatype for algorithms is **vectors**, e.g. `Matlab`.

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

  - easy to store in memory, easy to manipulate (+,-,x,/),
  - well understood in mathematics ($\mathbb{R}^d$),
  - probability theory and statistics (multivariate analysis).

- We will first study classification algorithms **tailored for vectors**.

- You might say: *But biological data is* $\boldsymbol{never}$ *formatted as a vector in real-life!*
  ...why study $\boldsymbol{vector}$*-based algorithms?*

Kernels are the **trojan-horses** which will help us
deal with complex structures using **algorithms tailored for vectors** [11]

The training set

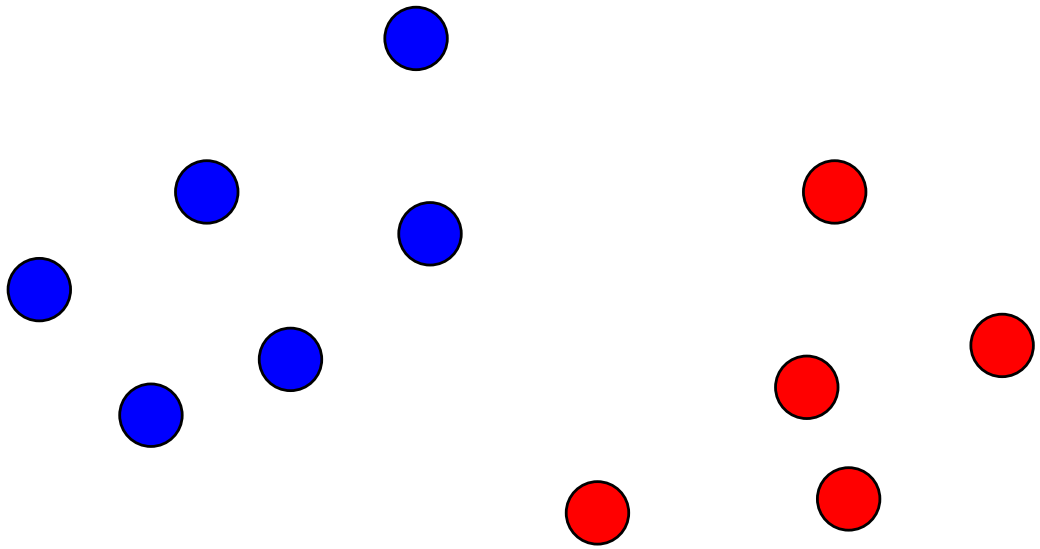
- The **Data** we have: a bunch of vectors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_N$.
- Ideally, to infer a "yes/no" rule, we need the **correct answer** for each vector.
- We consider thus a set of **pairs of variables**

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

- 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**.

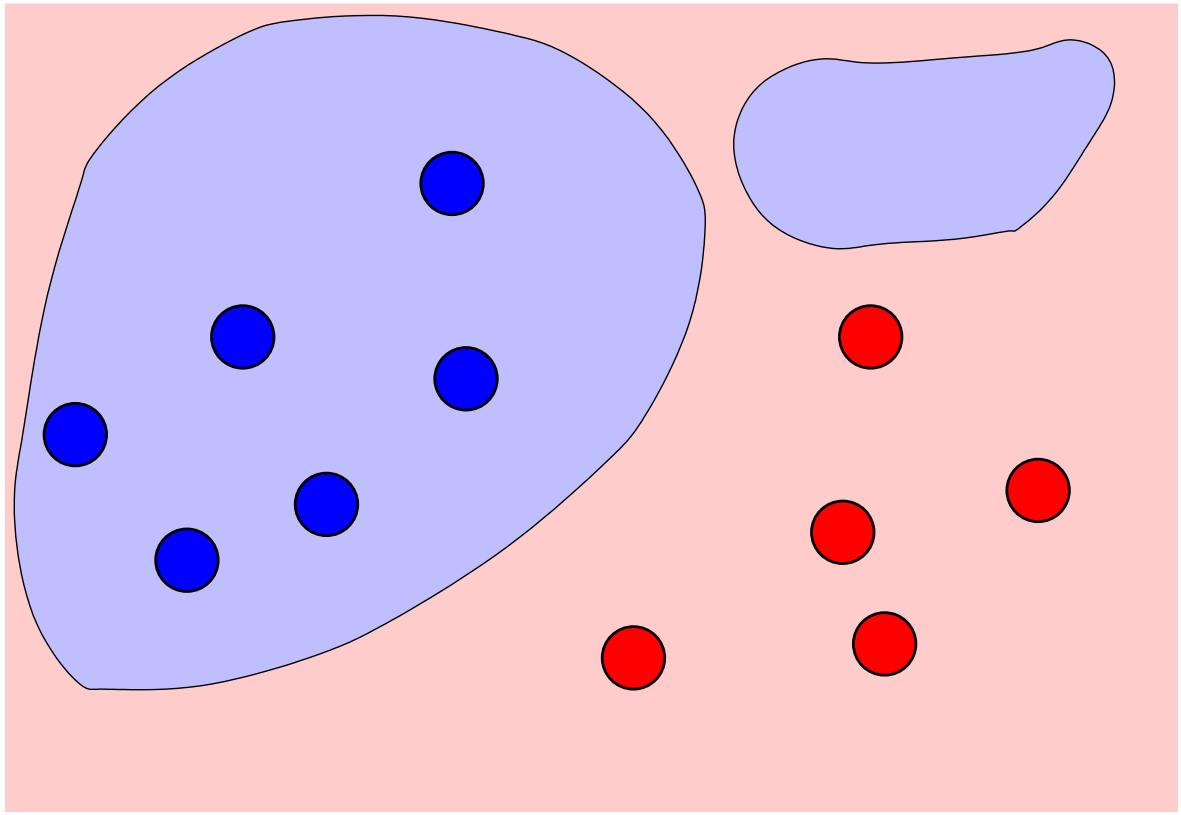
Many thanks to J.P. Vert for some of the following slides

Classification Separation Surfaces for Vectors



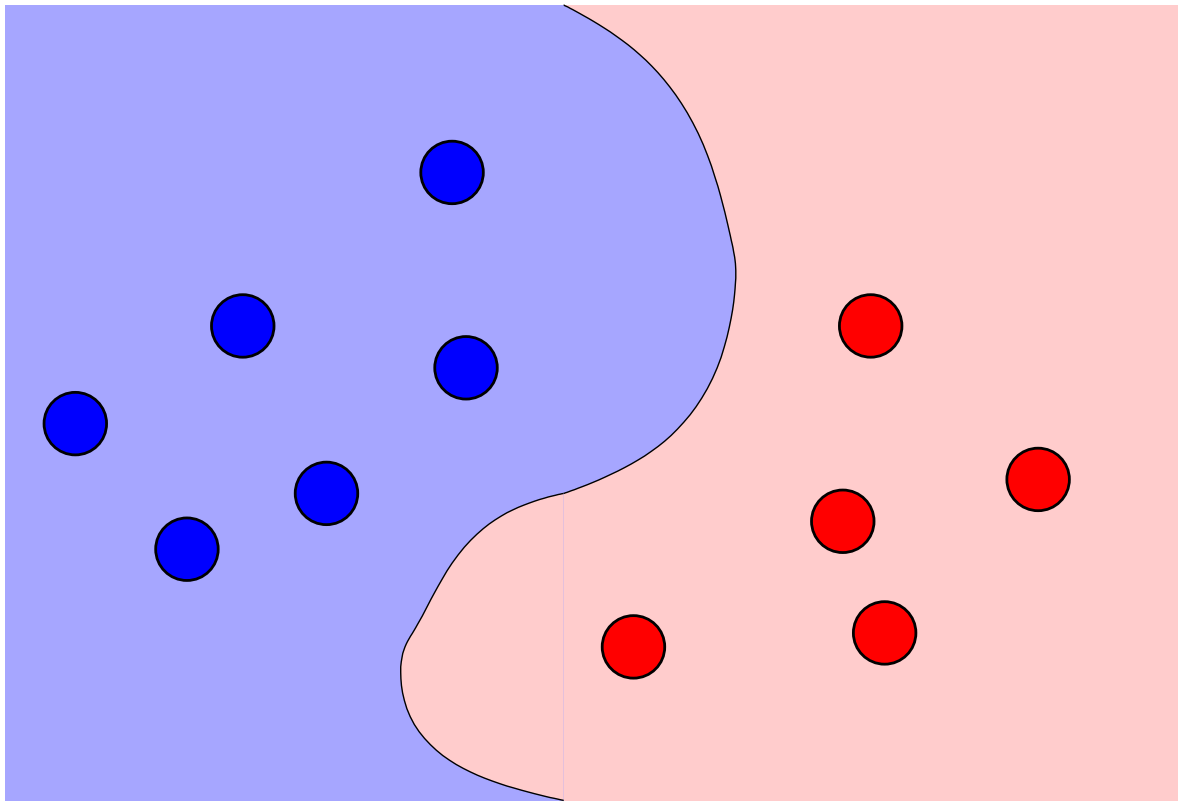
What is a classification rule?

Classification Separation Surfaces for Vectors



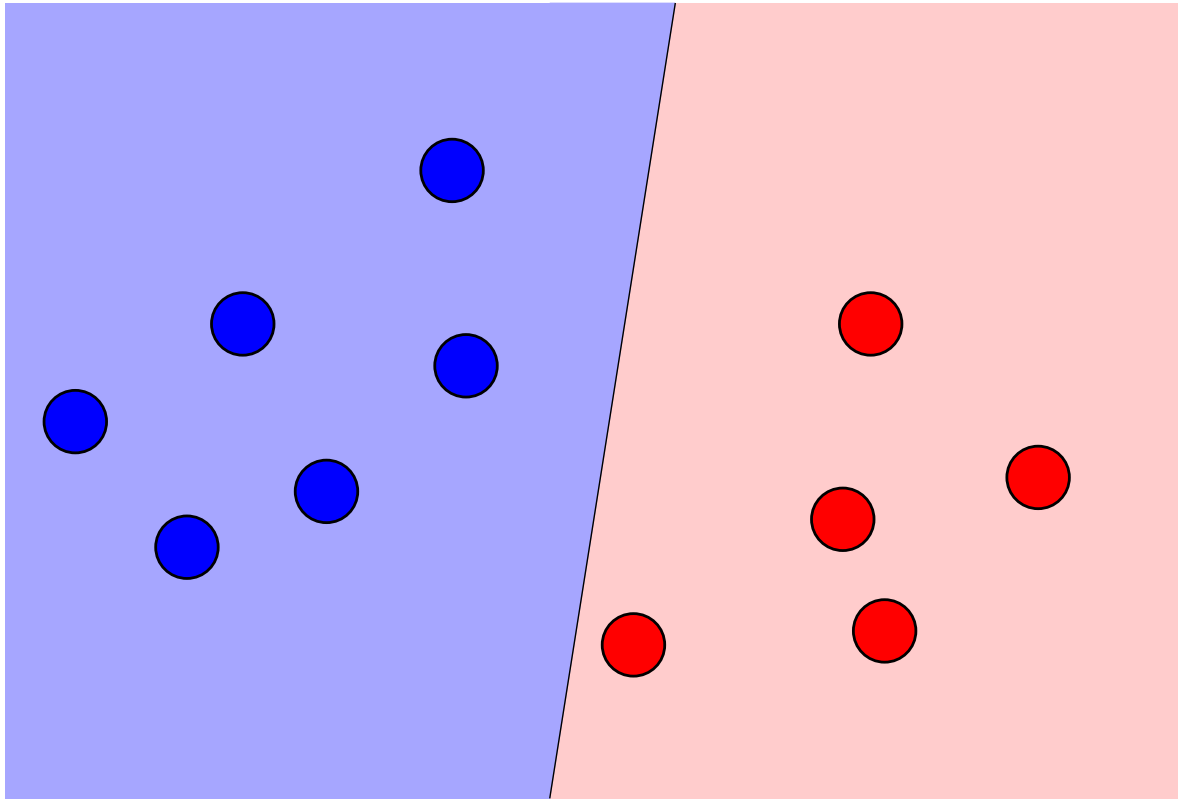
Classification rule = **separation surface**. A surface in 2D is a line. A loop here

Classification Separation Surfaces for Vectors



A curved line

Classification Separation Surfaces for Vectors



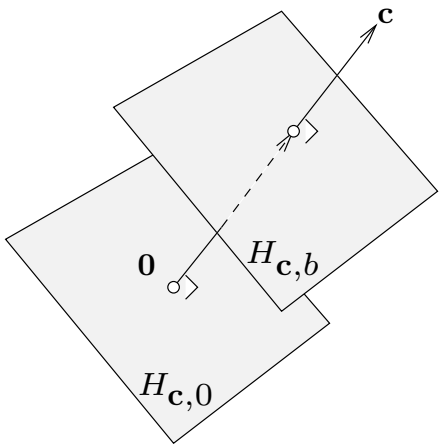
Even more **simple**,
consider **straight lines**?

Linear Classifiers

- **Straight lines** (hyperplanes when $d > 2$) are a **very powerful tool**.
- 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 $\mathbb{R}^d$

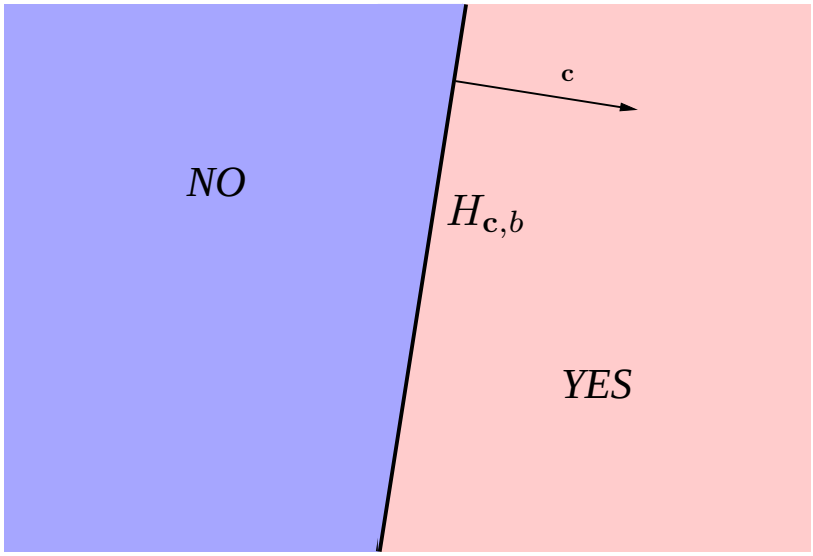


Linear Classifiers

- 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,

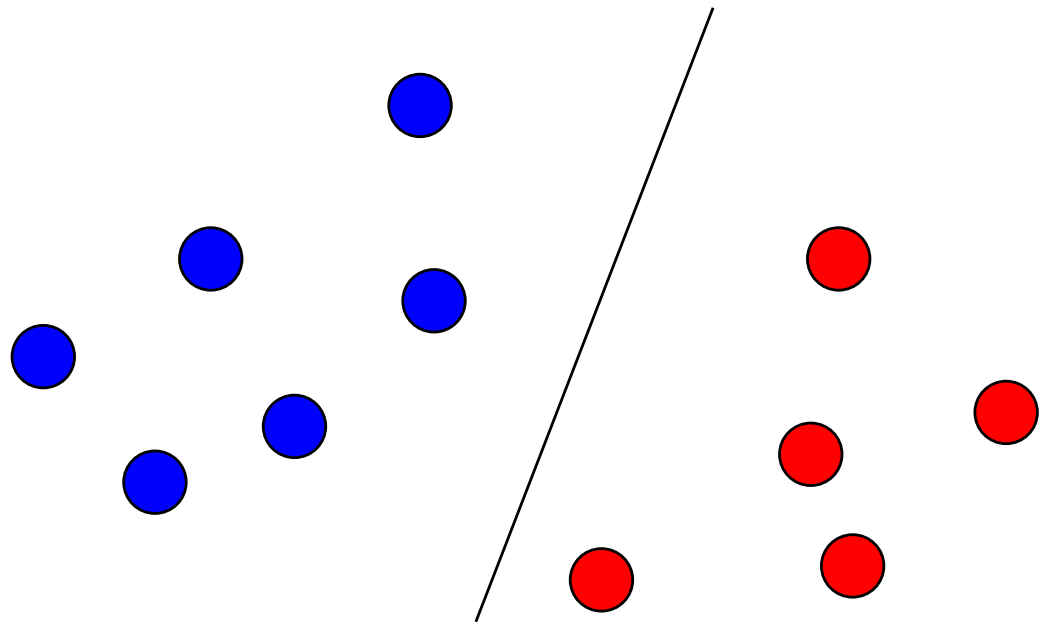
$$\text{“training set”} \quad \left\{ \left(\mathbf{x}_i \in \mathbb{R}^d,\ \mathbf{y}_i \in \{0,1\} \right)_{i=1..N} \right\} \overset{????}{\Longrightarrow} \text{“best”} \mathbf{c}^\star,\ b^\star$$

  has different answers. Depends on the meaning of “best” [4]:

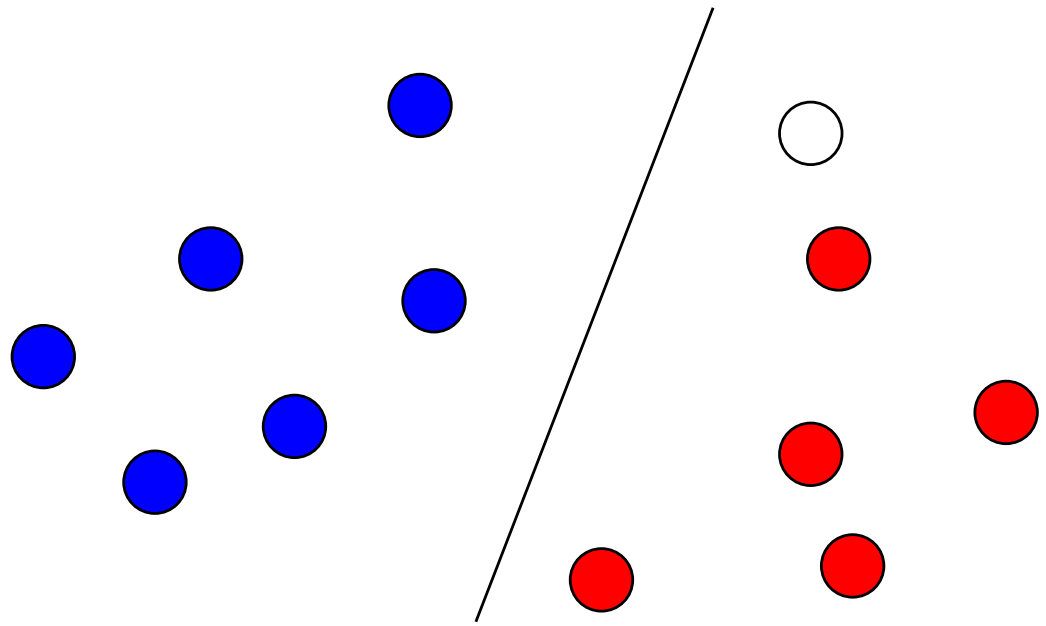
- **Linear Discriminant Analysis** (or Fisher’s Linear Discriminant);
- **Logistic regression** maximum likelihood estimation;
- **Perceptron**, a one-layer neural network;
- *etc.*

Today’s focus: the **Support vector machine** [10]

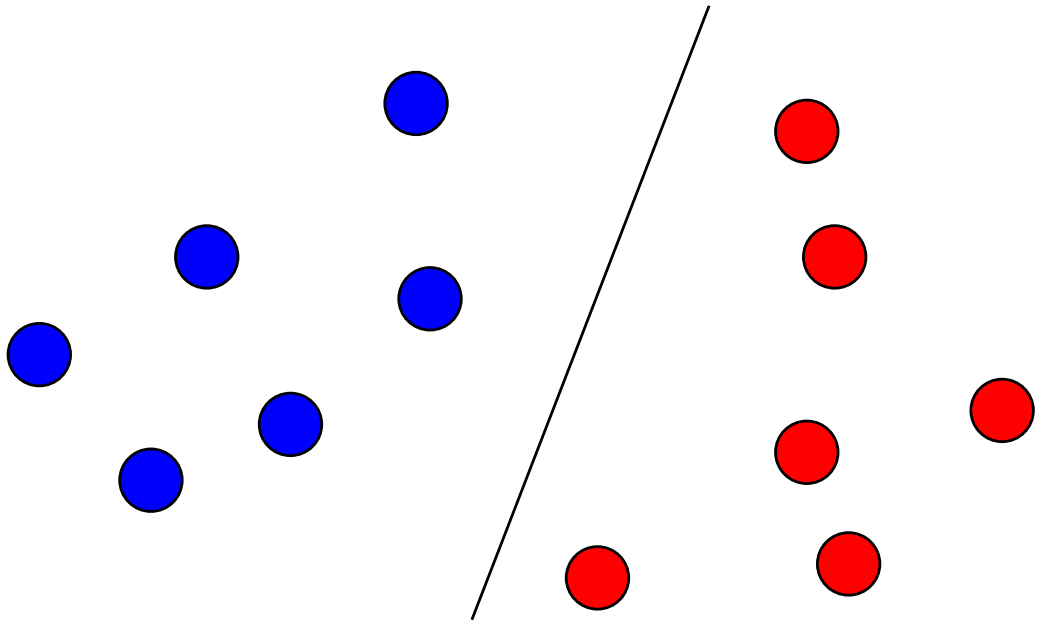
Classification Separation Surfaces for Vectors



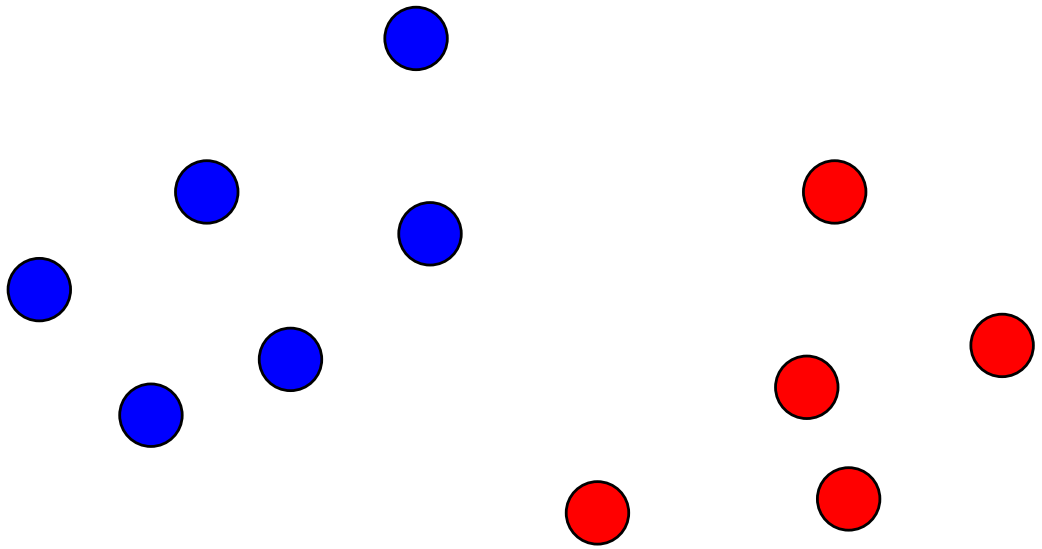
Classification Separation Surfaces for Vectors



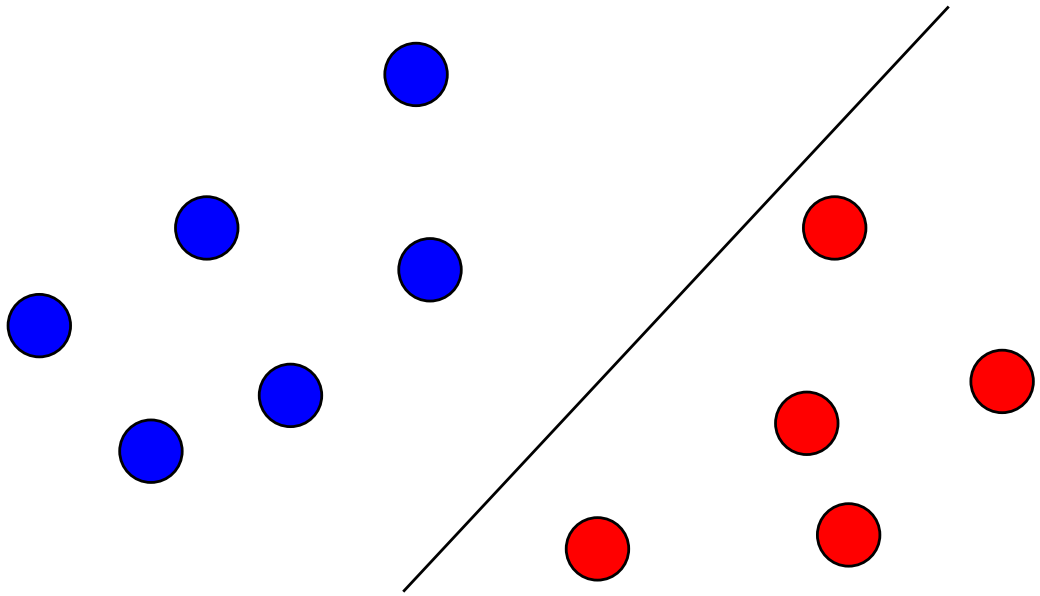
Classification Separation Surfaces for Vectors



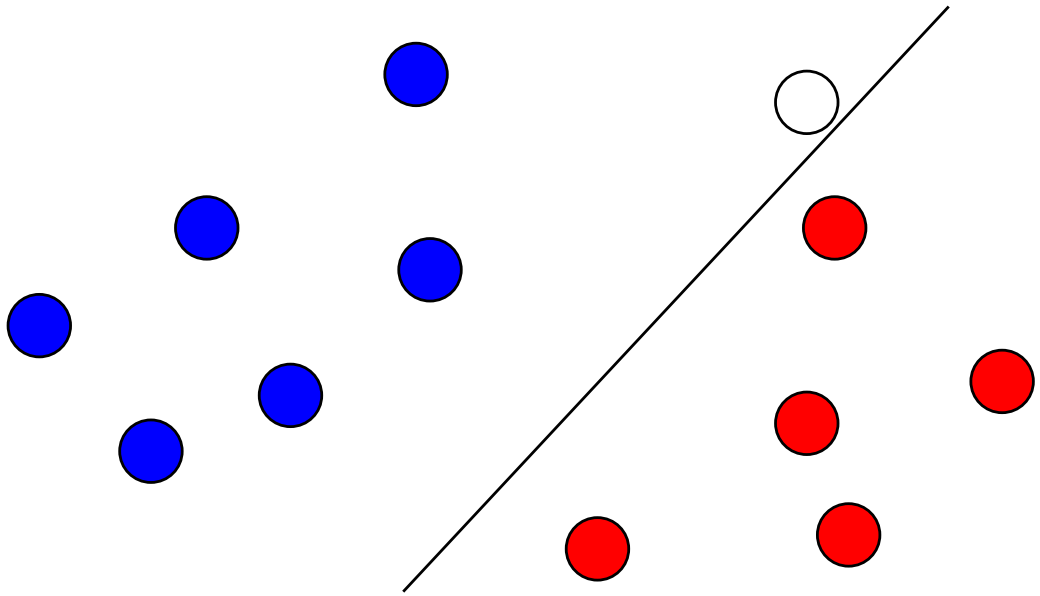
Linear classifier, some degrees of freedom



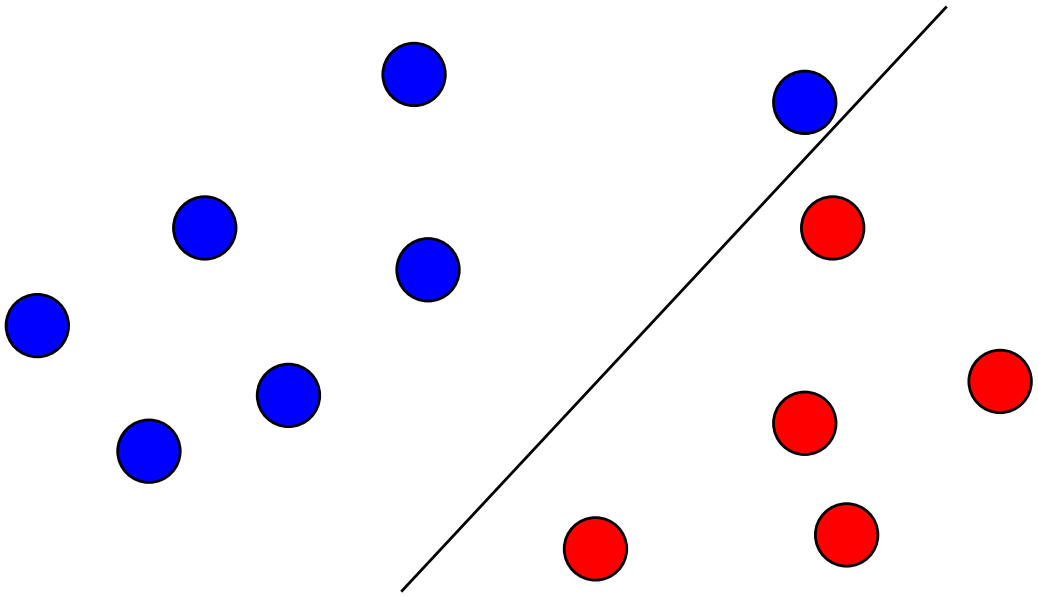
Linear classifier, some degrees of freedom



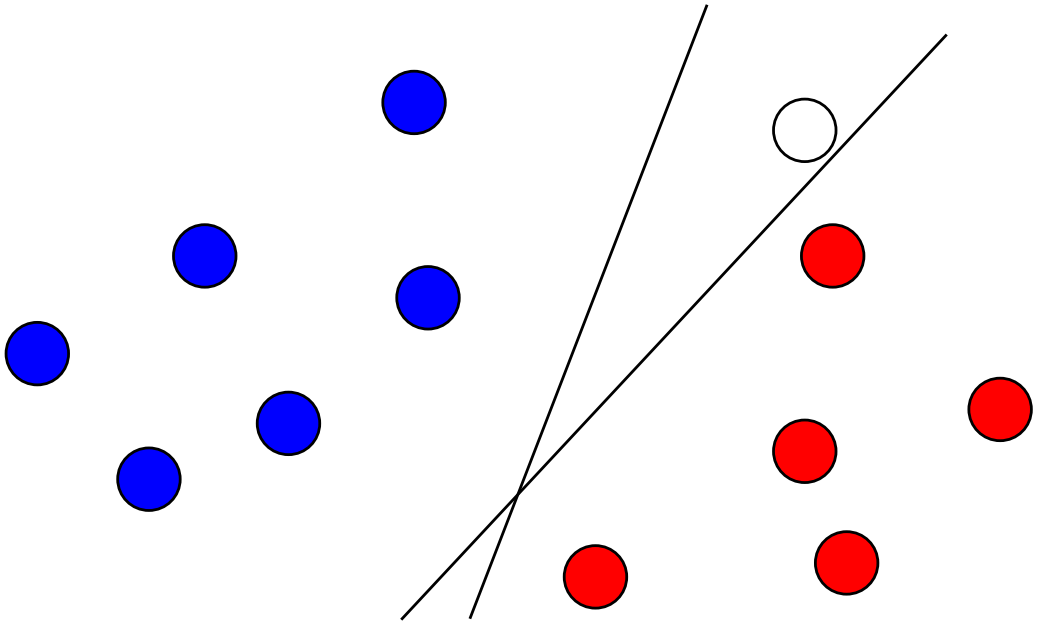
Linear classifier, some degrees of freedom



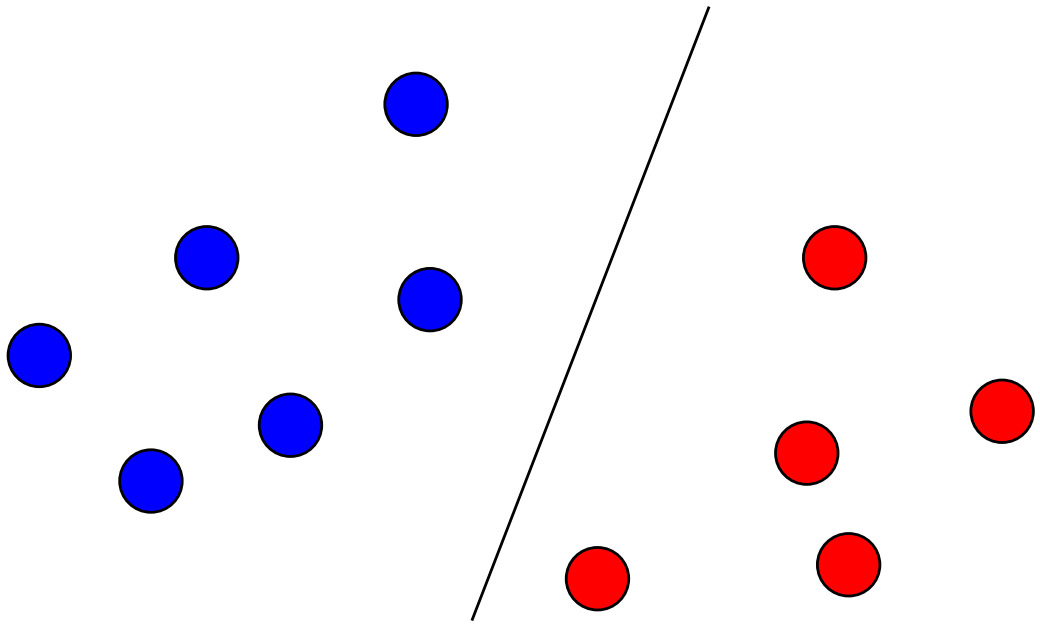
Linear classifier, some degrees of freedom



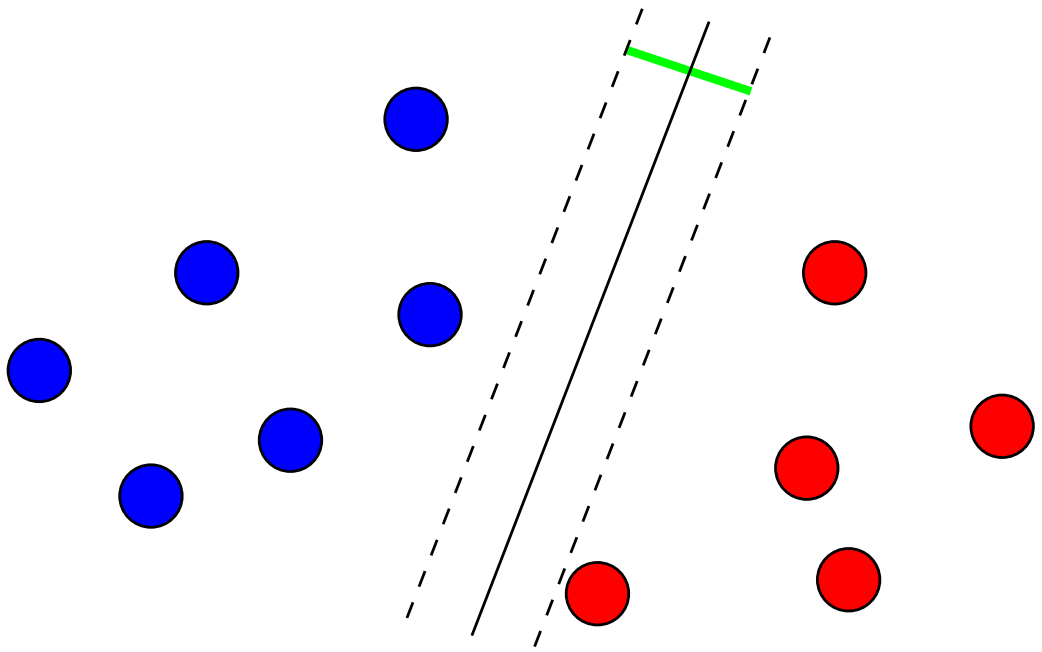
Which one is better?



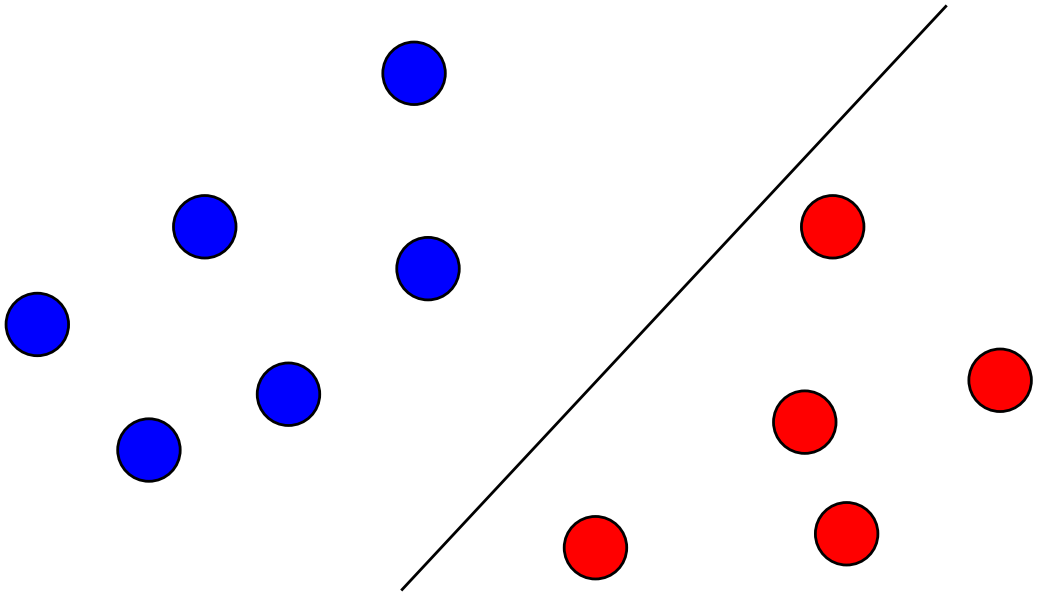
A criterion to select a linear classifier: the margin [1]



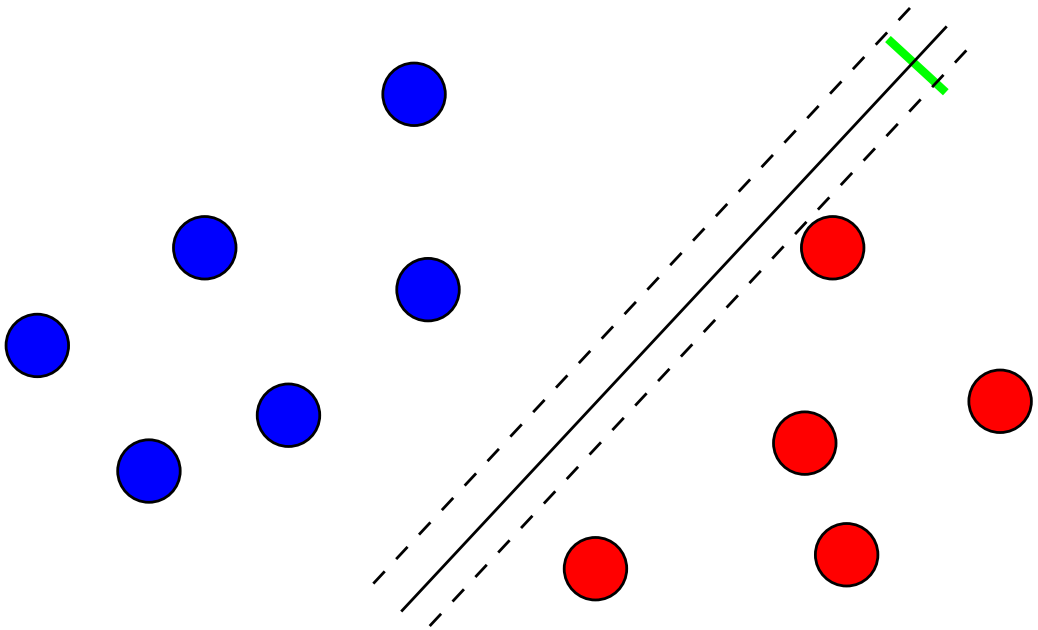
A criterion to select a linear classifier: the margin [1]



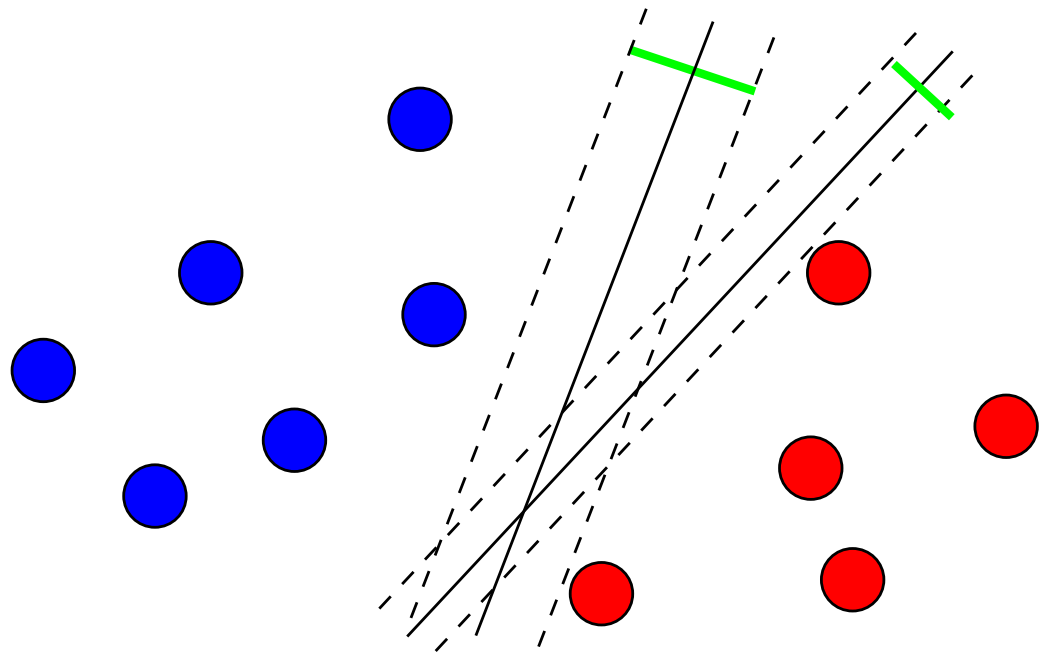
A criterion to select a linear classifier: the margin [1]



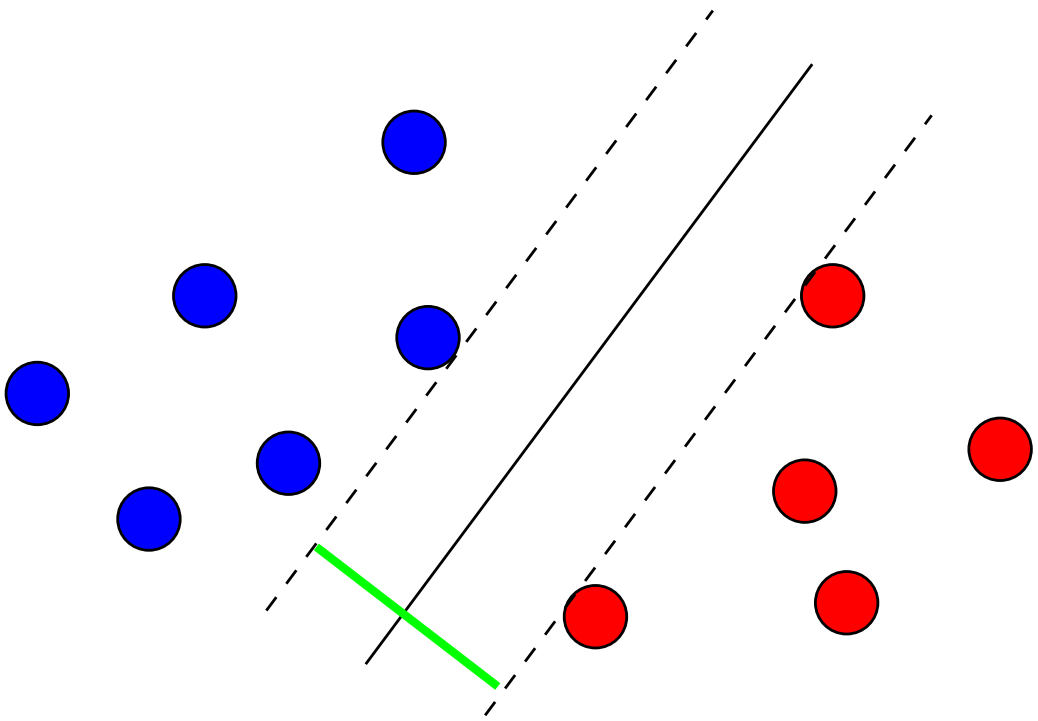
A criterion to select a linear classifier: the margin [1]



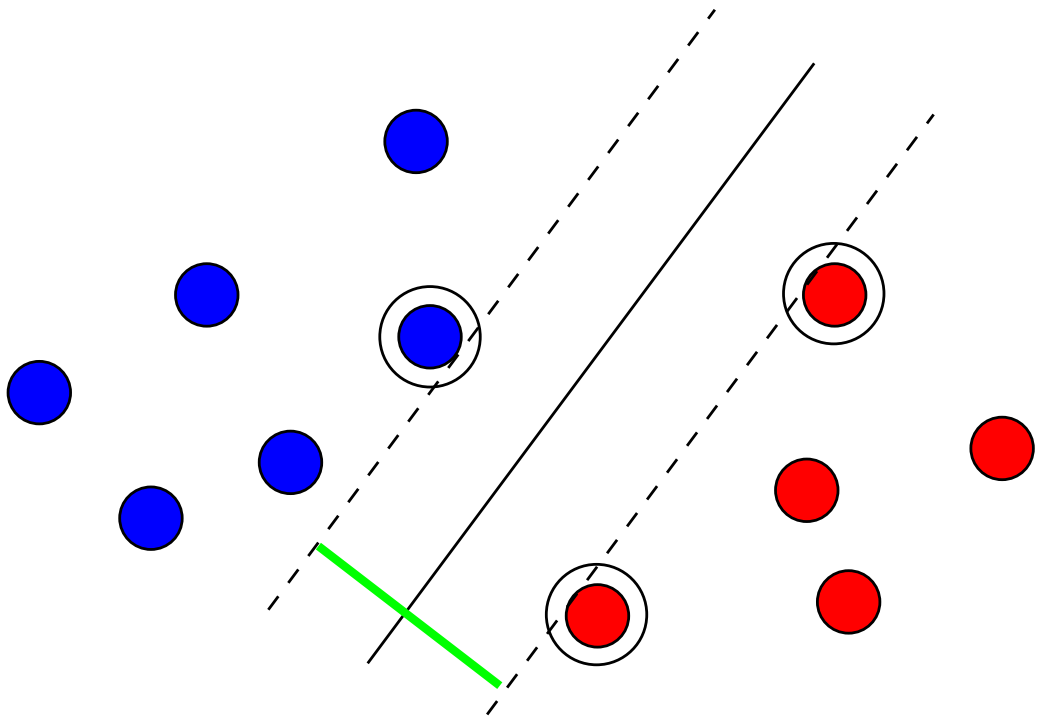
A criterion to select a linear classifier: the margin [1]



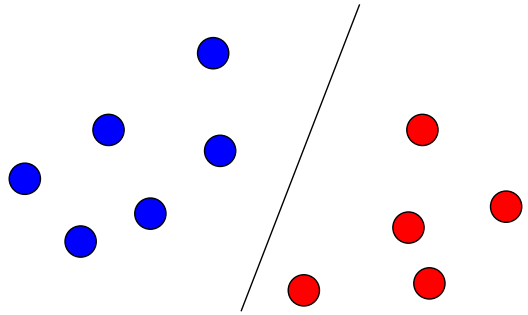
Largest Margin Linear Classifier [1]



Support Vectors with Large Margin



In equations



- 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:

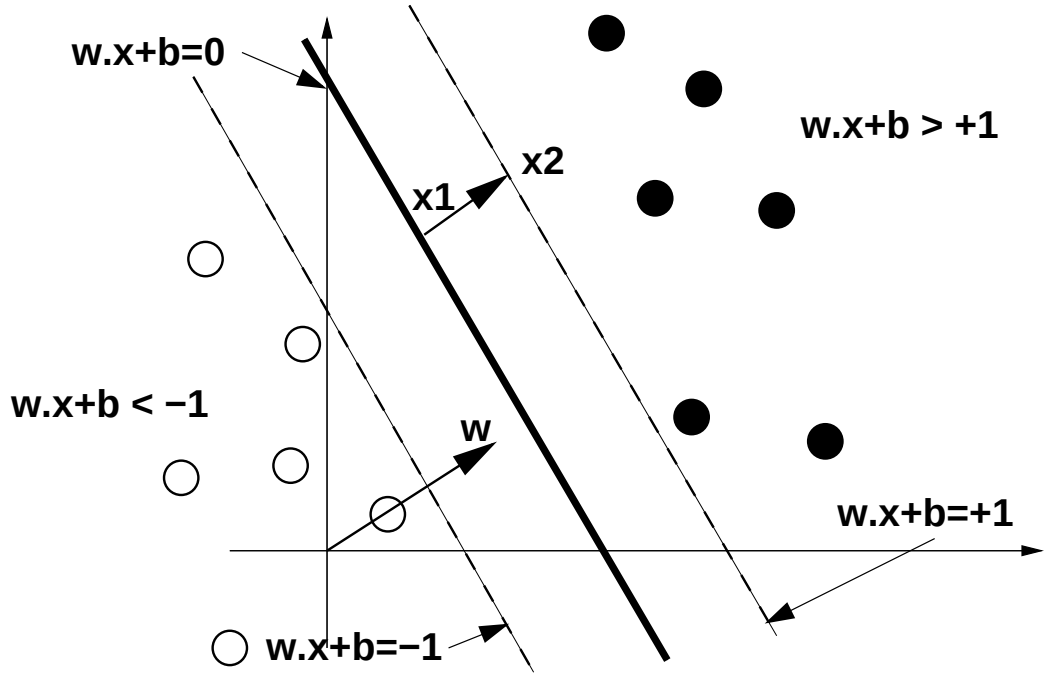
$$\begin{cases} \mathbf{w}^T \mathbf{x}_i + b > 0 & \text{if } \mathbf{y}_i = 1\,, \\ \mathbf{w}^T \mathbf{x}_i + b < 0 & \text{if } \mathbf{y}_i = -1\,. \end{cases}$$

- Next, we give a formula to compute the margin as a function of $\mathbf{w}$.

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:

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



- Consider $\mathbf{x}_1$ and $\mathbf{x}_2$ such that $\mathbf{x}_2 - \mathbf{x}_1$ is parallel to $\mathbf{w}$.

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

- Margin $= 2/\|\mathbf{w}\|$: 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:

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

  where γ is by definition the **margin**.

All training points should be on the appropriate side

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

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

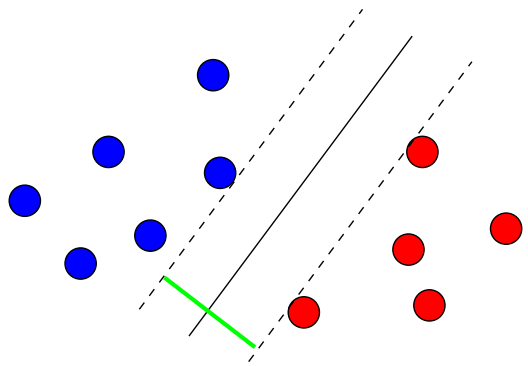
- For negative examples ($y_i = -1$) this means:

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

- 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



- Find $(\mathbf{w}, b)$ 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

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

is only defined when

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

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

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. 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 $\mathbf{y_i} \left(\mathbf{w}^T \mathbf{x_i} + \boldsymbol{b} \right) = \mathbf{1}$.
- $\alpha_i \neq 0$ **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 $\alpha_i > 0$, $\mathbf{y}_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$,

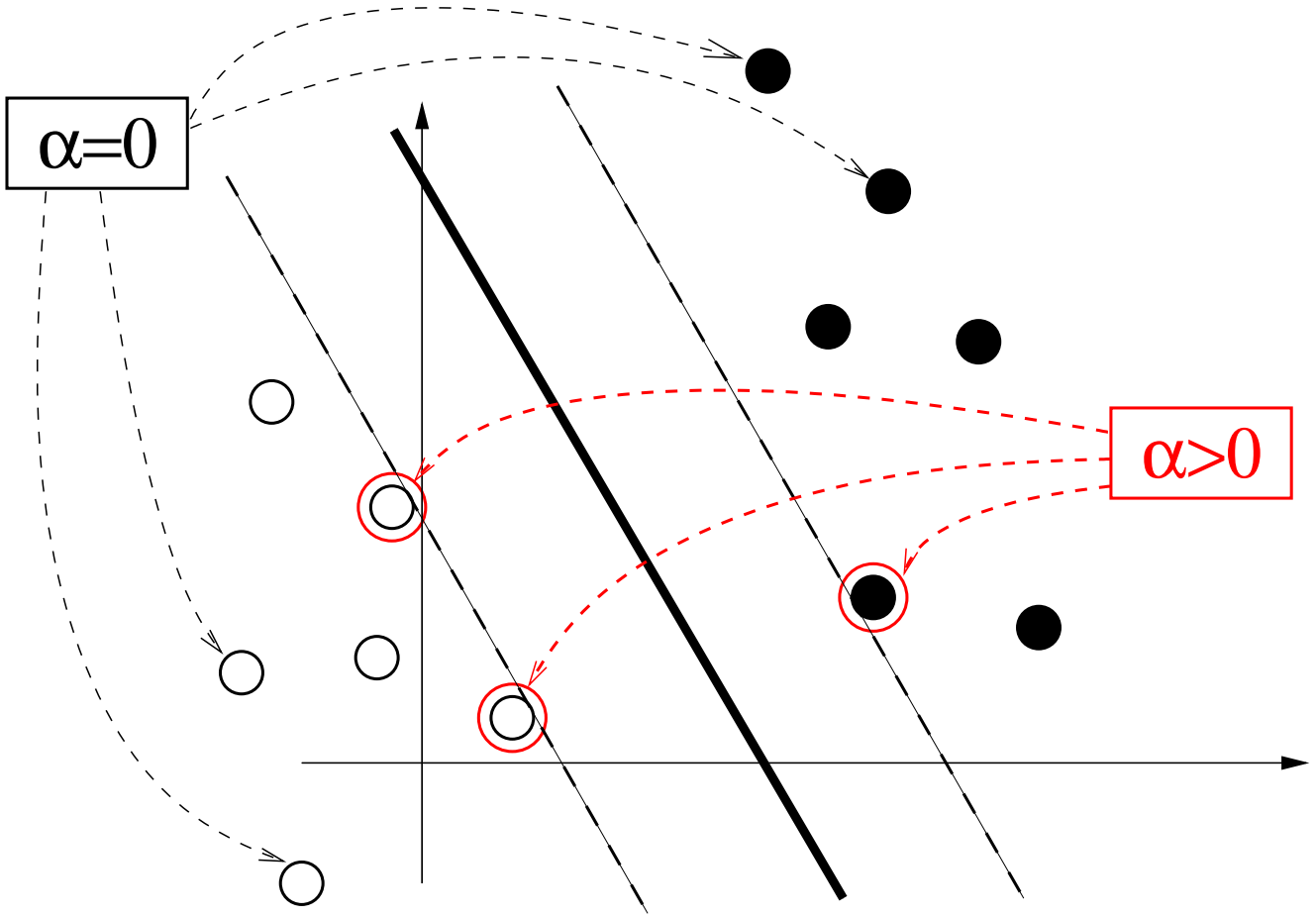
$$b^* = -\frac{1}{2}\left(\min_{\mathbf{y}_i=1,\alpha_i>0}(\mathbf{w}^T\mathbf{x}_i) + \max_{\mathbf{y}_i=-1,\alpha_i>0}(\mathbf{w}^T\mathbf{x}_i)\right)$$

- the **decision function** is therefore:

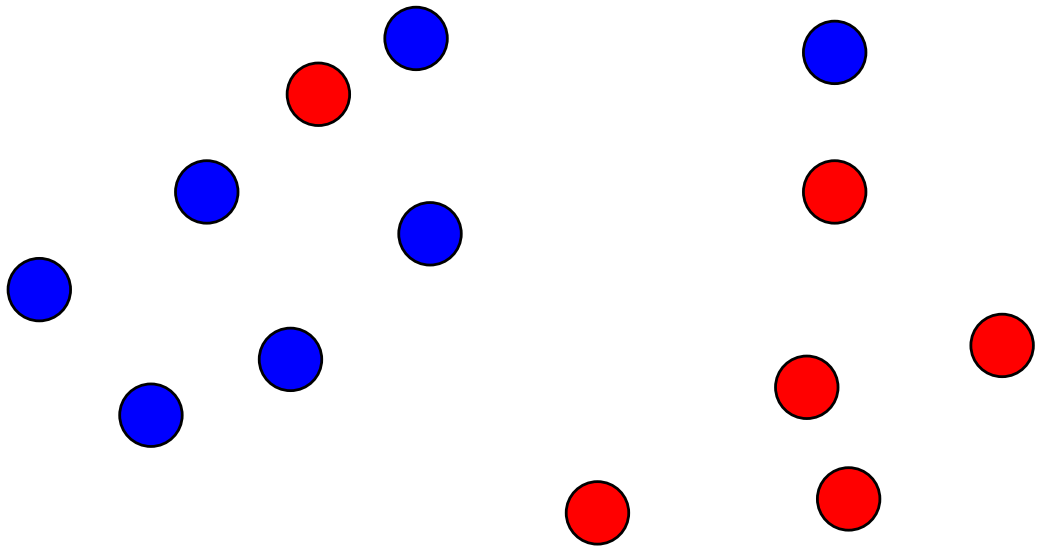
$$\begin{aligned} f^*(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b^* \\ &= \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i^T \mathbf{x} + b^*. \end{aligned}$$

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

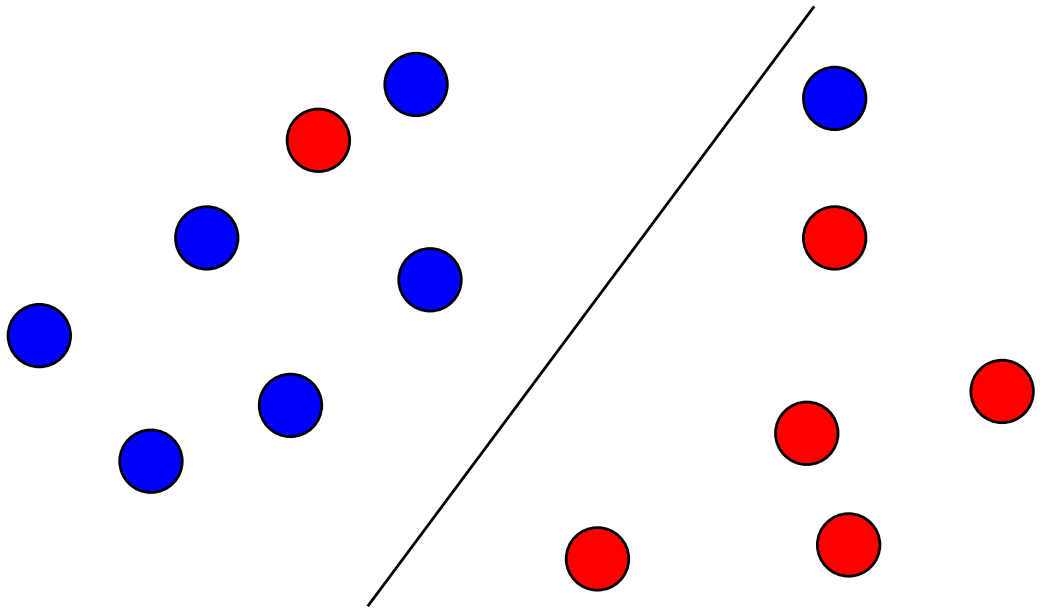
Interpretation: support vectors



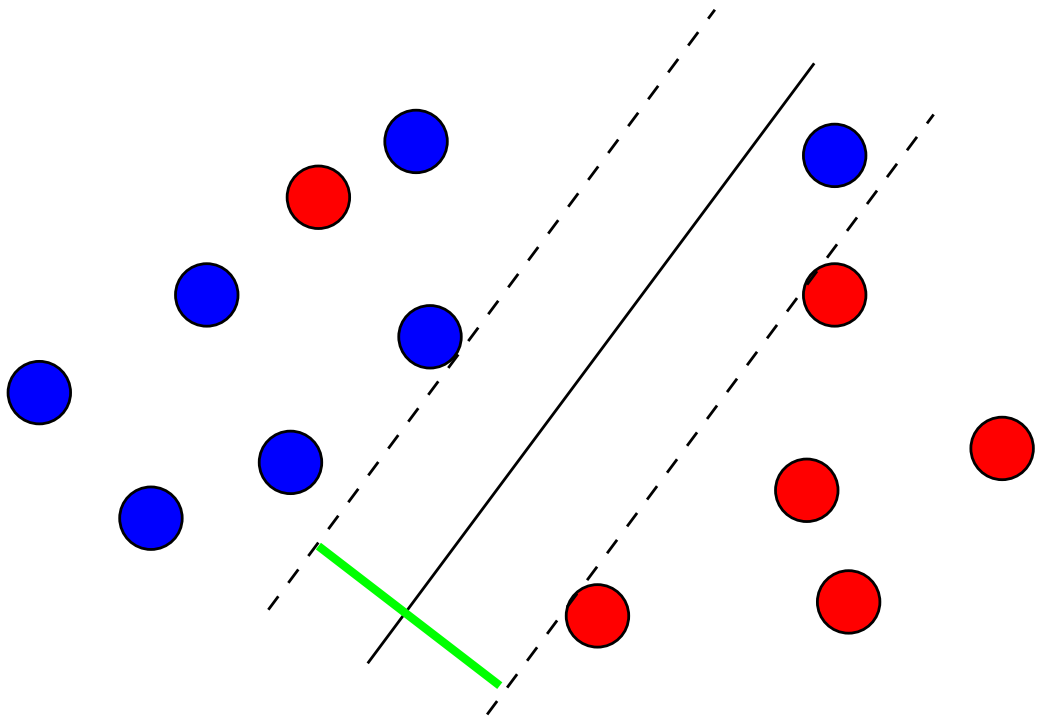
What happens when the data is not linearly separable?



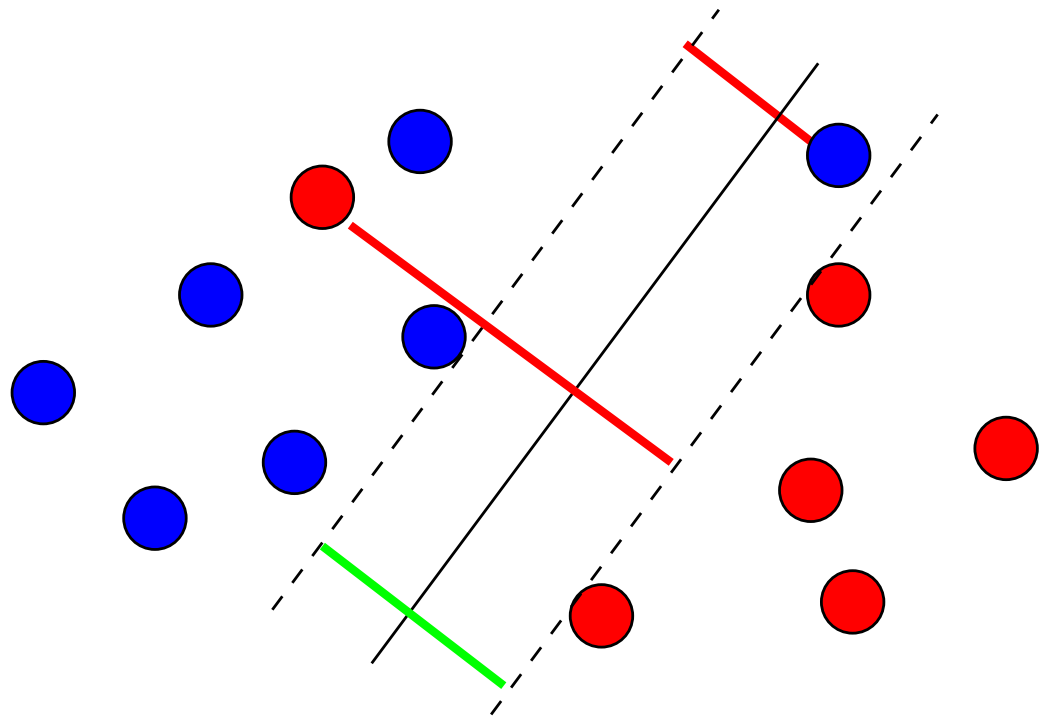
What happens when the data is not linearly separable?



What happens when the data is not linearly separable?



What happens when the data is not linearly separable?

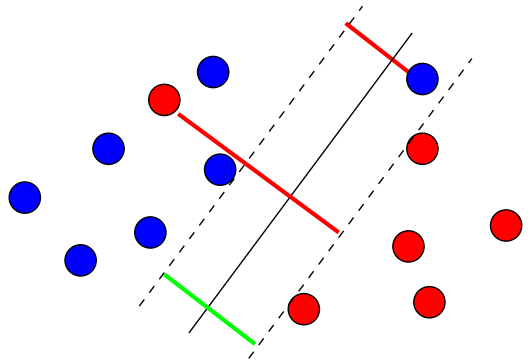


Soft-margin SVM [2]

- 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 [2]

- 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

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

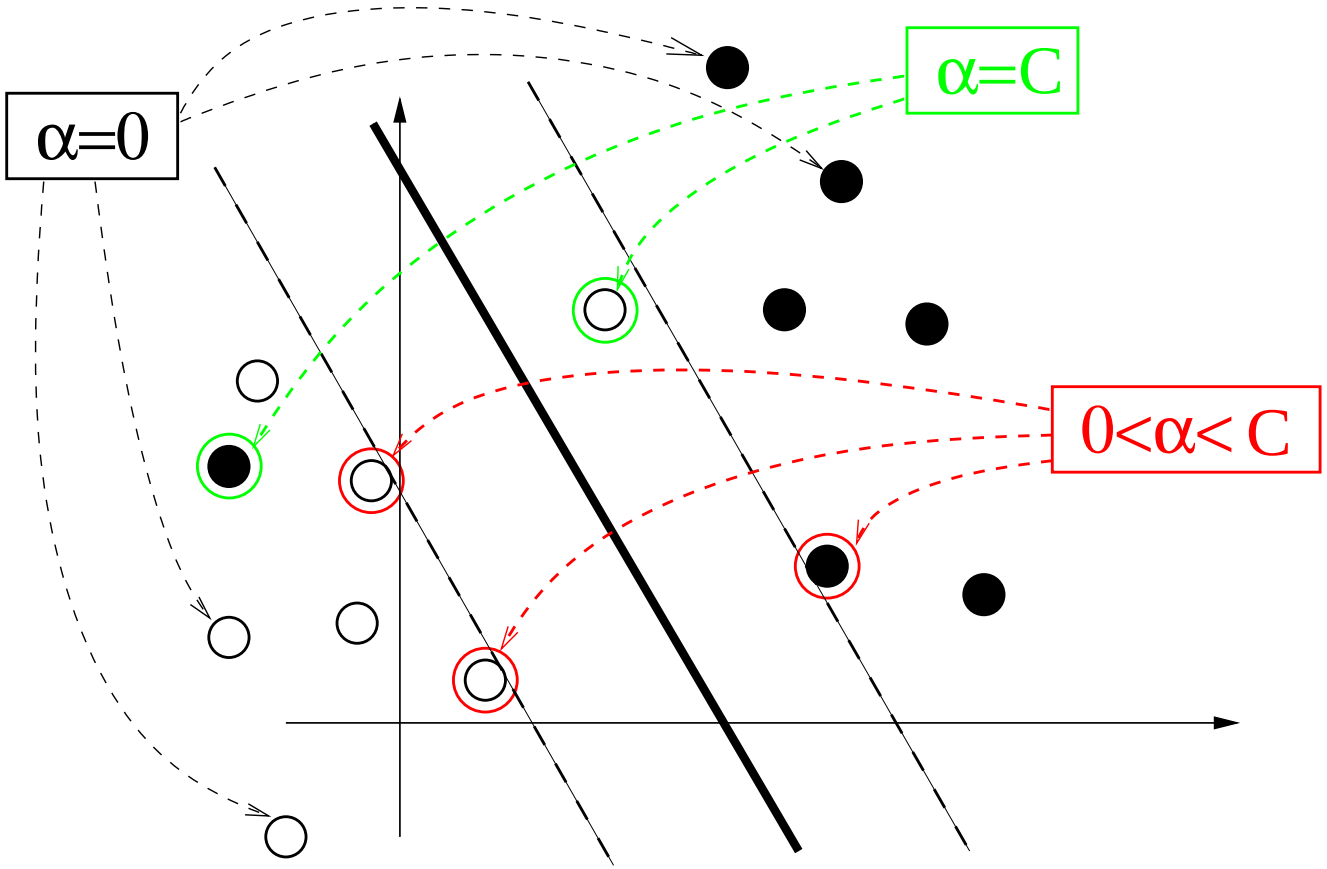
- In that case the dual function

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

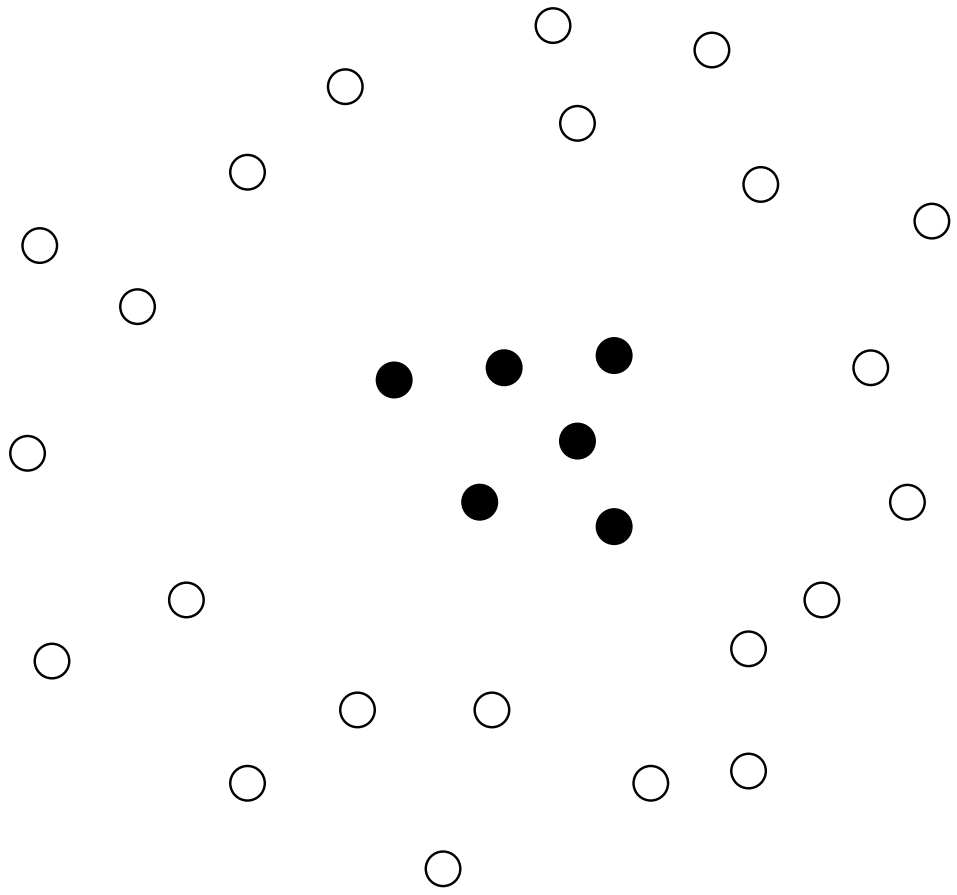
  which is finite under the constraints:

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

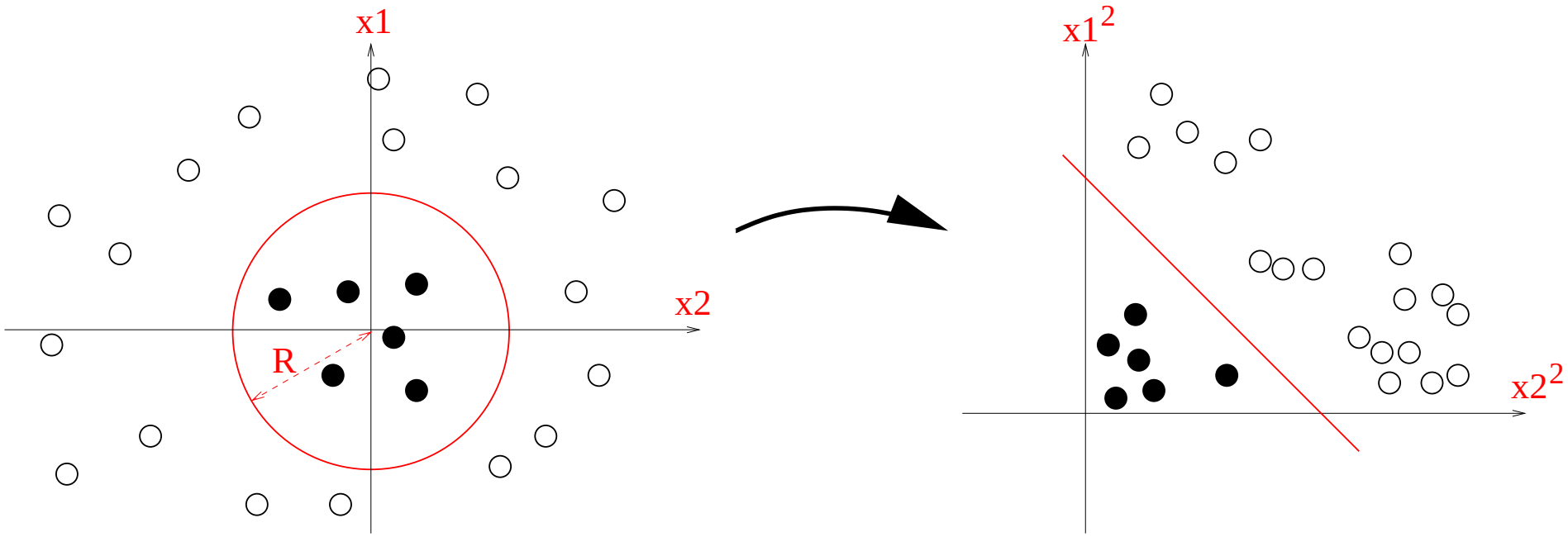
Interpretation: bounded and unbounded support vectors



Sometimes linear classifiers are of little use



Solution: non-linear mapping to a feature space



Let $\phi(\mathbf{x}) = (x_1^2, x_2^2)'$, $\mathbf{w} = (1,1)'$ and $b = 1$. Then the decision function is:

$$f(\mathbf{x}) = x_1^2 + x_2^2 - R^2 = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b,$$

Kernel trick for SVM's [2]

- 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 [10]

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 $\mathbf{y}$:

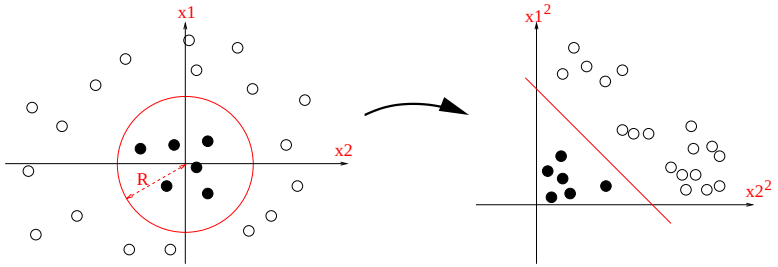
$$\begin{array}{ll} \text{maximize} & g(\alpha) = \alpha^T \mathbf{1} - \frac{1}{2}\alpha^T (\mathbf{y}^T \boldsymbol{K} \mathbf{y})\alpha \\ \text{such that} & 0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n \\ & \sum_{i=1}^n \alpha_i \mathbf{y}_i = 0. \end{array}$$

- $\boldsymbol{K}$'s **positive definiten** $\Leftrightarrow$ **problem has an 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 $\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \in \mathbb{R}^3$:

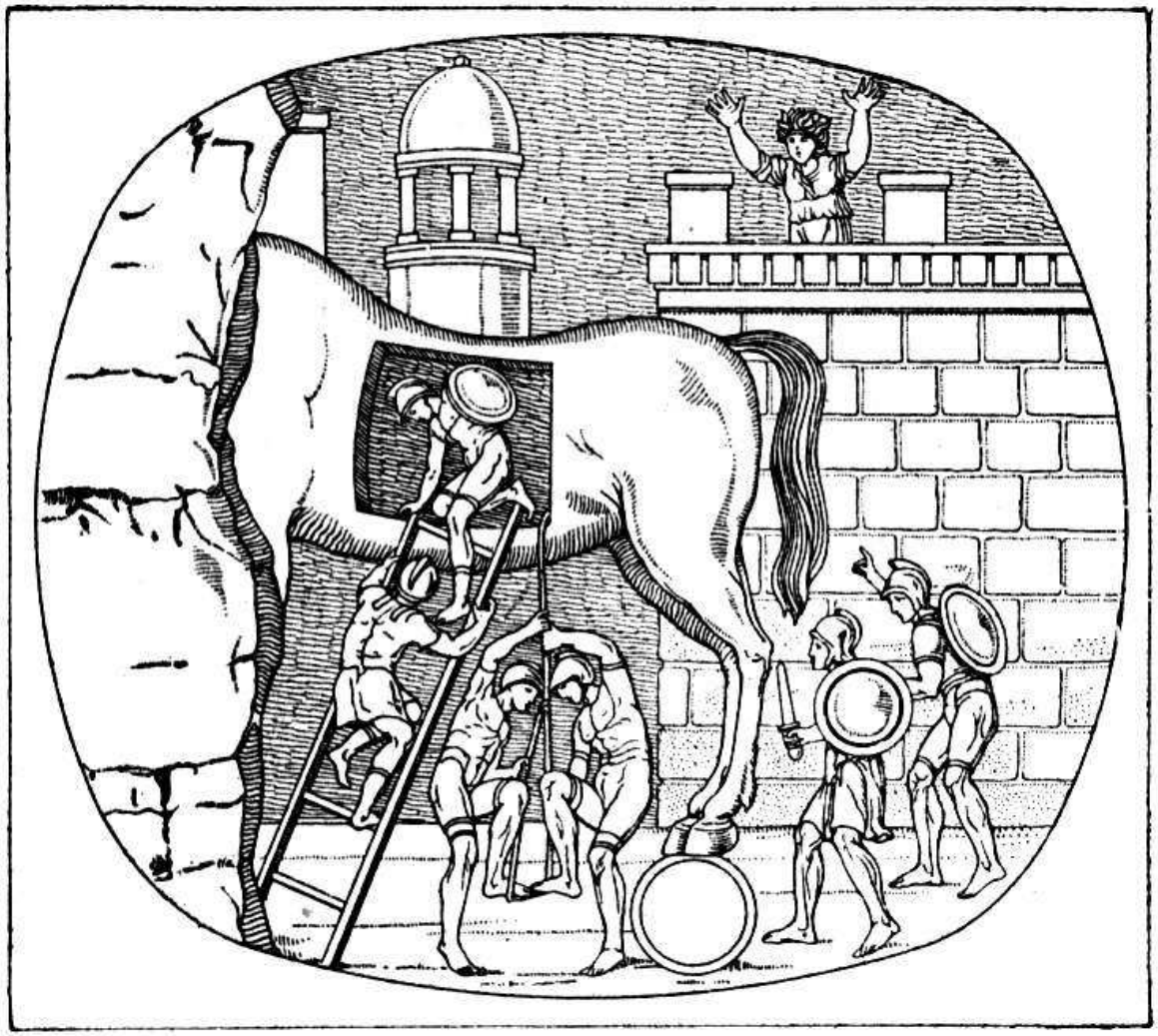
$$\begin{aligned} \boldsymbol{K(\mathbf{x}, \mathbf{x}')} &= x_1^2 x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2 \\ &= \{x_1x_1' + x_2x_2'\}^2 \\ &= \boldsymbol{\{\mathbf{x}^T\mathbf{x}'\}^2} \, . \end{aligned}$$



- Many more:

Kernels are Trojan Horses onto Linear Models

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



Some kernels for biological structures

Kernels for Sequences

Sequences in Biological Sequences

- DNA sequences: 3 billion bases (ATGC) long sequence.
- Protein sequences: variable-length word of a 20-letter alphabet

Challenges to define a good sequence kernel

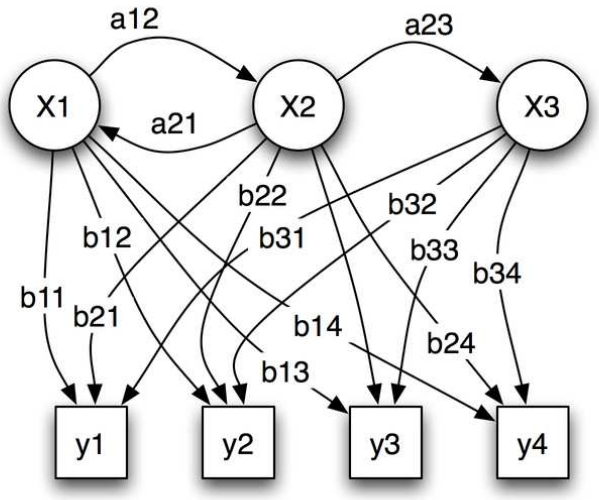
- **positive-definiteness**
- small computational effort required to compute $k(\mathbf{x}_i, \mathbf{x}_j)$
  ...for N points we have to compute N^2 similarities...
- ability to handle **variable-length** data.

Kernels for Sequences

- Existing tools:
  - **Sequence alignments**: BLAST, Smith-Waterman

```
Query:   61 KDAELNKAIPELEYMARYDAVTQDLADLGDKPYEYGKPLPHETGNKAIGWLYCAEGSNLG 120
            KDAELNKAIPELEYMARYDAVTQDL DLG++PY++ K LP+E GNKAIGWLYCAEGSNLG
Sbjct:   82 KDAELNKAIPELEYMARYDAVTQDLKDLGEEPYKFDKELPYEAGNKAIGWLYCAEGSNLG 141
```

  - Estimate statistical models for each class, $e.g.$ **Hidden Markov Models**



- Problem... no **positive definiteness**.

Kernels for Sequences

Finite set of features

- Represent strings as histograms of subsequences (spectrum [7], weighted degree [12])
- Use possible mismatches in these representations to account for mutations [8, 6]

Infinite and advanced features

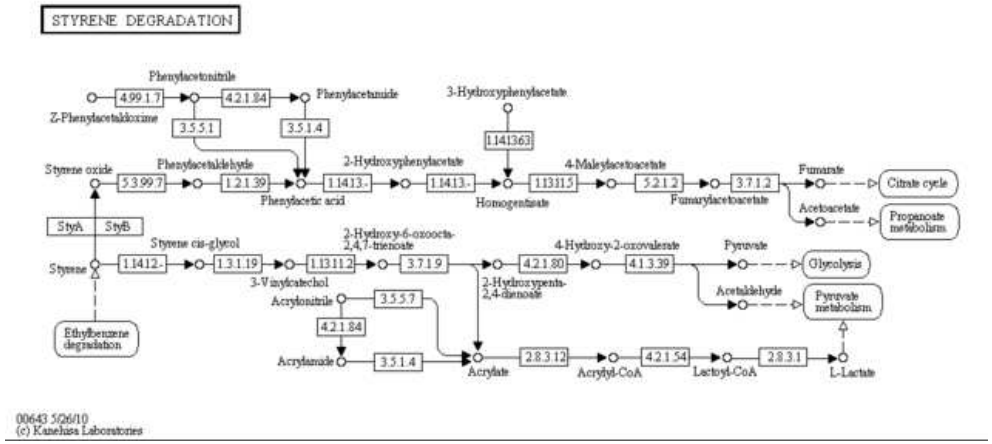
- Use probabilistic models to generate features $p_\theta(\mathbf{s})$ where θ is a parameter,

$$k(\mathbf{s}, \mathbf{s}') = \int_{\theta \in \Theta} p_\theta(\mathbf{s}) p_\theta(\mathbf{s}') \omega(d\theta). \text{ [3]}$$

- Use feature representations that translate into sums over **all possible alignments** [13]

Kernels for Graphs

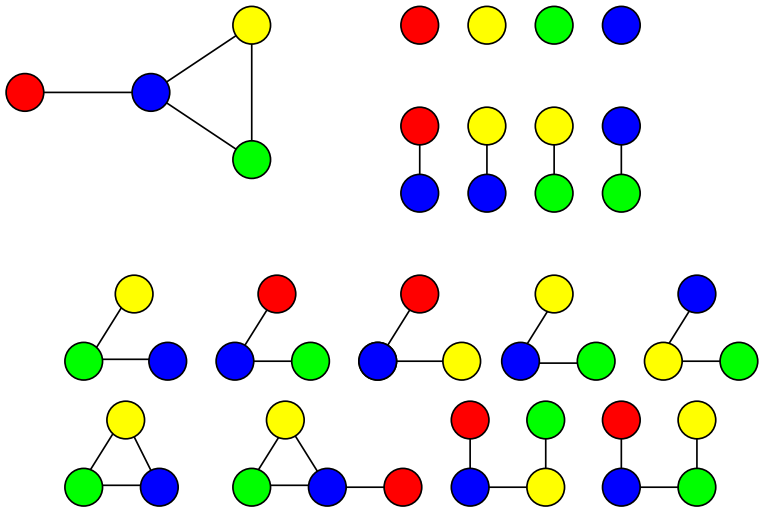
- Molecules, protein folds
- Protein interaction networks
- Metabolic pathways



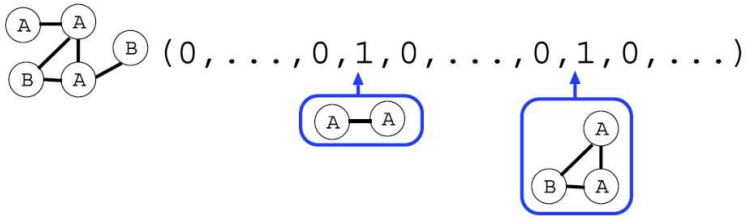
- Sequences can also be seen as linear graphs.
- How to compare them?

Kernels for Graphs

- Natural idea: decompose graphs into sub-graphs...



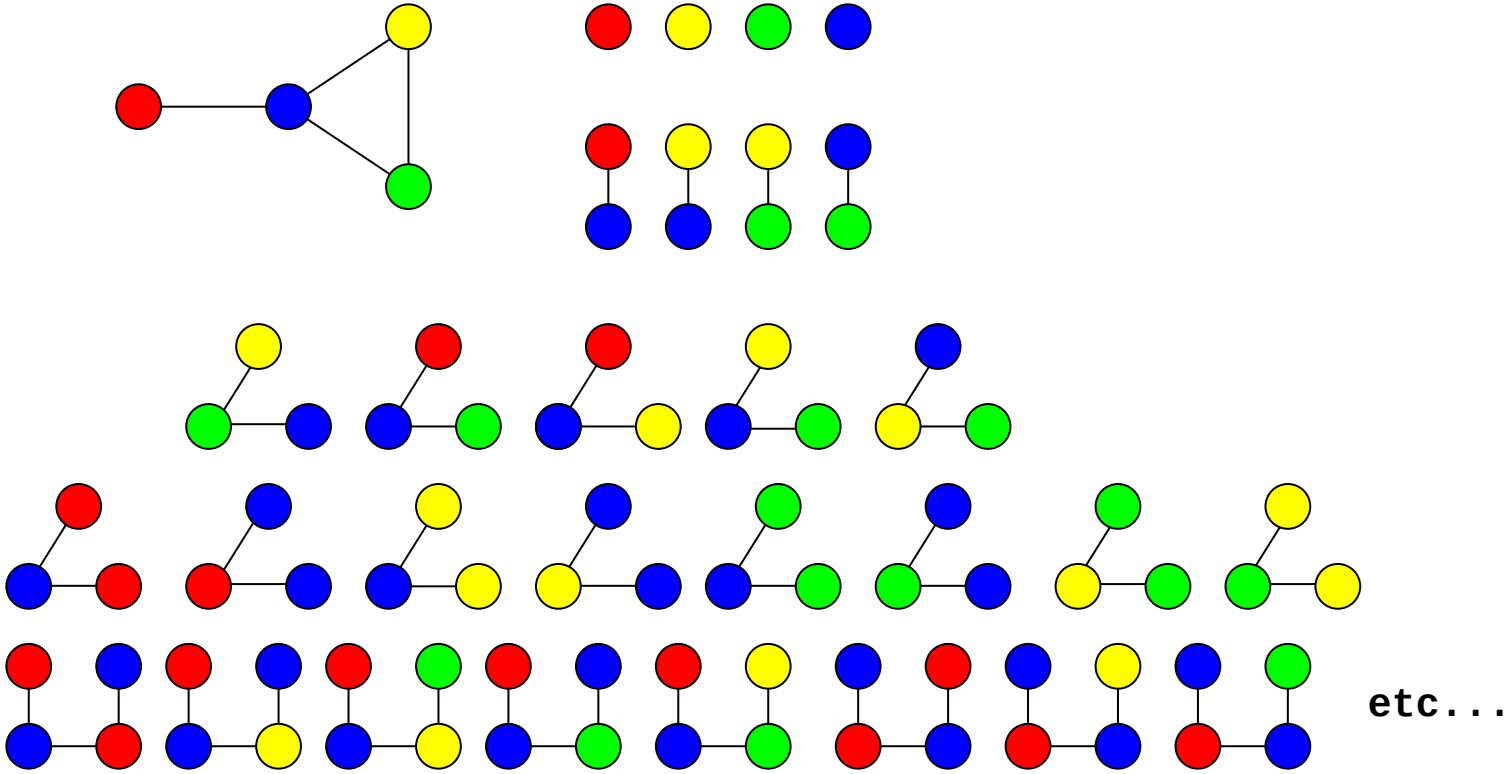
- ... and use histograms of sub-graphs?



- Problem: this approach is not feasible because it is not **tractable**.
- **Combinatorial limits**: Computing these histograms is a NP-hard problem.

Kernels for Graphs

- Instead, previous contributions [5, 9, 14] have considered walks in the graph



- Graph $\longrightarrow$ {set of walks} $\longrightarrow$ {sequences (of vertex labels, edge labels $etc.$)}.

Kernels for Graphs

- **Idea**: Use these large sets of sequences to compare two graphs:
- First, for any **arbitrary** subsequence s, count how many times it appears in *any* walk w of a graph G, weighted by a **weight** on the walk $\lambda(w)$.

$$\phi_s(G) = \sum_{w \text{ walk in } G} \lambda(w) 1_{(s \text{ in } w)}$$

- Compare two graphs by taking the dot-product of ϕ's:

$$k(G, G') = \sum_{\text{all sequences } s \text{ in } \mathcal{S}} \phi_s(G)\phi_s(G')$$

For some settings (λ, $\mathcal{S}$, restricted walks)
k can be computed in **polynomial time**,
without having to compute the ϕ vectors.

References

[1] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th annual ACM workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273, 1995.

[3] Marco Cuturi and Jean-Philippe Vert. The context-tree kernel for strings. *Neural Networks*, 18(8), 2005.

[4] T. Hastie, R. Tibshirani, and J. Friedman. *Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd edition)*. Springer Verlag, 2009.

[5] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In T. Faucett and N. Mishra, editors, *Proceedings of the Twentieth International Conference on Machine Learning*, pages 321–328. AAAI Press, 2003.

[6] C. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *The Journal of Machine Learning Research*, 5:1435–1455, 2004.

[7] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: a string kernel for svm protein classific ation. In *Proc. of PSB 2002*, pages 564–575, 2002.

[8] Christina Leslie, Eleazar Eskin, Jason Weston, and William Stafford Noble. Mismatch string kernels for svm protein classification. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *NIPS 15*. MIT Press, 2003.

[9] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Extensions of marginalized graph kernels. In R. Greiner and D. Schuurmans, editors, *Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2004)*, pages 552–559. ACM Press, 2004.

[10] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization , Optimization, and Beyond*. MIT Press, 2002.

[11] Bernhard Schölkopf, Koji Tsuda, and Jean-Philippe Vert. *Kernel Methods in Computational Biology*. MIT Press, 2004.

[12] S. Sonnenburg, G. Rätsch, and B. Schölkopf. Large scale genomic sequence SVM classifiers. In *Proceedings of the 22nd international conference on Machine learning*, pages 848–855. ACM, 2005.

[13] Jean-Philippe Vert and Yoshihiro Yamanishi. Supervised graph inference. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, 2005.

[14] SVN Vishwanathan, K.M. Borgwardt, I.R. Kondor, and N.N. Schraudolph. Graph kernels. *Journal of Machine Learning Research*, 9:1–37, 2008.