Support Vector Machines

Machine Learning course

Amaury Habrard & Marc Sebban

{amaury.habrard,marc.sebban}@univ-st-etienne.fr

Laboratoire Hubert Curien, UMR CNRS 5516
Université Jean Monnet Saint-Étienne

Semester 2

SVM in a few words

- Support Vector Machines (SVM): an important breakthrough in Machine Learning (mid 90's).
  - **Efficient** learning, in a **potentially infinite** feature space.
  - Strong **theoretical properties**.
  - Broad **applicability** (e.g., structured data).
  - Very **successful in practice** for many real-world problems.
- Gave rise to many other methods.
- Still a hot area of research.
- Used extensively by the Machine Learning, Data Mining and Computer Vision department here at the lab.

Materials

- Slides and materials available online.
- Software: LibSVM library, Scikit learn.
- References:
  - C. Cortes and V. Vapnik. Support Vector Networks. *Machine Learning*, 20, pp. 1–25. 1995.
  - V. Vapnik. Statistical Learning Theory. *Wiley-Interscience Publication*. 1998.
  - C. Burges. A tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2), pp. 955–974. 1998.
  - N. Cristianini and J. Shawe-Taylor. An introduction to Support Vector Machines and Others Kernel-Based Learning Methods. *Cambridge University Press*. 2000.
  - B. Scholkopf and A. Smola. Learning with Kernels, Support Vector Machines, Regularization, Optimization and Beyond. *MIT University Press*. 2002.
  - In general any PR/ML book has at least an introduction.

Acknowledgments: Aurélien Bellet, Liva Ralaivola.

Introduction

Classification

- **Classification as a real-world task:**
  - Does some patient have a serious disease?
  - What kind of secondary structure does a sequence of amino acids correspond to?
  - Is some molecule toxic for some organism?
  - Does this image represent a face?
  - Is this email a spam?
  - ...
- **Automating the classification task:**
  - Dealing with large number of objects to classify.
  - Faster than manual classification.
  - Less costly.

Supervised classification: notations

- We are given a **training set** T of n labeled instances

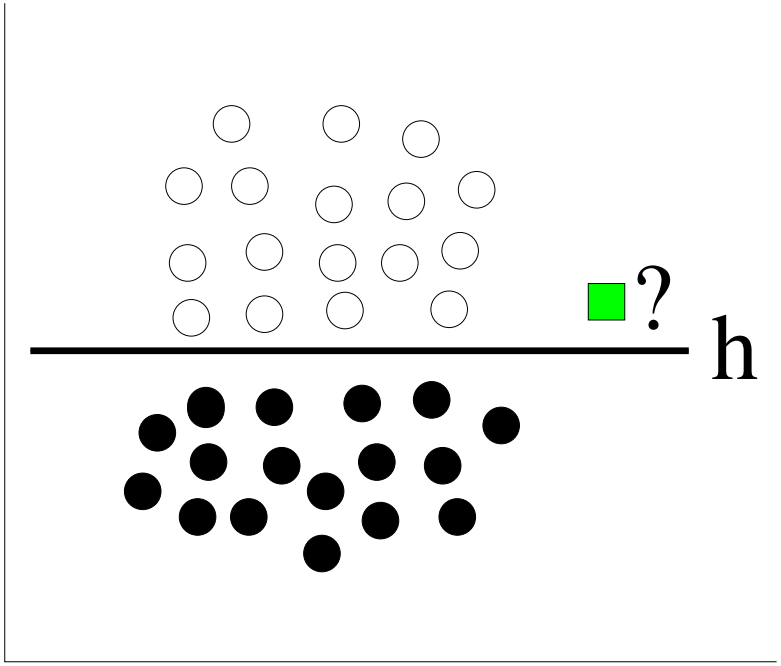
$$T = \{\mathbf{z_i} = (\mathbf{x_i}, y_i)\}_{i=1}^{n},$$

  where $\mathbf{z_i}$'s follow some distribution P, $\mathbf{x_i}$'s lie in some **feature space** $\mathcal{X}$ (e.g., $\mathbb{R}^d$) and y_i's are the labels.
- We focus here on **binary classification**: $\forall i, y_i \in \{-1, 1\}$.
- Examples:
  - $\mathbf{x_i}$ is the data about patient i (weight, blood pressure, age, etc), y_i is whether it has a serious disease.
  - $\mathbf{x_i}$ is the representation of image i, y_i is whether it contains a face.
  - $\mathbf{x_i}$ represents the words that appear in email i, y_i is whether it is a spam.

Supervised classification: goal

Goal

Using T, learn a classifier $h : \mathcal{X} \to \{-1, 1\}$ that, given a new instance drawn from P, is able to predict its label (with low error).



Supervised classification: strategy

- Simplified strategy: pick a class $\mathcal{C}$ of classifiers (e.g., linear separators) and find $h \in \mathcal{C}$ that minimizes the **empirical error** of h on T:

$$\hat{E}_T(h) = \frac{1}{n} \sum_{i=1}^{n} I[h(\mathbf{x_i}) \neq y_i].$$

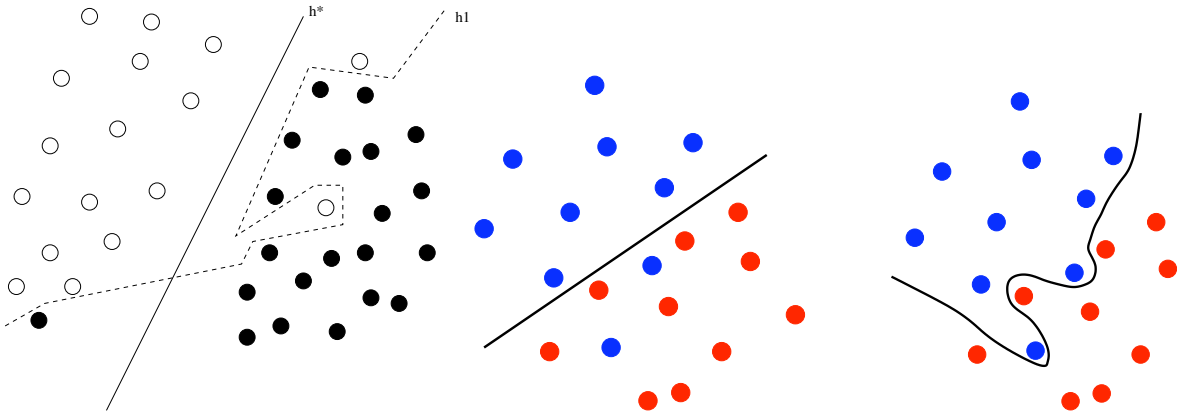
- We hope to get a low **generalization error**:

$$E(h) = \mathbb{E}_{z=(\mathbf{x},y)\sim P} I[h(\mathbf{x}) \neq y].$$

We can't always trust $\hat{E}_T(h)$!

- Usually have access to a **limited** number of training data.
- Therefore, can find a **complex** h which leads to "learning almost by heart" situation ($\hat{E}_T(h) \simeq 0$).
- Often results in high generalization error: $\hat{E}_T(h)$ is too optimistic. This is called **overfitting**.

Overfitting



Question

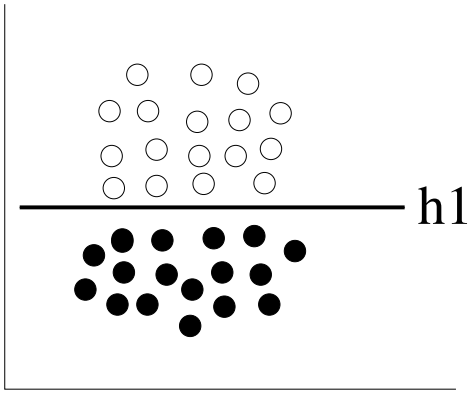
Is h_2 reasonable?

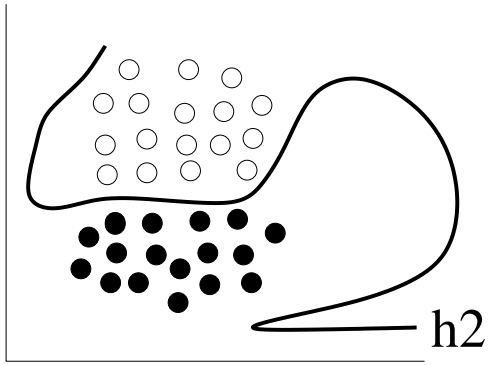
We can bound the generalization error $E(h)$ of a classifier h as follows:

$$E(h) < \hat{E}_T(h) + \mathcal{O}\left(\frac{\text{complexity}(h)}{\sqrt{|T|}}\right)$$

where complexity(h) is a measure of the capacity/complexity of the hypothesis h or the class of hypothesis considered by the algorithm.

Occam's razor



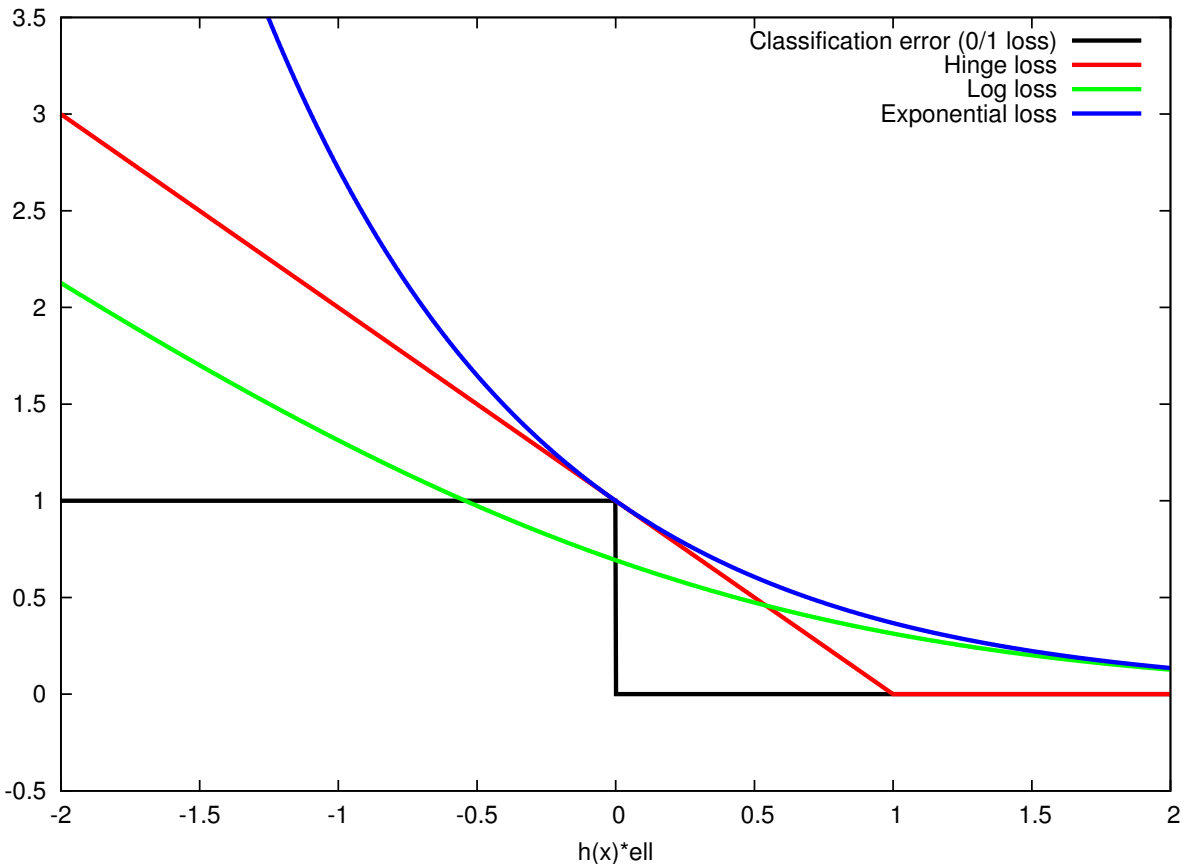


Occam's razor principle

- "Choose the simplest explanation consistent with past data".
- In supervised classification: **tradeoff between complexity** of h **and empirical error** of h.
- The choice of the class of classifiers $\mathcal{C}$ is also important.

Convex loss functions

- Recall that the empirical error (0/1 loss) is not convex.
- In practice, use a **convex loss** as surrogate function for the 0/1 loss.



Linear learning machines

Linear classifiers

- Assume that the feature space is $\mathcal{X} = \mathbb{R}^d$, i.e., objects are represented as d-dimensional real-valued vectors.
- We pick $\mathcal{C}$ to be the class of **linear classifiers** in $\mathbb{R}^d$.
- Therefore $h \in \mathcal{C}$ is of the form:

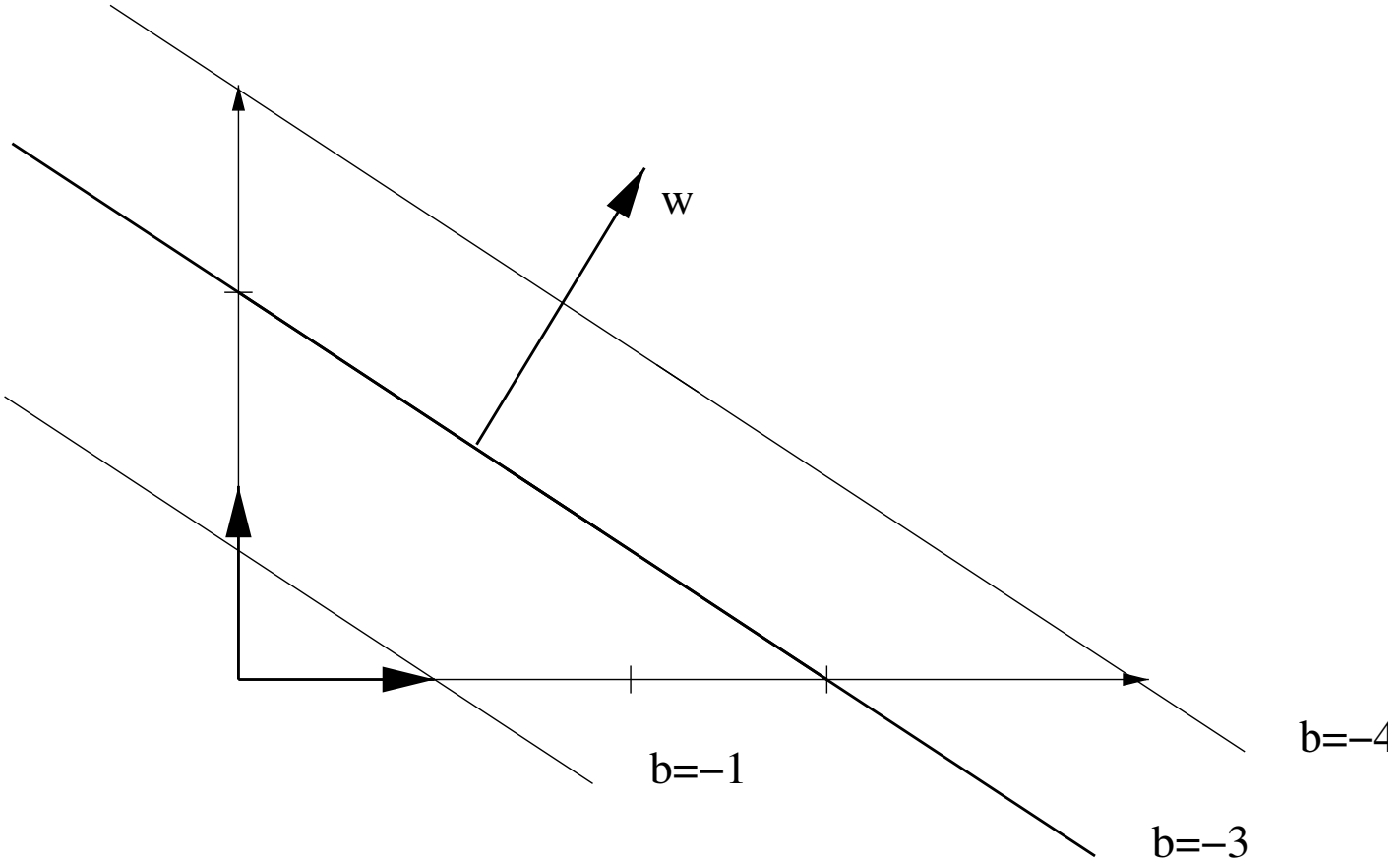
$$\begin{aligned} h & : & \mathbb{R}^d \to \{-1, 1\} \\ h(\mathbf{x}) & = & \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \\ & = & \text{sign}\left(\sum_{i=1}^{d} w_i x_i + b \right) \end{aligned}$$

- $\langle \cdot \rangle$ is the **inner product**.
- $\mathbf{w}$ is the **weight vector** and b the **bias**.

(Note: b can be removed by adding 1 as a new feature to all instances $\mathbf{x}$: $(x_1, \ldots, x_d, 1)$)

Linear classifiers: illustration

The classifier h defines a hyperplane (subspace of dimension $d-1$) of equation $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ which separates $\mathbb{R}^d$ into two regions. The figure shows h for $\mathbf{w} = (1, 3/2)^T$ and different values of b.



Perceptron

- We have $d + 1$ values to learn ($\mathbf{w}$ and b).
- A famous algorithm: Perceptron (Rosenblatt, 1960). Can be seen as the most simple kind of neural network.

Perceptron algorithm

Input: Training set T, learning rate $\eta > 0$
begin
  $k \leftarrow 0$, $w_k \leftarrow \vec{\mathbf{0}}$, $b_k \leftarrow 0$, $R \leftarrow \max_{i=1}^{n} \|\mathbf{x_i}\|_2$
  repeat
    for $i \leftarrow 1$ **to** n **do**
      if $y_i(\langle \mathbf{w_k}, \mathbf{x_i} \rangle + b_k) \leq 0$ **then**
        $\mathbf{w_{k+1}} \leftarrow \mathbf{w_k} + \eta y_i \mathbf{x_i}$
        $b_{k+1} \leftarrow b_k + \eta y_i R^2$
        $k \leftarrow k + 1$
  until no mistake is made
  return k, w_k, b_k

Exercise

- Let $\mathcal{X} = \mathbb{R}^2$.
- Assume at step k we have $\mathbf{w_k} = (1\ \text{-}1)^t$ and $b_k = 1$, (^{t} denotes the vector transpose notation).
- Consider the new example $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i = (3\ 2)^t$ and $y_i = -1$
- take $\eta = 0.1$ and $R = 5$ what is the new hyperplane?

Exercise

- Let $\mathcal{X} = \mathbb{R}^2$.
- Assume at step k we have $\mathbf{w_k} = (1\ \text{-}1)^t$ and $b_k = 1$
- Consider the new example $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i = (3\ 2)^t$ and $y_i = -1$
- take $\eta = 0.1$ and $R = 5$ what is the new hyperplane?

- we have $y_i(\langle \mathbf{w_k}, \mathbf{x_i} \rangle + b_k) = -1(1 \times 3 - 1 \times 2 + 1) = -2 < 0$: error, so the classifier must be updated

Exercise

- Let $\mathcal{X} = \mathbb{R}^2$.
- Assume at step k we have $\mathbf{w_k} = (1\ \text{-}1)^t$ and $b_k = 1$
- Consider the new example $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i = (3\ 2)^t$ and $y_i = -1$
- take $\eta = 0.1$ and $R = 5$ what is the new hyperplane?

- we have $y_i(\langle \mathbf{w_k}, \mathbf{x_i} \rangle + b_k) = -1(1 \times 3 - 1 \times 2 + 1) = -2 < 0$: error, so the classifier must be updated
- $\mathbf{w_{k+1}} = \mathbf{w_k} + \eta y_i \mathbf{x}_i = (1\ \text{-}1)^t + 0.1 \times (-1) \times (3\ 2)^t = (0.7\ \text{-}1.2)^t$

Exercise

- Let $\mathcal{X} = \mathbb{R}^2$.
- Assume at step k we have $\mathbf{w_k} = (1\ \text{-}1)^t$ and $b_k = 1$
- Consider the new example $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i = (3\ 2)^t$ and $y_i = -1$
- take $\eta = 0.1$ and $R = 5$ what is the new hyperplane?

- we have $y_i(\langle \mathbf{w_k}, \mathbf{x_i} \rangle + b_k) = -1(1 \times 3 - 1 \times 2 + 1) = -2 < 0$: error, so the classifier must be updated
- $\mathbf{w_{k+1}} = \mathbf{w_k} + \eta y_i \mathbf{x}_i = (1\ \text{-}1)^t + 0.1 \times (-1) \times (3\ 2)^t = (0.7\ \text{-}1.2)^t$
- $b_{k+1} = b_k + \eta y_i R^2 = 1 + 0.1 \times (-1) \times *5^2 = -1.5$

Exercise

- Let $\mathcal{X} = \mathbb{R}^2$.
- Assume at step k we have $\mathbf{w_k} = (1\ \text{-}1)^t$ and $b_k = 1$
- Consider the new example $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i = (3\ 2)^t$ and $y_i = -1$
- take $\eta = 0.1$ and $R = 5$ what is the new hyperplane?

- we have $y_i(\langle \mathbf{w_k}, \mathbf{x_i} \rangle + b_k) = -1(1 \times 3 - 1 \times 2 + 1) = -2 < 0$: error, so the classifier must be updated
- $\mathbf{w_{k+1}} = \mathbf{w_k} + \eta y_i \mathbf{x}_i = (1\ \text{-}1)^t + 0.1 \times (-1) \times (3\ 2)^t = (0.7\ \text{-}1.2)^t$
- $b_{k+1} = b_k + \eta y_i R^2 = 1 + 0.1 \times (-1) \times *5^2 = -1.5$
- $y_i(\langle \mathbf{w_{k+1}}, \mathbf{x_i} \rangle + b_{k+1}) = -1(0.7 * 3 - 1.2 * 2\ -1.5) = -1 \times -1.8 > 0$: correct classification

Margin of an instance

- The weight vector and the bias are updated iteratively.
- The algorithm converges if the data is linearly separable.
- The number of iterations depends on the **margin**.

Definition (Margin of an instance)

The **margin** γ_i of the instance $\mathbf{z_i} = (\mathbf{x_i}, y_i)$ with respect to a hyperplane $(\mathbf{w}, b)$ is defined by:

$$\gamma_i = y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b).$$

Remark

If $\gamma_i > 0$, then $\mathbf{z_i}$ is correctly classified by $(\mathbf{w}, b)$.

Remark 2

The distance of any $\mathbf{x_0}$ to a hyperplane s.t. $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ is $\frac{|\langle \mathbf{w}, \mathbf{x_0} \rangle + b|}{\|w\|}$

Margin of a hyperplane

Definition (Margin of a hyperplane)

The **margin** γ of a hyperplane $(\mathbf{w}, b)$ with respect to a set of example $T = \{\mathbf{z_i} = (\mathbf{x_i}, y_i)\}_{i=1}^{n}$ is defined as:

$$\gamma = \min_{1 \leq i \leq n} \gamma_i = \min_{1 \leq i \leq n} y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b).$$

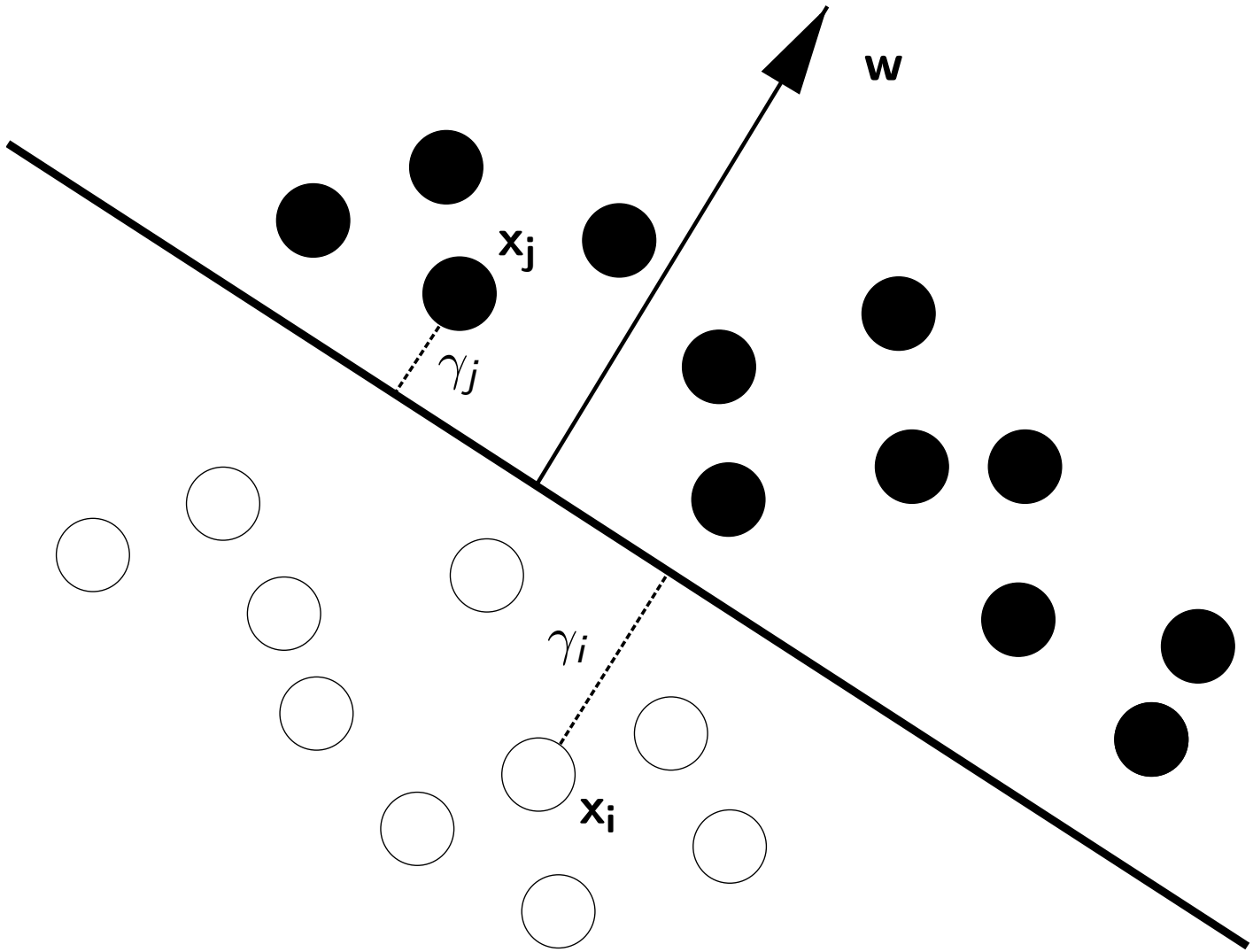
Remark

If we consider the **normalized** hyperplane

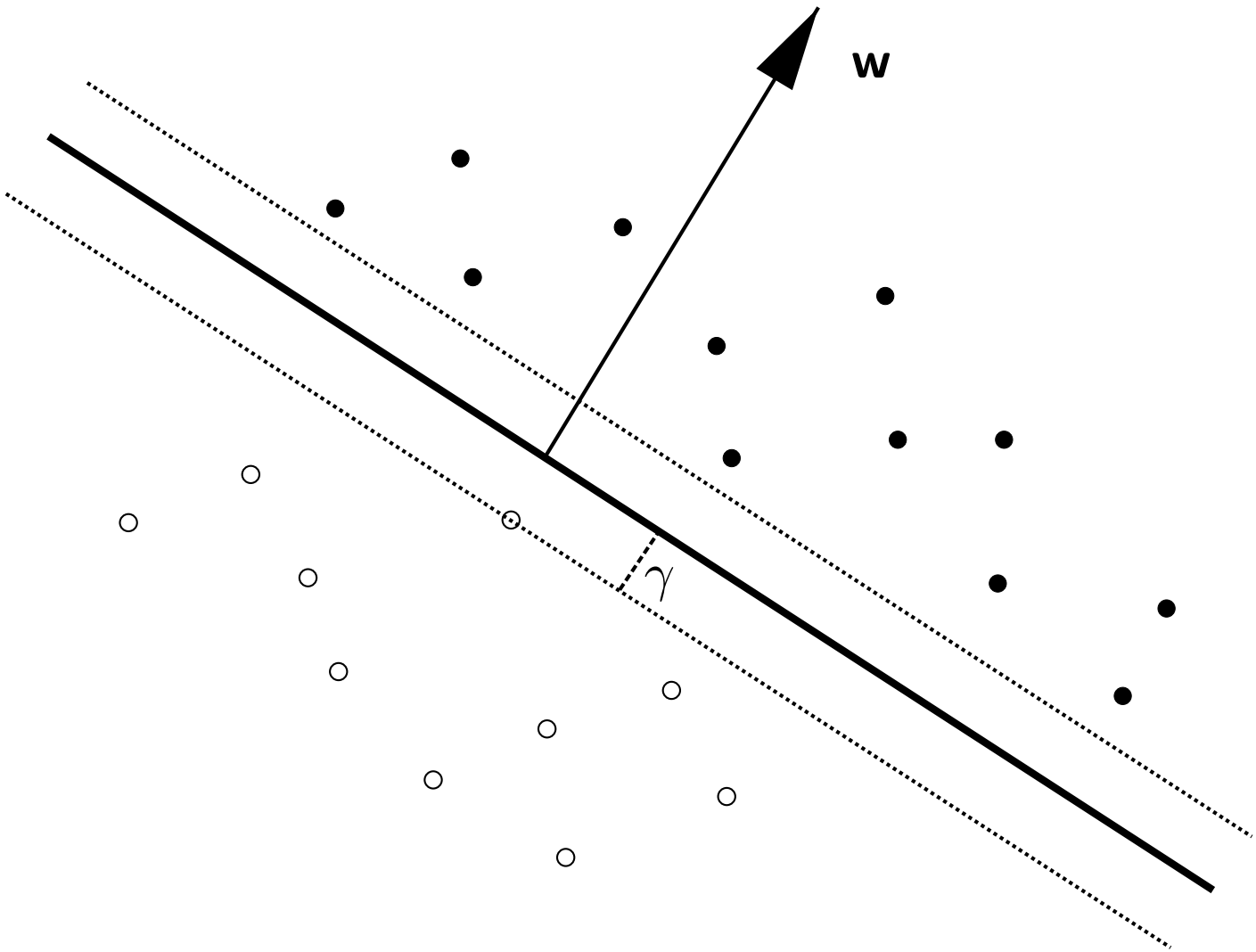
$$\left(\frac{1}{\|\mathbf{w}\|_2} \mathbf{w}, \frac{1}{\|\mathbf{w}\|_2} b \right),$$

then the definitions of margin refer to the **geometric margin** (Euclidean distance between the instances and the hyperplane).

Margin of an instance: illustration



Margin of a hyperplane: illustration



Convergence of Perceptron

Theorem (Convergence of the Perceptron algorithm)

Let $T = \{\mathbf{z_i} = (\mathbf{x_i}, y_i)\}_{i=1}^{n}$ *be the training sample and let* $R = \max_{1 \leq i \leq n} \|\mathbf{x_i}\|_2$. *If there exists a hyperplane* $(\mathbf{w}^*, b^*)$ *so that*

$$\|\mathbf{w}^*\| = 1,$$
$$y_i(\langle \mathbf{w}^*, \mathbf{x_i} \rangle + b^*) \geq \gamma, \quad 1 \leq i \leq n,$$

then the number of mistakes of the Perceptron algorithm is at most $\left(\frac{2R}{\gamma}\right)^2$.

Hyperplane and linear combination

Remark (important)

The Perceptron algorithm builds $\mathbf{w}$ by iteratively adding misclassified positive instances to or subtracting misclassified negative instances from the initial vector $\mathbf{w_0} = \vec{\mathbf{0}}$. Therefore, the hyperplane returned by the algorithm is a **linear combination of the training instances**:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x_i}.$$

- α_i is positive and proportional to the number of times the (misclassified) example $\mathbf{x_i}$ triggered the update of $\mathbf{w}$.
- Therefore, "difficult" examples have a high α_i.

Perceptron dual form

- α can be seen as a representation of the hyperplane in a different space (dual form).
- h can be rewritten in the dual space:

$$\begin{aligned} h(x) &= \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \\ &= \text{sign}\left(\left(\sum_{i=1}^{n} \langle \alpha_i y_i \mathbf{x_i}, \mathbf{x} \rangle\right) + b\right) \\ &= \text{sign}\left(\left(\sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{x_i}, \mathbf{x} \rangle\right) + b\right). \end{aligned}$$

Dual Perceptron algorithm

We can also express the learning algorithm in dual form (i.e., learn $\boldsymbol{\alpha}$ directly).

Dual Perceptron algorithm

Input: Training set T
begin
  $k \leftarrow 0$, $\boldsymbol{\alpha} \leftarrow \vec{\mathbf{0}}$, $b \leftarrow 0$, $R \leftarrow \max_{i=1}^{n} \|\mathbf{x_i}\|_2$
  repeat
    for $i \leftarrow 1$ **to** n **do**
      if $y_i \left(\sum_{j=1}^{n} \alpha_j y_j \left\langle \mathbf{x_j}, \mathbf{x_i} \right\rangle + b \right) \leq 0$ **then**
        $\alpha_i \leftarrow \alpha_i + 1$
        $b \leftarrow b + y_i R^2$
  until no mistake is made
  return $\boldsymbol{\alpha}$, b

Gram matrix

Remark (important)

In the dual version of the algorithm, the interface to training data (the $\mathbf{x_i}$'s) is limited to inner products between instances. Therefore, to learn the hyperplane, we only need the **Gram matrix** G, whose components are the inner products between all pairs of training instances:

$$G_{ij} = \langle \mathbf{x_i}, \mathbf{x_j} \rangle .$$

Summary

Summary

1. The Perceptron algorithm as a way to learn a linear classifier.
2. The dual representation of the problem allows us to learn the classifier in another representation space, where we only need inner products between training instances (you will soon understand why this is great).

Limitations of Perceptron

1. The solution is not unique (sensitive to the order of the $\mathbf{x_i}$'s). We will see that this can be fixed by selecting the hyperplane that **maximizes the margin**.
2. If the data is not linearly separable, we're stuck (the algorithm diverges). We will see that this can be fixed by using a **kernel function**.

Feature space and kernel trick

Dealing with the nonlinearly separable setting

- In this section, we deal with the key limitation of linear learning machines: they can't deal with the nonlinearly separable setting.
- A solution could be to **work with a more complex class of classifiers** $\mathcal{C}$.
- But this means **giving up the simplicity of linear classifiers**:
  - Learning them is easy and efficient.
  - So is prediction.
  - They are robust to overfitting.
- Also means giving up the notion of margin and the promising dual form where we learn using inner products only.

SVM solution

Keep the classifier linear but learn it in a **better feature space**!

A graphical Illustration

PolynomialKernelExample

https://www.youtube.com/watch?v=9NrALgHFwTo

Another one

https://www.youtube.com/watch?v=3liCbRZPrZA

Better feature space: example

- Newton's law of universal gravitation:

$$f(m_1, m_2, r) = G\frac{m_1 m_2}{r^2},$$

where m_1, m_2 are the masses, r the distance between the center of the masses and G the gravitational constant.

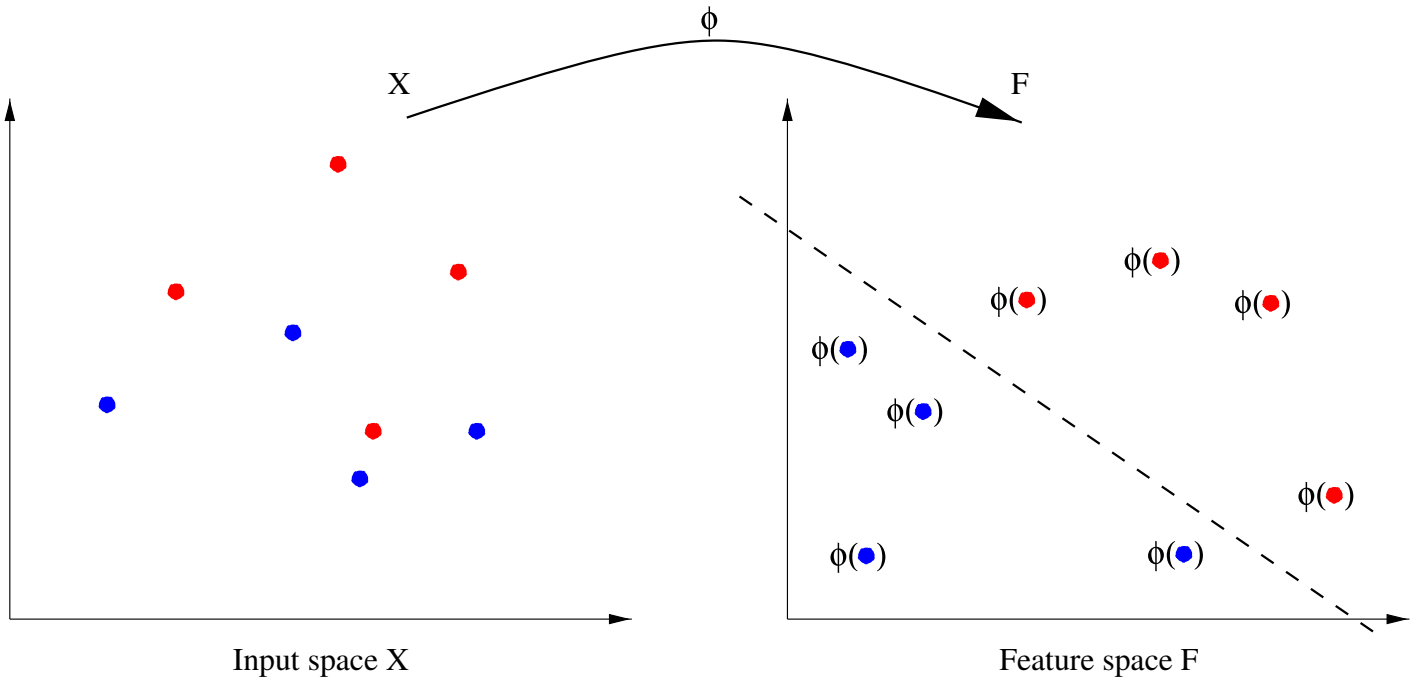
- This fonction is not linear: it can't be learned by a linear learning machine.
- Change of coordinates using nonlinear transformation Φ:

$$\Phi(m_1, m_2, r) = (\log m_1, \log m_2, \log r) = (x, y, z).$$

$$\begin{aligned} f(\Phi(m_1, m_2, r)) &= \log G + \log m_1 + \log m_2 - 2\log r \\ &= c + x + y - 2z, \end{aligned}$$

which is a linear fonction of x, y, z and is therefore learnable. (We can learn a linear classifier to decide if a mass will get far away from another)

Better feature space: illustration



Two-step strategy:

1. use a (nonlinear) transformation $\Phi : \mathcal{X} \rightarrow \mathcal{F}$ to project the data in a new feature space $\mathcal{F}$,
2. learn a linear separator in $\mathcal{F}$.

Better feature space: another illustration

The linear classifier learned in the nonlinear space $\mathcal{F}$ induced by Φ:

$$\begin{aligned} h(x) &= \text{sign}\left(\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b\right) \\ &= \text{sign}\left(\sum_{i=1}^{|\mathcal{F}|} w_i \Phi_i(\mathbf{x}) + b\right), \end{aligned}$$

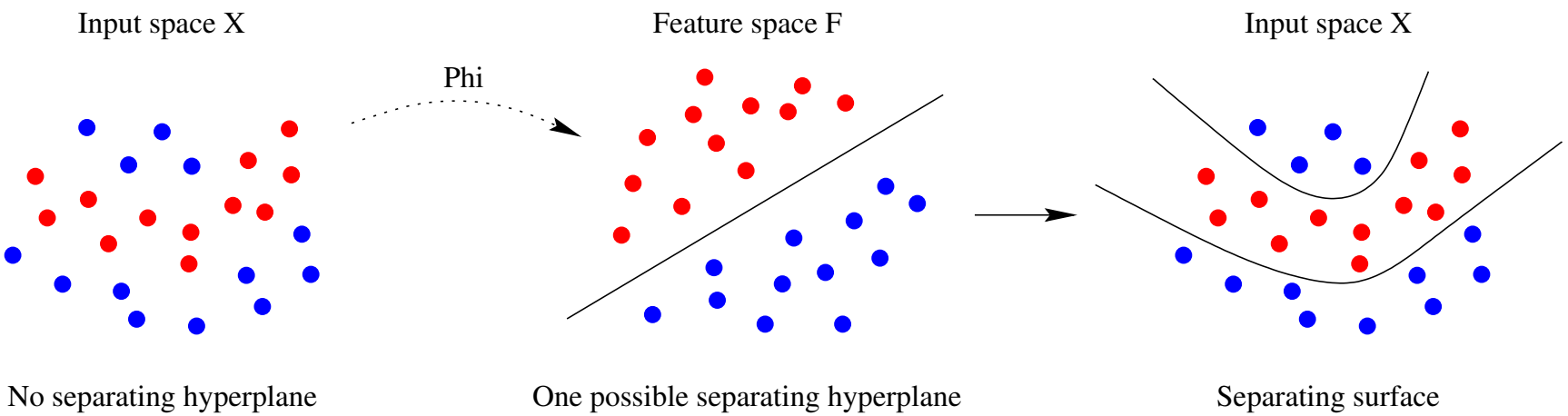
corresponds to a **nonlinear classifier in the input space** $\mathcal{X}$.

Better feature space: another illustration

The linear classifier learned in the nonlinear space $\mathcal{F}$ induced by Φ:

$$\begin{aligned} h(x) &= \text{sign}\left(\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle + b\right) \\ &= \text{sign}\left(\sum_{i=1}^{|\mathcal{F}|} w_i \Phi_i(\mathbf{x}) + b\right), \end{aligned}$$

corresponds to a **nonlinear classifier in the input space** $\mathcal{X}$.



Increasing the dimensionality

- Sometimes, this is not enough: must project the data to a **higher dimensional space**.
- For instance, suppose that background knowledge on our problem can be perfectly learned by using monomials of degree 2.
- Then this is a good idea to work in $\mathbb{R}^3$ instead of $\mathbb{R}^2$ by using the transformation:

$$\Phi(x_1, x_2) = (x_1^2, x_2^2, x_1 x_2).$$

- What if we have more input dimensions and/or need higher order monomials?

Curse of dimensionality

Curse of dimensionality

If the dimensionality gets too high, **projecting and learning becomes intractable**. For instance, for monomials of degree deg, must consider a space of dimension

$$\begin{pmatrix} d + deg - 1 \\ deg \end{pmatrix}$$

which is way too large even for relatively small values of d and deg.

Example

- $d = 5$, $deg = 5$: need 126 dimensions.
- $d = 10$, $deg = 5$: need 11628 dimensions.
- $d = 20$, $deg = 10$: need 20030010 dimensions.
- ...

Back to the dual form of Perceptron

- Recall the dual form of the Perceptron algorithm where interface to training data is limited to inner products.
- If we use a transformation $\Phi : \mathcal{X} \to \mathcal{F}$,

$$\text{replace} \quad \langle \mathbf{x_i}, \mathbf{x_j} \rangle \quad \text{by} \quad \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle$$

  to learn the linear separator in $\mathcal{F}$.
- If we had a way to compute $\langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle$ directly (without having to apply Φ), could **learn efficiently in one step**!
- Turns out that we can, using the notion of **kernel function**.

Kernel function

Definition (Kernel function)

A function $K : \mathcal{X} \times \mathcal{X} \to [-1, 1]$ is a (Mercer) **kernel** if there exists a function $\Phi : \mathcal{X} \to \mathcal{F}$ such that $K(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$. Equivalently, we say that K is a kernel if it is PSD.

PSD matrix and PSD function

These are two different concepts, although deeply linked. For instance, a symmetric function K is PSD iff any matrix $\mathbf{K} = (K(x_i, x_j))_{(i,j)=(1,1)}^{(n,n)}$ is PSD (Mercer's property). Another example: a matrix $\mathbf{M}$ is PSD iff it arises as a Gram (kernel) matrix of some set of vectors.

Remark

The definition of kernel implies that K is **symmetric**, i.e., $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$,

$$K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x}).$$

Kernel construction

Theorem (Construction of kernels from other kernels)

Let $\mathcal{X} \subseteq \mathbb{R}^d$, $\Phi : \mathcal{X} \to \mathbb{R}^m$, $f : \mathcal{X} \to \mathbb{R}$, $p \in \mathbb{N}$ and $a \in \mathbb{R}_+$. Let K_1 and K_2 be two kernels on $\mathcal{X} \times \mathcal{X}$, and K_3 be a kernel on $\mathbb{R}^m \times \mathbb{R}^m$. Let B a PSD matrix. Then the following functions are kernels:

$$
\begin{aligned}
K(\mathbf{x}, \mathbf{x}') &= K_1(\mathbf{x}, \mathbf{x}') + K_2(\mathbf{x}, \mathbf{x}') \\
K(\mathbf{x}, \mathbf{x}') &= aK_1(\mathbf{x}, \mathbf{x}') \\
K(\mathbf{x}, \mathbf{x}') &= K_1(\mathbf{x}, \mathbf{x}')K_2(\mathbf{x}, \mathbf{x}') \\
K(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x})f(\mathbf{x}') \\
K(\mathbf{x}, \mathbf{x}') &= K_3(\Phi(\mathbf{x}), \Phi(\mathbf{x}')) \\
K(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^T B \mathbf{x}'
\end{aligned}
$$

Kernel functions: examples

- "Explicit" kernels:
  - Linear kernel: $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$. Equivalent to no kernel.
  - Other kernels constructed from an explicit transformation Φ (e.g., many kernels for structured data). More on this later!
- Popular "implicit" kernels:
  - Polynomial kernels: $K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^{deg}$, with $c \in \mathbb{R}, deg \in \mathbb{N}$. Contains all monomials of degree up to deg, i.e., dimension of $\mathcal{F}$ is

$$\begin{pmatrix} deg + d \\ deg \end{pmatrix}$$

    Proof coming up.
  - Gaussian/RBF kernel: $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|_2^2}{2\sigma^2}\right)$, where $\sigma^2 > 0$ is the width parameter. Most widely-used kernel. Corresponding $\mathcal{F}$ is of infinite dimension!
    Other usual notation: $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}'\|_2^2\right)$, $\gamma > 0$
  - Hyperbolic Tangent (Sigmoid) Kernel $K(\mathbf{x}, \mathbf{x}') = \tanh(\alpha \langle \mathbf{x}, \mathbf{x}' \rangle + c)$

Polynomial kernel

First take $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^2$.

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= \left(\sum_{i=1}^{d} x_i x_i' \right)^2 = \left(\sum_{i=1}^{d} x_i x_i' \right) \left(\sum_{j=1}^{d} x_j x_j' \right) \\ &= \sum_{(i,j)=(1,1)}^{(d,d)} x_i x_i' x_j x_j' = \sum_{(i,j)=(1,1)}^{(d,d)} (x_i x_j)(x_i' x_j') \end{aligned}$$

K is a kernel with $\Phi(\mathbf{x}) = (x_i x_j)_{(i,j)=(1,1)}^{(n,n)}$, a vector whose components are all the products $x_i x_j$ (**monomials of degree 2**). $\Phi(\mathbf{x}) \in \mathcal{F}$ is of dimension

$$\begin{pmatrix} d+1 \\ 2 \end{pmatrix}$$

Polynomial kernel ctd

Now take $K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^2$.

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= \left(\sum_{i=1}^{d} x_i x_i' + c \right) \left(\sum_{j=1}^{d} x_j x_j' + c \right) \\ &= c^2 + \sum_{(i,j)=(1,1)}^{(d,d)} x_i x_i' x_j x_j' + \sum_{i=1}^{d} \sqrt{2c} x_i \sqrt{2c} x_i' \end{aligned}$$

In this case, the features are **all the monomials of degree up to 2**, with weights controlled by c. The dimension of $\mathcal{F}$ is:

$$1 + \binom{d+1}{2} + d$$

Polynomial kernel ctd(2)

The previous reasoning can be generalized to any $deg \in \mathbb{N}$:

- $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^{deg}$: the features are the $\begin{pmatrix} deg + d - 1 \\ deg \end{pmatrix}$ monomials of degree deg.
- $K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^{deg}$: the $\begin{pmatrix} deg + d \\ deg \end{pmatrix}$ monomials of degree up to deg.

Remark

Can also prove that these are kernels by construction.

The kernel trick

- Using a kernel K, in the dual form of the Perceptron algorithm,

$$\text{replace} \quad \langle \mathbf{x_i}, \mathbf{x_j} \rangle \quad \text{by} \quad \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle = K(\mathbf{x_i}, \mathbf{x_j})$$

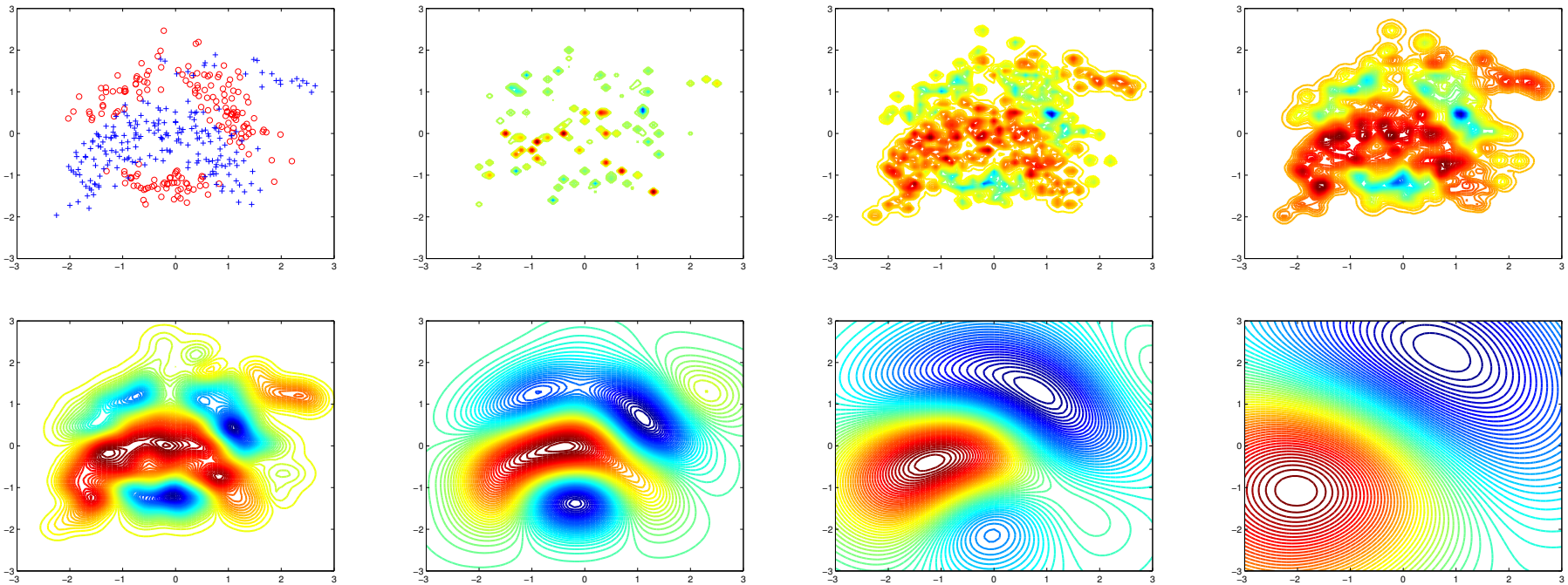
- Furthermore, the linear classifier in dual space can be rewritten as:

$$h(x) = \text{sign}\left(\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x_i}, \mathbf{x}) \right) + b \right).$$

- Several advantages:
  - No need to represent $\Phi(\mathbf{x}) \in \mathcal{F}$ explicitly.
  - The dimension of $\mathcal{F}$ does not affect the evaluation of h (we need at most n evaluations of the kernel).
  - The dimension of $\mathcal{F}$ does not affect the number of parameters to learn (we learn $n+1$ values).
  - In fact, we don't even need to know Φ! (can be implicit)

Kernel trick: illustration

- For the Gaussian kernel and $\sigma \in [0.01, 0.05, 0.1, 0.2, 0.5, 1, 2]$:



Can also check `http://svr-www.eng.cam.ac.uk/~kkc21/thesis_main/node31.html`.

Summary

Summary

- Changing the feature space as a way to deal with the nonlinearly separable setting.
- Need to use a **nonlinear** transformation as well as to **increase the dimensionality** of the feature space.
- The **kernel trick** allows us to do this in an efficient way.

Remaining issues

- What if the problem **remains nonlinearly separable**?
- We'd like to learn a **large margin** separator.
- We'd like to establish the **validity** of the technique (from a theoretical/statistical point of view).

Hard-margin Support Vector Machines

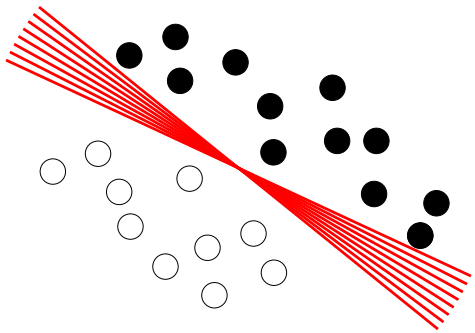
Hard-margin Support Vector Machines

Hard-margin Support Vector Machines:

- Learn a **linear** classifier.
- In the feature space induced by a **kernel** (using the kernel trick).
- Maximization of the **margin** of the hyperplane.
- Done by **convex optimization** (no local minimum).
- Takes advantage of **duality theory** (Lagrange).
- For now, we deal with the linearly-separable setting.

Maximizing the margin

- As we've seen with the Perceptron, there usually exists more than one hyperplane that separate the data.



- In fact, it's even "worse" since a separating hyperplane has one degree of freedom:

$$(\mathbf{w}, b) \equiv (\lambda\mathbf{w}, \lambda b), \lambda > 0.$$

Solution

Fix the (functional) margin to be 1 while **maximizing the geometric margin**.

Maximizing the margin ctd

We want $(\mathbf{w}, b)$ to achieve a functional margin of 1 on a positive instance $\mathbf{x}^+$ and a negative instance $\mathbf{x}^-$, i.e.,

$$\begin{cases} \langle \mathbf{w}, \mathbf{x}^+ \rangle + b &= 1 \\ -(\langle \mathbf{w}, \mathbf{x}^- \rangle + b) &= 1, \end{cases}$$

which is equivalent to

$$\begin{cases} \langle \mathbf{w}, \mathbf{x}^+ \rangle &= 1 - b \\ \langle \mathbf{w}, \mathbf{x}^- \rangle &= -b - 1. \end{cases}$$

Maximizing the margin ctd(2)

Then the geometric margin of $(\mathbf{w}, b)$ is:

$$\begin{cases} \gamma &= \frac{1}{\|\mathbf{w}\|_2} \langle \mathbf{w}, \mathbf{x}^+ \rangle + \frac{1}{\|\mathbf{w}\|_2} b \\ \gamma &= -\frac{1}{\|\mathbf{w}\|_2} \langle \mathbf{w}, \mathbf{x}^- \rangle - \frac{1}{\|\mathbf{w}\|_2} b \end{cases}$$

and thus

$$\begin{aligned} \gamma &= \frac{1}{2\|\mathbf{w}\|_2} (\langle \mathbf{w}, \mathbf{x}^+ \rangle - \langle \mathbf{w}, \mathbf{x}^- \rangle) \\ \gamma &= \frac{1}{2\|\mathbf{w}\|_2} (1 - b + 1 + b) \\ \gamma &= \frac{1}{\|\mathbf{w}\|_2}. \end{aligned}$$

SVM hard-margin primal

SVM hard-margin primal form

Given a training set $T = \{\mathbf{z_i} = (\mathbf{x_i}, y_i)\}_{i=1}^{n}$ of linearly-separable instances. The largest-margin hyperplane $(\mathbf{w}^*, b^*)$ separating the instances of T is the solution to the following optimization problem.

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1, \quad 1 \leq i \leq n \end{aligned}$$

Questions

1. Why do we minimize $\frac{1}{2}\|\mathbf{w}\|_2^2$?
2. Do we know how to solve this efficiently?

SVM hard-margin primal

SVM hard-margin primal form

Given a training set $T = \{\mathbf{z_i} = (\mathbf{x_i}, y_i)\}_{i=1}^{n}$ of linearly-separable instances. The largest-margin hyperplane $(\mathbf{w}^*, b^*)$ separating the instances of T is the solution of the following optimization problem.

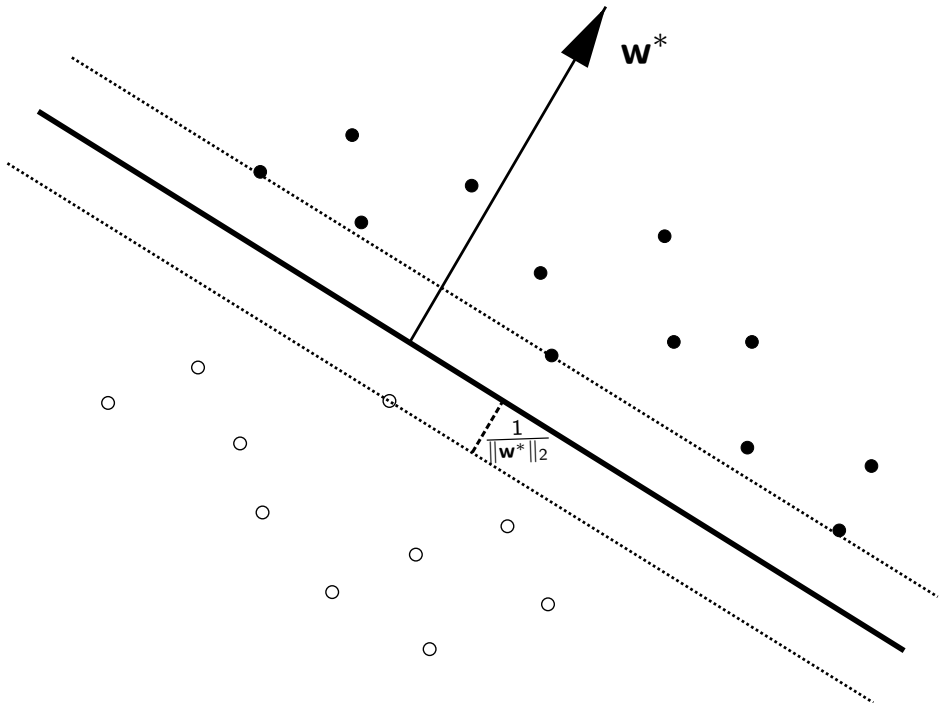
$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1, \quad 1 \leq i \leq n \end{aligned}$$

Answers

1. To maximize the margin and keep the objective convex. The factor $\frac{1}{2}$ is just for mathematical convenience (coming soon).
2. Yes, since it is a strictly convex problem. More precisely, it's a strictly convex Quadratic Program (QP).

Primal: summary

- Cool! Our formulation is strictly convex so we have **a unique solution**: the hyperplane that maximizes the geometric margin $\frac{1}{\|\mathbf{w}^*\|_2}$.



- Now we'd like to be able to use our amazing **kernel trick**...
- ...but this is not possible with this formulation!
- We can't apply the same reasoning as for the Perceptron algorithm.

Towards the dual

Solution

Strong duality holds, so use Lagrange duality and solve dual instead!

- The Lagrangian of the problem is:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{n} \alpha_i [1 - y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b)],$$

  where $\boldsymbol{\alpha} \succeq 0$.

- Recall that the Lagrange dual function $g(\boldsymbol{\alpha})$ is obtained by minimizing L over the variables of the primal problem ($\mathbf{w}$ and b). To do that we set the derivatives of L with respect to $\mathbf{w}$ and b to 0:

$$\begin{cases} \frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{n} y_i \alpha_i \mathbf{x_i} = 0 \\ \frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{b}} &= \sum_{i=1}^{n} y_i \alpha_i = 0 \end{cases}$$

Towards the dual ctd

- We thus have:

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x_i} \quad \text{and} \quad \sum_{i=1}^{n} y_i \alpha_i = 0.$$

- Substitute in L to get the dual function $g(\boldsymbol{\alpha})$ to maximize:

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{n} \alpha_i [1 - y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b)] \\ &= \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j} \rangle - \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j} \rangle - b \sum_{i=1}^{n} y_i \alpha_i + \sum_{i=1}^{n} \alpha_i \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j} \rangle \\ &= g(\boldsymbol{\alpha}). \end{aligned}$$

SVM hard-margin dual

SVM hard-margin dual form

$$\max_{\boldsymbol{\alpha}} \quad g(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j} \rangle$$

$$\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$\boldsymbol{\alpha} \succeq 0$$

Remarks

- This problem is convex (duality theory).
- Since strong duality holds, the optimal value of the dual is equal to the optimal value of the primal.
- The largest margin separating hyperplane can be recovered by taking $\mathbf{w}^* = \sum_{i=1}^{n} y_i \alpha_i^* \mathbf{x_i}$. What about b^*?

Understanding the optimal solution

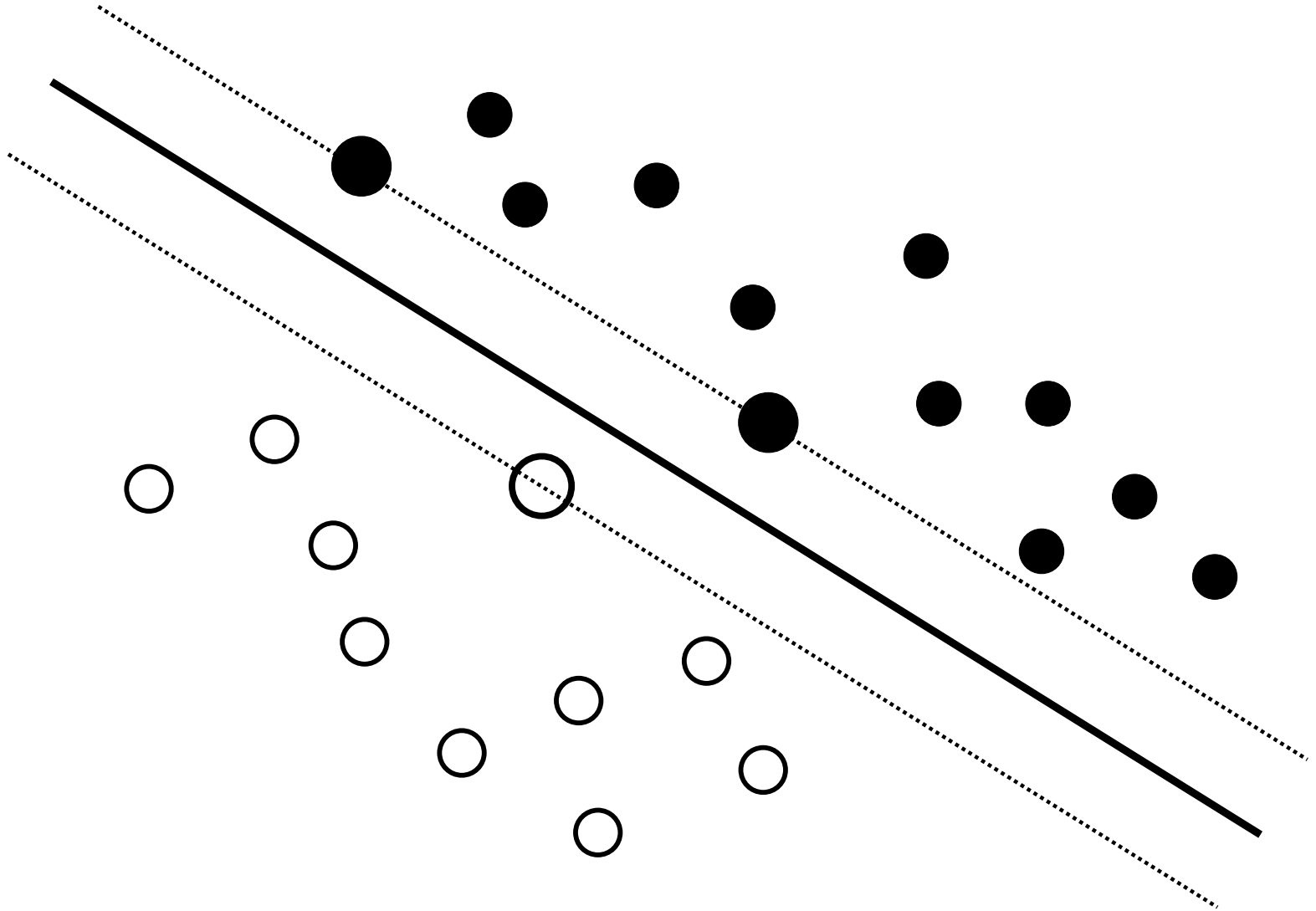
- To understand the optimal solution better and be able to compute b^*, we apply the **Karush-Kuhn-Tucker conditions**.
- Given a convex constrained optimization problem, the KKT conditions give **necessary and sufficient** conditions for a point to be optimal.
- For our problem, we get:

$$\alpha_i^*[y_i(\langle \mathbf{w}^*, \mathbf{x_i} \rangle + b^*) - 1] = 0, \quad 1 \leq i \leq n.$$

1. **Case 1**: $y_i(\langle \mathbf{w}^*, \mathbf{x_i} \rangle + b^*) > 1$. $\mathbf{x_i}$ is not on the margin and $\alpha_i^* = 0$.
2. **Case 2**: $y_i(\langle \mathbf{w}^*, \mathbf{x_i} \rangle + b^*) = 1$. $\mathbf{x_i}$ is on the margin and $\alpha_i^* \neq 0$. From these points we can deduce b^*.

- Therefore, $\mathbf{w}^*$ is defined only in terms of the training instances that lie on the margin. We call these points **support vectors**.

Support vectors: illustration



Half-way summary

- The dual algorithm is expressed in terms of inner products only.
- We characterized the set SV of **support vectors** on which the hyperplane is based.
- $\mathbf{w}^*$ can be rewritten as: $\mathbf{w}^* = \sum_{i=1}^{n} y_i \alpha_i^* \mathbf{x_i} = \sum_{i \in SV} y_i \alpha_i^* \mathbf{x_i}$.
- The linear classifier is now written as: $h(x) = \sum_{i \in SV} y_i \alpha_i^* \langle \mathbf{x_i}, \mathbf{x} \rangle + b^*$.
- Can use the **kernel trick**!
- Intuitively, α_i^* tells how important $\mathbf{x_i}$ is in the solution. Instances that are not support vectors ($\alpha_i^* = 0$) have no influence in the solution and can be ignored after learning!
- For this reason, we may say that SVM are **sparse**. (Note: replacing the $\|\cdot\|_2$-norm by the $\|\cdot\|_1$-norm will increase the sparsity $\rightarrow$ sparse SVM)

Kernelized SVM hard-margin dual

Kernelized SVM hard-margin dual form

$$\max_{\boldsymbol{\alpha}} \quad g(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(\mathbf{x_i}, \mathbf{x_j})$$

$$\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$\boldsymbol{\alpha} \succeq 0$$

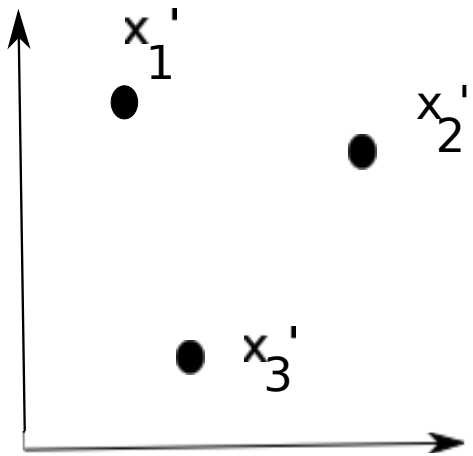
Kernelized linear classifier

Rewrite as $h(x) = \sum_{i \in SV} y_i \alpha_i^* K(\mathbf{x_i}, \mathbf{x}) + b^*$.

Illustration with the Gaussian kernel

$K(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\|\mathbf{x}-\mathbf{x}'\|}{\sigma^2})$

- 3 examples (seen as landmark points)
  $z_1 = (\mathbf{x}'_1, l_1)$, $z_2 = (\mathbf{x}'_2, l_2)$, $z_3 = (\mathbf{x}'_3, l_3)$
  $(l_1 = l_2 = +1)$
- For any $\mathbf{x}$ close to any $\mathbf{x}'_i$:
  $K(\mathbf{x}, \mathbf{x}'_i) \simeq \exp(-\frac{0}{\sigma^2}) \simeq 1$
- For any $\mathbf{x}$ far from any $\mathbf{x}'_i$:
  $K(\mathbf{x}, \mathbf{x}'_i) \simeq \exp(\frac{-\text{Large value}}{\sigma^2}) \simeq 0$



- Take $\alpha = (1, 1, 0)$ and $b^* = -0.5$
  - For $\mathbf{x}$ close to $\mathbf{x}'_1$ and far from $\mathbf{x}'_2$ and $\mathbf{x}'_3$, decision:
    $= \alpha_1 * 1 + \alpha_2 * 0 + \alpha_3 * 0 - 0.5 = 1 - 0.5 = 0.5 \geq 0 \rightarrow$ positive
  - Any $\mathbf{x}$ far from all $\mathbf{x}'_i$:
    $= \alpha_1 * 0 + \alpha_2 * 0 + \alpha_3 * 0 - 0.5 = -0.5 < 0 \rightarrow$ negative
  - Draw the decision frontier

A basic generalization bound

- In statistical learning, one of the way to express generalization guarantees is to derive **PAC bounds**.
- They guarantee that we learn a classifier that has a generalization error smaller than ϵ (**Approximately Correct**).
- We allow that with a probability less than δ, we might not achieve this goal (**Probably Approximately Correct**).

Theorem (Vapnik and Chervonenkis)

Let T be a training set of n instances independently drawn from a distribution P over $\mathbb{R}^d \times \{-1, +1\}$. Let h be a linear classifier that perfectly separates T. Then, with a probability at least $1 - \delta$,

$$E(h) \leq \frac{2}{n}\left((d+1)\log\frac{2en}{d+1} + \log\frac{2}{\delta}\right).$$

Analysis of the basic bound

- The bound gets better as the number of training instances n grows.
- But it gets worse as d grows! This is the **penalization for complexity of the model** (a linear classifier has VC dimension $d + 1$).
- This bound essentially says that **kernelizing a linear classifier is not a good idea** because we will overfit.
- To better understand why SVM work, we need to derive a bound based on the **margin**.

($\Rightarrow$ There exist other theoretical justifications based on other theoretical frameworks (algorithmic (uniform) stability, algorithmic robustness, PAC-Bayesian theory, ...))

A margin-based generalization bound

Theorem (Vapnik)

Let T be a training set of n instances independently drawn from a distribution P over $\mathbb{R}^d \times \{-1, +1\}$. Let h be a linear classifier that perfectly separates T with margin at least γ. We assume $\|\mathbf{x}\|_2 \leq R$ for all $\mathbf{x}$ drawn from P. Then, with a probability at least $1 - \delta$,

$$E(h) \leq O\left(\frac{1}{n}\left(\frac{R^2}{\gamma^2} + \log\frac{1}{\delta}\right)\right).$$

- This bound is **independent from the dimension of the feature space**!
- Depends on the number of training examples and the margin.
- A **theoretical validation of the SVM framework** (margin maximization + kernel trick).

Summary

Hard-margin SVM

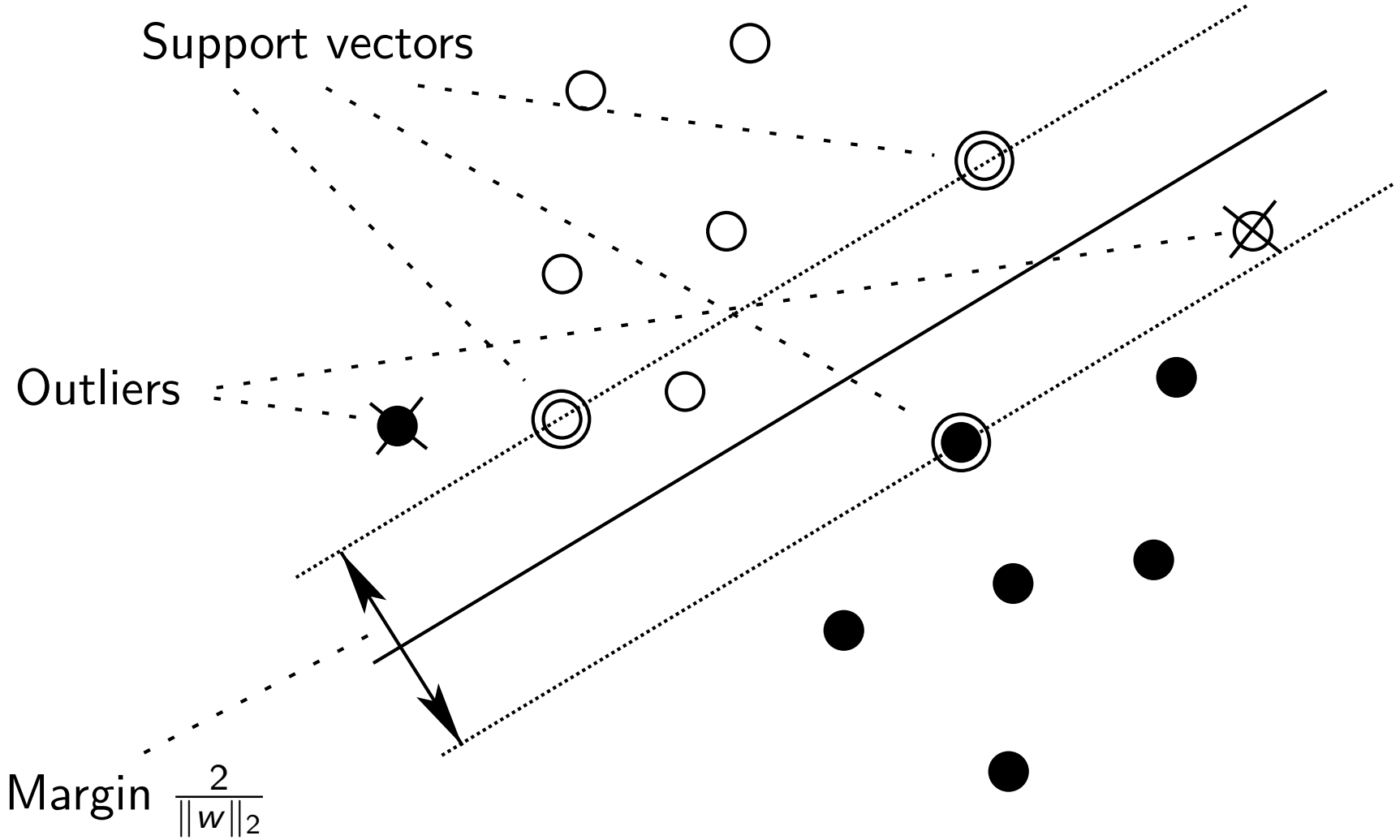
- Max-margin algorithm + kernel trick.
- Theoretically well-founded framework.
- The solution does not need all the examples in general.
- One thing still missing: relax the margin constraints!

Soft-margin Support Vector Machines

Soft-margin SVM

- In real world, problems are **not** linearly separable even in very high (infinite) dimension.
- This is because data is often **noisy** and there may exist **outliers**.
- Hard-margin SVM finds the largest-margin separating hyperplane.
- In soft-margin SVM, we want to optimize a **tradeoff** between the **margin** achieved by the solution and the **margin violations**:
  - This can be done by **relaxing** the hard-margin formulation using slack variables.
  - We can still derive the **dual problem**, allowing us to use the **kernel trick**.
  - We can still derive a **generalization bound**, which depends on the margin and the amount of violation (and of course on the size of the training set).

Soft-margin SVM: illustration



Back to the hard-margin primal

SVM hard-margin primal form

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1, \quad 1 \leq i \leq n \end{aligned}$$

- We want to allow some instances to violate the margin constraint but induce a penalty if this happens.

1-norm soft-margin primal SVM

SVM 1-norm soft-margin primal form

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i}\rangle + b) \geq 1 - \xi_i, \quad 1 \leq i \leq n \\ & \boldsymbol{\xi} \succeq 0 \end{aligned}$$

- We want to allow some instances to violate the margin constraint but induce a penalty if this happens.
- We do this by introducing slack variables $\boldsymbol{\xi} \succeq 0$ and a parameter $C \geq 0$.
- C is the tradeoff parameter:
  - $C = 0$: maximize the margin whatever it takes (stupid).
  - $C = +\infty$: equivalent to hard-margin.
  - $0 < C < +\infty$: tradeoff between maximizing the margin and minimizing the amount of violation of the margin.

Understanding the 1-norm soft-margin primal SVM

SVM 1-norm soft-margin primal form

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \tfrac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i}\rangle + b) \geq 1 - \xi_i, \quad 1 \leq i \leq n \\ & \boldsymbol{\xi} \succeq 0 \end{aligned}$$

Questions

1. Is the problem still convex?
2. What is the value of ξ_i when $\mathbf{x_i}$ satisfies the margin?
3. What is the value of ξ_i when $\mathbf{x_i}$ is misclassified?
4. What is the value of ξ_i when $\mathbf{x_i}$ doesn't satisfy the margin but is correctly classified?

Understanding the 1-norm soft-margin primal SVM

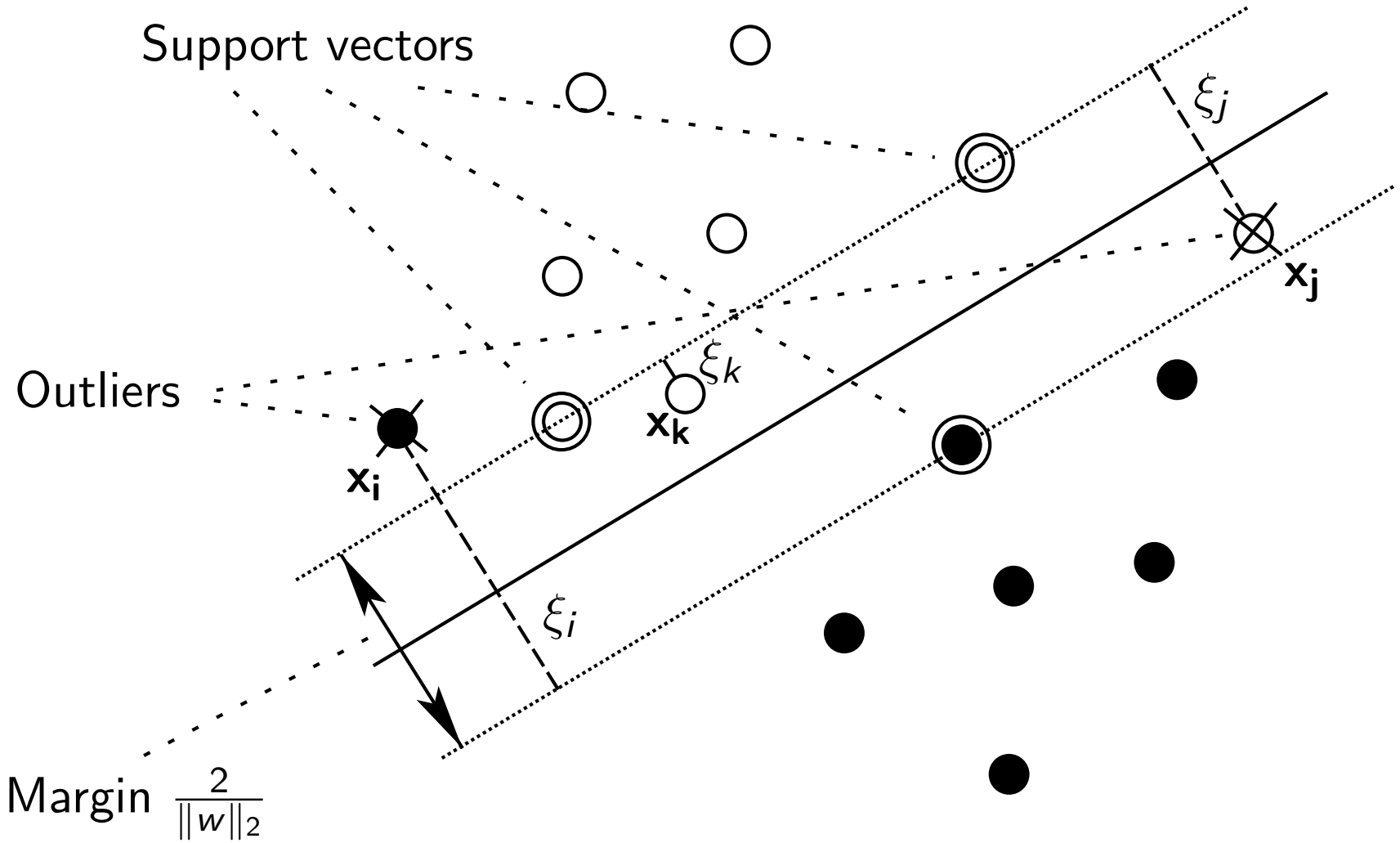
SVM 1-norm soft-margin primal form

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \tfrac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^n \xi_i \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \mathbf{x_i}\rangle + b) \geq 1 - \xi_i, \quad 1 \leq i \leq n \\ & \boldsymbol{\xi} \succeq 0 \end{aligned}$$

Answers

1. Is the problem still convex? Yes!
2. What is the value of ξ_i when $\mathbf{x_i}$ satisfies the margin? Answer: $\xi_i = 0$.
3. What is the value of ξ_i when $\mathbf{x_i}$ is misclassified? Answer: $\xi_i > 1$.
4. What is the value of ξ_i when $\mathbf{x_i}$ doesn't satisfy the margin but is correctly classified? Answer: $0 < \xi_i \leq 1$.

Soft-margin SVM: illustration ctd



Towards the dual

- The Lagrangian of the (constrained) problem is:

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i[1 - \xi_i - y_i(\langle \mathbf{w}, \mathbf{x_i}\rangle + b)] - \sum_{i=1}^{n}\beta_i\xi_i,$$

  where $\boldsymbol{\alpha}, \boldsymbol{\beta} \succeq 0$.

- We set the derivatives of L with respect to $\mathbf{w}$, b and $\boldsymbol{\xi}$ to 0:

$$\begin{cases} \frac{\partial L(\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{n} y_i\alpha_i\mathbf{x_i} = 0 \\ \frac{\partial L(\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})}{\partial \mathbf{b}} &= \sum_{i=1}^{n} y_i\alpha_i = 0 \\ \frac{\partial L(\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta})}{\partial \boldsymbol{\xi}} &= C - \alpha_i - \beta_i = 0 \end{cases}$$

Towards the dual ctd

- We thus have:

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x_i}, \quad \sum_{i=1}^{n} y_i \alpha_i = 0 \quad \text{and} \quad \beta_i = C - \alpha_i.$$

- Substitute in L to get the dual function $g(\boldsymbol{\alpha}, \boldsymbol{\beta})$ to maximize:

$$\begin{aligned}
L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i[1 - \xi_i - y_i(\langle \mathbf{w}, \mathbf{x_i}\rangle + b)] - \sum_{i=1}^{n}\beta_i\xi_i \\
&= \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j}\rangle + C\sum_{i=1}^{n}\xi_i - \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j}\rangle \\
&\quad + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\alpha_i\xi_i - \sum_{i=1}^{n}(C - \alpha_i)\xi_i \\
&= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j}\rangle \\
&= g(\boldsymbol{\alpha}, \boldsymbol{\beta}).
\end{aligned}$$

SVM 1-norm soft-margin dual

SVM 1-norm soft-margin dual form

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \quad g(\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j} \rangle$$

$$\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$C - \alpha_i - \beta_i = 0, \quad 1 \leq i \leq n$$

$$\boldsymbol{\alpha} \succeq 0$$

$$\boldsymbol{\beta} \succeq 0$$

SVM 1-norm soft-margin dual

SVM 1-norm soft-margin dual form

$$\max_{\boldsymbol{\alpha}} \quad g(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j} \rangle$$

$$\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq C, \quad 1 \leq i \leq n$$

Remarks

- Almost the same formulation as the hard-margin one.
- What is the intuition behind this additional constraint?

SVM 1-norm soft-margin dual

SVM 1-norm soft-margin dual form

$$\max_{\boldsymbol{\alpha}} \quad g(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x_i}, \mathbf{x_j} \rangle$$

$$\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq C, \quad 1 \leq i \leq n$$

Intuition behind the constraint

Recall that α_i corresponds to the weight we put on $\mathbf{x_i}$ and that we put more weight on harder instances. Now, can only put a bounded weight! Note that if $C = +\infty$, this is equivalent to hard-margin.

Generalization bound

Theorem (Vapnik)

Let T be a training set of n instances independently drawn from a distribution P over $\mathbb{R}^d \times \{-1, +1\}$. Let h a linear classifier learned on T with margin at least γ and $\boldsymbol{\xi} = (\max[0, \gamma - y_i h(\mathbf{x_i})])_{i=1}^n$. We assume $\|\mathbf{x}\|_2 \leq R$ for all $\mathbf{x}$ drawn from P. Then, with a probability at least $1 - \delta$,

$$E(h) \leq O\left(\frac{1}{n}\left(\frac{R^2 + \|\boldsymbol{\xi}\|_2^2}{\gamma^2} + \log\frac{1}{\delta}\right)\right).$$

When $\|\boldsymbol{\xi}\|_2^2 = 0$ (i.e., h perfectly separates T), this reduces to the hard-margin generalization bound.

Notes on the parameters

Primal formulation

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad y_i(\langle \mathbf{w}, \mathbf{x_i}\rangle + b) \geq 1 - \xi_i, \quad 1 \leq i \leq n, \boldsymbol{\xi} \succeq 0$$

Note

- Large C: lower bias, high variance
- Small C: higher bias, low variance

Note on σ^2, Gaussian kernel $\exp(\frac{\|\mathbf{x}-\mathbf{x}'\|}{\sigma^2})$

- Large σ^2: higher bias, low variance (features vary more smoothly)
- Small σ^2: lower bias, high variance (features vary less smoothly)

$\Rightarrow$ Need to tune the hyperparameters (C and kernel parameters (σ, degree of the polynomial kernel, ...)) by cross-validation for example.

Unconstrained formulation

Note that we can formulate the problem in an unconstrained form.

Unconstrained 1-norm SVM primal form

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}[1 - y_i(\langle \mathbf{w}, \mathbf{x_i}\rangle + b)]_+,$$

where $[1-a]_+ = \max(0, 1-a)$ is the **hinge-loss**.

This formulation is convex but not differentiable and easier to deal with for deriving generalization guarantees.

Uniform stability and soft-margin

We consider the simplified version:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \lambda\|\mathbf{w}\|_2^2 + \frac{1}{n}\sum_{i=1}^{n}[1 - y_i(\langle \mathbf{w}, \mathbf{x_i}\rangle)]_+$$

- the hinge loss is 1-admissible (consider the 4 cases):
  $|[1 - y(\langle \mathbf{w}, \mathbf{x}\rangle + b)]_+ - [1 - y(\langle \mathbf{w}, \mathbf{x}'\rangle + b)]_+| \leq |\langle \mathbf{w}, \mathbf{x}'\rangle - \langle \mathbf{w}, \mathbf{x}\rangle|$
- We assume the kernel is bounded $\langle \mathbf{x}, \mathbf{x}\rangle \leq \kappa$
- Since $\mathbf{w}$ is optimum, we have $\|w\|_2^2 \leq 1/\lambda$.
- Using the same strategy as in the exercise on regression (we have the same regularizer), we can prove that the stability constant is $\frac{\kappa^2}{2\lambda n}$.
- We have the following bound: $R \leq \hat{R} + \frac{\kappa^2}{\lambda n} + \left(1 + \frac{2\kappa^2}{\lambda}\right)\sqrt{\frac{\ln(1/\delta)}{2n}}$
- We can generalize to other kernels by assuming $k(\mathbf{x}, \mathbf{x}') \leq \kappa \rightarrow$ exercise

Summary

Soft-margin SVM:

- We now can deal with any binary classification problem (even nonlinearly separable).
- Other formulations exist, based for instance on penalizing by $\sum_{i=1}^{n} \xi_i^2$.
- Can still use the kernel trick and derive a generalization bound.
- Can deal with the multiclass setting (using a one-vs-one or one-vs-all approach).
- Can be adapted for regression (including kernelized regression) and other problems.

Two important problems left open:

1. The size of the problem grows with the number of training instances. How can we deal with large-scale learning?
2. The choice of the kernel, as well as the parameter C, is essential to the performance. Can we do better than cross-validation?

Some improvements

Fast solving methods

After SVM was first proposed, more efficient solving methods were developed to deal with large-scale learning.

- **Chunking methods**: learn SVM on a chunk (subset) of the training examples and keep only the support vectors. Add the examples that violates the most the KKT conditions and learn a new model. Iterate until the KKT conditions are satisfied. At each iteration, $\boldsymbol{\alpha}$ is initialized to the previous optimal value. No proof of convergence, but works well in practice.
- **Sequential Minimal Optimization (SMO)**: extreme case of chunking where we only consider chunks of size 2. At each step, SMO chooses α_i and α_j that will be modified to improve the objective function. This is very efficient since an analytical solution to the SVM problem exists when we only consider 2 points. Smart heuristics to pick the points. Guaranteed to converge. Used in popular librairies, such as LIBSVM.
- **Stochastic Gradient Descent (SGD)** (better with logistic loss)

Multiple Kernel Learning (MKL)

- First proposed in 2004 and improved many times (still a very hot topic of research).
- Motivation: for some tasks, it can be useful to exploit the information brought by several kernels.
- Idea: learn a SVM model based on a **combination of kernels**, and learn also this combination.
- Recent improvements use smart regularization on the weights to induce sparsity and therefore an **automatic selection of kernels**.
- Can be used to automatically choose a kernel and its parameters.

Kernel Learning

- Motivation: standard kernels may not be very appropriate to the task of interest. This is why people constantly propose new kernels for specific domains.
- Idea: what about **automatically learning the kernel function** from data?
- Subfield of metric and similarity learning.
- However, can't learn a kernel of arbitrary form. So essentially learn the parameters of the kernel.
- Some approaches only learn the kernel matrix: more flexible but hard to generalize to new points.
- Other approaches learn a similarity function or metric (not a valid kernel) and transform it to a kernel function (or not).
- There also exists alternative theories of learning with nonPSD similarity functions.

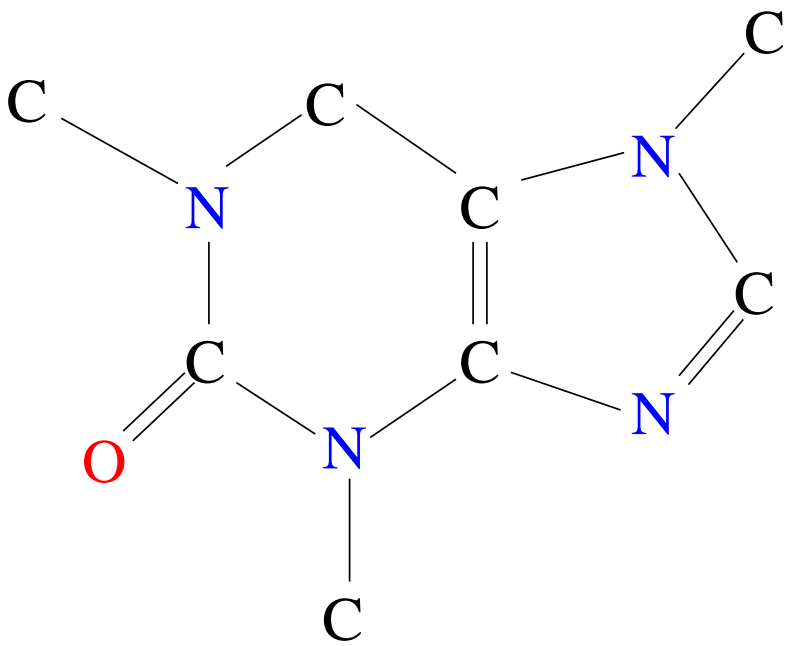
Theoretical framework

- Kernel trick can be used for various problems: kernel perceptron, kernel ridge regression, kernel fisher discriminant, kernel principal component analysis
- Representer theorem: (roughly speaking) for well regularized machine learning optimization problem, any solution admits a representation of the form: $f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot)$
  $\rightarrow$ We can directly look for such representations and learn the alpha's
- Reproducing Kernel Hilbert Space: Hilbert space on functions.
- Various generalization results (Uniform stability, Rademacher analysis, VC dimension, ...)
- Various extensions where support vectors are landmarks, various norms to better control sparsity, ...

Structured kernels

Structured data

- Typically, dealing with structured data is difficult (can be of various sizes, no straightforward way to do calculus, etc).
- However, real-life data is often more structured than simple vectors.



Classifications problems

Classifying sequences

- Text classification.
- Music/sound/speech classification.
- Proteins structure prediction.
- ...

Classifying graphs

- Natural language processing (sentences viewed as trees).
- Digital image interpretation (different areas of one image are interpreted as vertices of a graph).
- Classification of chemical compounds.
- ...

1st example - edit distance kernel: $k(\mathbf{x}, \mathbf{x}') = \exp(\gamma ED(\mathbf{x}, \mathbf{x}'))$, where $\gamma > 0$ must be tuned to ensure PSD of k.

Structured kernels

- Using a kernel offers an easy way of dealing with structured data...
- ... as long as an appropriate structured kernel is available!
- They are often built using an explicit transformation Φ.
- Sequences: spectrum kernel, mismatch kernel, Fisher kernels, edit kernels,...
- Graphs: convolution kernels, walk/path-based kernels,...

Illustration: spectrum kernel

Strategy: count the number of subsequences of length k.

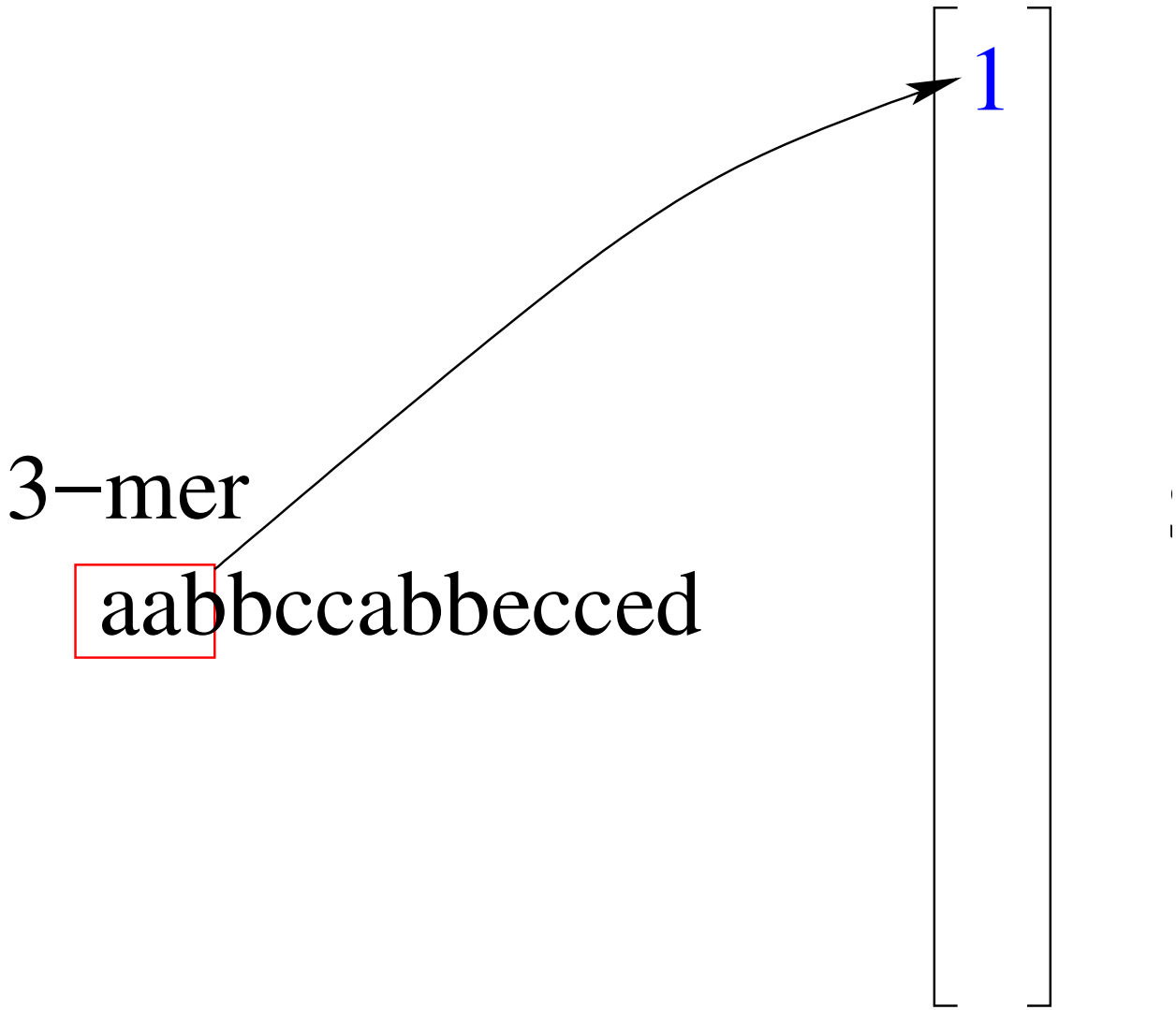


Illustration: spectrum kernel

Strategy: count the number of subsequences of length k.

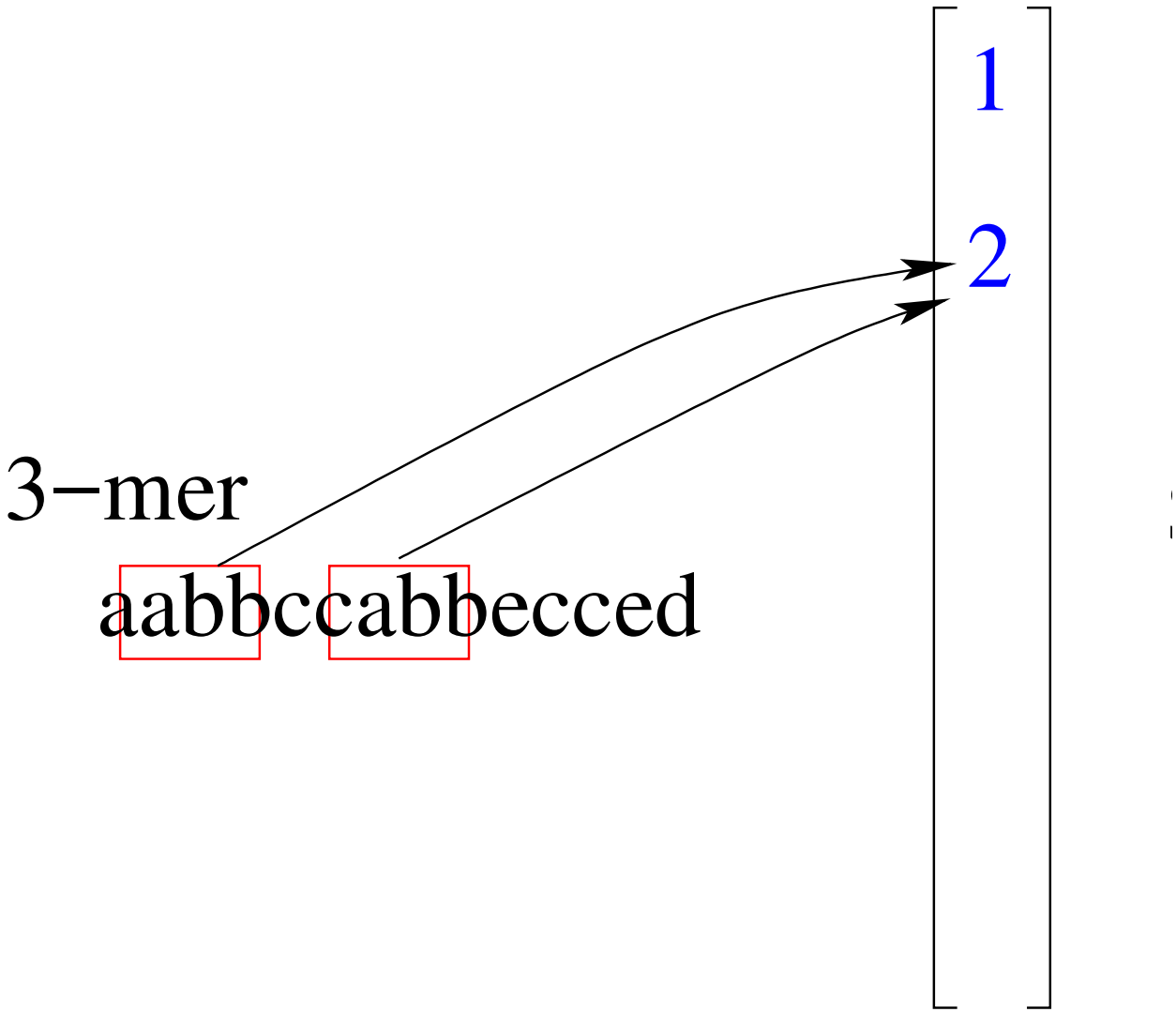


Illustration: spectrum kernel

Strategy: count the number of subsequences of length k.