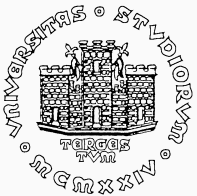
Computer Vision and Pattern Recognition

Course ID: 554SM – Fall 2018

Felice Andrea Pellegrino

University of Trieste
Department of Engineering and Architecture



554SM –Fall 2018

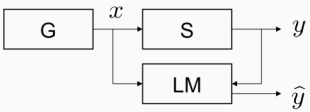
Lecture 7: Support vector machines for classification

Index

- Basics of Statistical Learning Theory
  - Empirical risk minimization principle
  - Capacity of the hypothesis set
  - Structural risk minimization principle
- Binary classification:
  - Linear machines
  - Perceptrons and potential functions
  - Nonlinear machines
- Multi-class classification:
  - One-against-rest SVM
  - Pairwise SVM
  - All-at-once SVM
- Incorporating a priori knowledge:
  - Virtual support vectors
  - Kernel jittering

Basics of Statistical Learning Theory

Supervised learning



$$x \sim F(x)$$
$$y \sim F(y|x)$$

The learning problem: finding the desired dependence from a limited number of observations

- Generator (G)
- Supervisor (S)
- Learning machine (LM)

- $F(x)$ probability distribution of $x \in \mathcal{X} \subseteq \mathbb{R}^n$
- $F(y|x)$ conditional distribution of $y \in \mathcal{Y} \subseteq \mathbb{R}$ (a special case is the deterministic one $y = f(x)$)
- $F(x, y) = F(x)F(x|y)$ joint distribution, defined in $\mathcal{Y} \times \mathcal{X}$. Fixed but unknown.

Supervised learning (cont.)

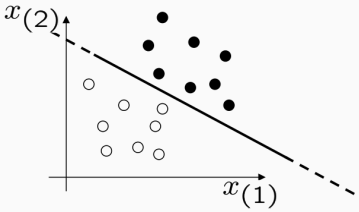
- $\mathcal{S} = \{(x_1, y_1), \ldots, (x_l, y_l)\}$ set of observed (x, y) pairs (*Training Set, TS*)
- $\mathcal{H}$ set of functions $h : \mathcal{X} \to \mathcal{Y}$, for example: $\mathcal{H} = \{h(x, \alpha), \alpha \in \Lambda\}$

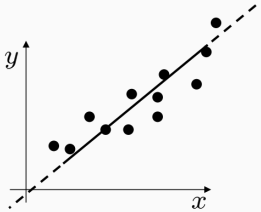
- Goal: choose *the best* among the functions $h \in \mathcal{H}$;
- the best function is the one guaranteeing the maximum "prediction performance" (in a sense to be specified)

Two typical problems

Two typical problems that can be cast in the supervised learning framework are:

- binary classification: $y \in \mathcal{Y} = \{-1, 1\}$
- regression: $y \in \mathcal{Y} \subseteq \mathbb{R}$.





The risk functional

- What is the best choice for $h \in \mathcal{H}$?
- Define a *risk functional*:

$$R(h) = \int_{\mathcal{X} \times \mathcal{Y}} L(y, h(x)) dF(x, y)$$

- $R(h)$ is the expected value of the loss $L(y, h(x))$ in which one incurs when the machine, having x as input, produces $h(x)$ instead of the value y produced by the supervisor

Examples of loss functions

- 0/1 loss function

$$L(y, h(x)) = \begin{cases} 0 \text{ if } y[h(x)] > 0 \\ 1 \text{ otherwise} \end{cases}$$

- Quadratic loss function

$$L(y, h(x)) = (y - h(x))^2$$

- Linear loss function

$$L(y, h(x)) = |y - h(x)|$$

Empirical risk

- The best choice is then $h^*(x)$ where

$$h^* = \arg\min_{h \in \mathcal{H}} R(h) = \arg\min_{h \in \mathcal{H}} \int L(y, h(x)) dF(x, y)$$

- However $F(x, y)$ is not known and therefore it is impossible to minimize $R(h)$ directly.
- What we actually know is the *empirical risk*:

$$R_{emp}(h) = \frac{1}{l} \sum_{i=1}^{l} L(y_i, h(x_i))$$

Empirical risk minimization principle

Empirical Risk Minimization (ERM) is an inductive principle applied in many well known methods, e.g.

- Least squares regression

$$R_{emp}(h) = \frac{1}{l} \sum_{i=1}^{l} (y_i - h(x_i))^2$$

- Density estimation using maximum likelihood

$$R_{emp}(p) = -\frac{1}{l} \sum_{i=1}^{l} \ln p(x_i)$$

Empirical risk minimization principle (cont.)

- Is it always a good idea minimizing the empirical risk?
- If not, which are the properties that a given learning problem must enjoy in order for the empirical risk minimization approach to be reasonable?
- Answers are provided by the *Statistical Learning Theory* (Vapnik, 2000).

Law of large numbers and consistency of ERM

- The law of large numbers (convergence of the mean to the expected value) guarantees that for a given $\bar{h}$, as the number of observations l increases:

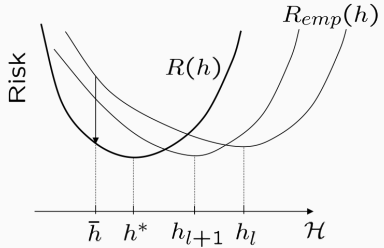
$$R_{emp}(\bar{h}) \rightarrow R(\bar{h})$$

  or, more precisely:

$$\lim_{l \rightarrow \infty} P(|R_{emp}(h) - R(h)| > \epsilon) = 0, \quad \forall \epsilon > 0$$

- However, this fact does not guarantee the consistency of the ERM, namely that for an infinite number of observations, by applying ERM, one gets the function $h^* \in \mathcal{H}$ that minimizes the expected risk.

Law of large numbers and consistency of ERM (cont.)

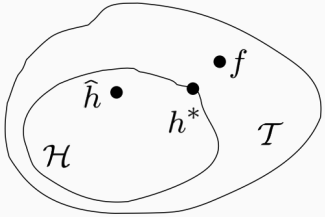


- ERM is said to be consistent if the sequence $\{h_l\}$ converges to h^*.
- Necessary and sufficient condition: uniform convergence (uniform in the class of functions that the machine can implement)

$$\lim_{l\to\infty} P\left(\sup_{h\in\mathcal{H}}\left(R(h) - R_{emp}(h)\right) > \epsilon\right) = 0, \quad \forall\epsilon > 0$$

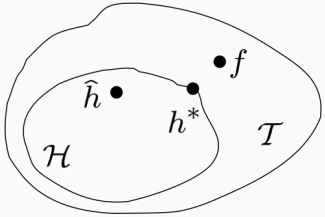
- How can we guarantee that? By choosing properly the set $\mathcal{H}$.

Approximation, estimation and generalization errors



- $\mathcal{T}$ (target set): the set of function that is assumed to contain the "true" one f
- h^*: the best possible choice in $\mathcal{H}$
- $\hat{h}$: the function chosen by the learning algorithm based on the training set.

Approximation, estimation and generalization errors



- Approximation error (due to the fact that $f \notin \mathcal{H}$)

$$R(h^*) - R(f)$$

- Estimation error (due to the fact that $\hat{h} \neq h^*$)

$$R(\hat{h}) - R(h^*)$$

- The *generalization error* (what we really what to minimize!) is the sum of the two:

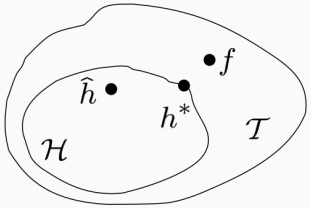
$$R(\hat{h}) - R(f) = [R(\hat{h}) - R(h^*)] + [R(h^*) - R(f)]$$

Minimizing the generalization error

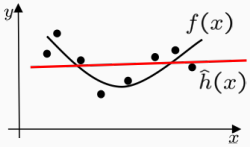
To minimize the generalization error we have to minimize the sum of two terms:

$$R(\hat{h}) - R(f) = [R(\hat{h}) - R(h^*)] + [R(h^*) - R(f)]$$

- Clearly, the second term (approximation error) can be reduced by taking $\mathcal{H}$ "rich";
- On the other hand, it can be shown that taking $\mathcal{H}$ too "rich" renders large the first term (or even inconsistent the ERM principle).



Bias-Variance dilemma



- Least squares regression of the function

$$y = f(x) + e$$

  where e represents a measurement noise;

- Bias:

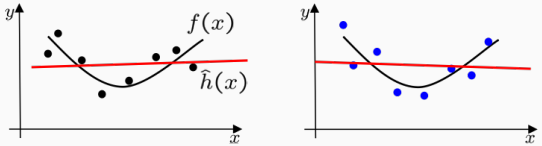
$$\mathbb{E}_{\mathcal{S}} R(\hat{h})$$

- Variance:

$$\mathbb{E}_{\mathcal{S}}[R(\hat{h}) - \mathbb{E}_{\mathcal{S}} R(\hat{h})]^2$$

- If the hypothesis set is restricted to linear functions, the solution has only small changes for different training sets (low variance). But the class of functions is poorly flexible (high bias)
- If we consider high degree polynomials, we observe a better fit (low bias) but the solution is more sensitive to noise and varies quite a lot for different training sets (high variance)

Bias-Variance dilemma



- Least squares regression of the function

$$y = f(x) + e$$

where e represents a measurement noise;

- Bias:

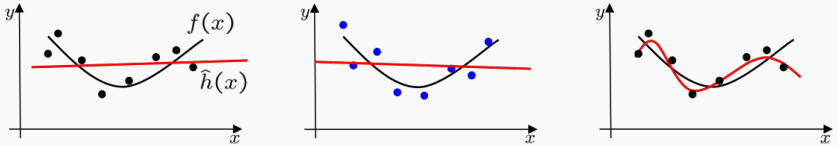
$$\mathbb{E}_{\mathcal{S}} R(\hat{h})$$

- Variance:

$$\mathbb{E}_{\mathcal{S}}[R(\hat{h}) - \mathbb{E}_{\mathcal{S}} R(\hat{h})]^2$$

- If the hypothesis set is restricted to linear functions, the solution has only small changes for different training sets (low variance). But the class of functions is poorly flexible (high bias)
- If we consider high degree polynomials, we observe a better fit (low bias) but the solution is more sensitive to noise and varies quite a lot for different training sets (high variance)

Bias-Variance dilemma



- Least squares regression of the function

$$y = f(x) + e$$

where e represents a measurement noise;

- Bias:

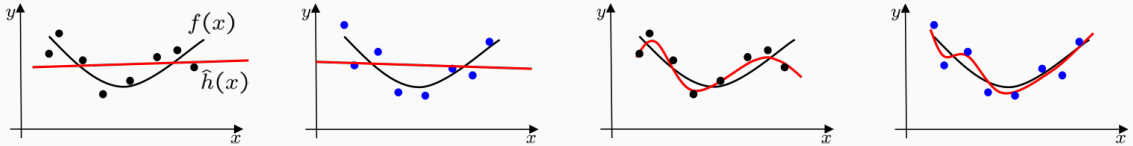
$$\mathbb{E}_{\mathcal{S}} R(\hat{h})$$

- Variance:

$$\mathbb{E}_{\mathcal{S}}[R(\hat{h}) - \mathbb{E}_{\mathcal{S}} R(\hat{h})]^2$$

- If the hypothesis set is restricted to linear functions, the solution has only small changes for different training sets (low variance). But the class of functions is poorly flexible (high bias)
- If we consider high degree polynomials, we observe a better fit (low bias) but the solution is more sensitive to noise and varies quite a lot for different training sets (high variance)

Bias-Variance dilemma



- Least squares regression of the function

$$y = f(x) + e$$

  where e represents a measurement noise;

- Bias:

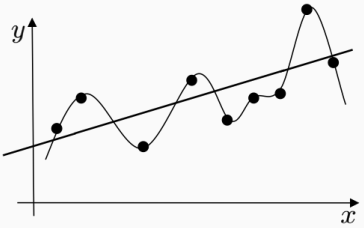
$$\mathbb{E}_{\mathcal{S}} R(\hat{h})$$

- Variance:

$$\mathbb{E}_{\mathcal{S}}[R(\hat{h}) - \mathbb{E}_{\mathcal{S}} R(\hat{h})]^2$$

- If the hypothesis set is restricted to linear functions, the solution has only small changes for different training sets (low variance). But the class of functions is poorly flexible (high bias)
- If we consider high degree polynomials, we observe a better fit (low bias) but the solution is more sensitive to noise and varies quite a lot for different training sets (high variance)

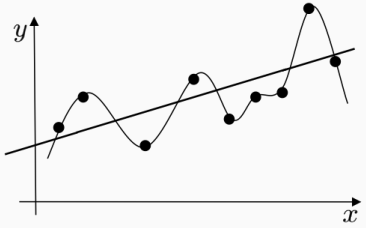
Overfitting

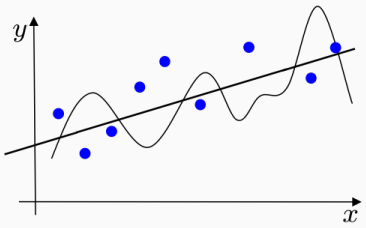


When the hypothesis set is too rich, the ERM leads to *overfitting*.

A function is said to "overfit" if there is another function in $\mathcal{H}$ having less expected risk (therefore, it is preferable) but more empirical risk (therefore is discarded by a learning algorithm that implements ERM).

Overfitting





When the hypothesis set is too rich, the ERM leads to *overfitting*.

A function is said to "overfit" if there is another function in $\mathcal{H}$ having less expected risk (therefore, it is preferable) but more empirical risk (therefore is discarded by a learning algorithm that implements ERM).

Capacity of the hypothesis set: an example

- Binary classification:
$$\mathcal{Y} = \{-1, +1\}\,,\ \mathcal{S} = \{(x_1, y_1), \ldots, (x_l, y_l)\}$$
- Let's suppose that $\mathcal{H}$ is the set of all the functions from $\mathcal{X}$ to $\mathcal{Y}$.
- Then, for all $\mathcal{S}$, there exist in $\mathcal{H}$ infinitely many functions having zero empirical risk, precisely all the functions such that
$$h(x_i) = y_i\ \forall i = 1, \ldots, l$$
- Based on the training set only, it is not possible to decide which is the best one. Therefore, no learning is possible.

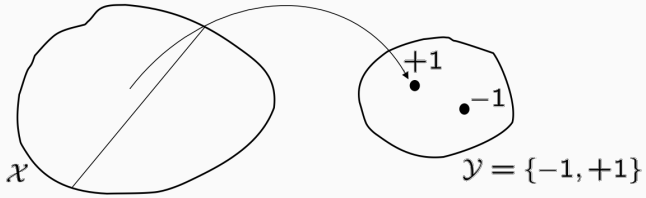
Capacity of the hypothesis set: an example (cont.)

- It follows that it is necessary to restrict the hypothesis set.
- How "rich" is the set? If it is too rich, then its capability of explaining the observed data does not decrease as $ln \to \infty$. Each new sample does not carry any information that can be used for the classification of its neighbors.
- How do we quantify the "richness" of the hypothesis set? A possibility is to use the *Vapnik-Chervonenkis dimension* (*VC-dimension*).

- Consider, for the moment, *indicator functions* only:

$$h : \mathcal{X} \to \{-1, +1\}$$

- Indicator functions realize dichotomies, i.e. they assign to either one of two classes each of the members of $\mathcal{X}$



VC-dimension (cont.)

- A given $\mathcal{Z} \subseteq \mathcal{X}$ is said to be *shattered* by the hypothesis set $\mathcal{H}$ if $\mathcal{H}$ realizes all the possible dichotomies of $\mathcal{Z}$.
- The VC-dimension of a set $\mathcal{H}$ of indicator functions defined in $\mathcal{X}$ is the cardinality of the biggest subset of $\mathcal{X}$ that is shattered by $\mathcal{H}$:

$$VC(\mathcal{H}) = \max_{\mathcal{Z} \subseteq \mathcal{X}} \{|\mathcal{Z}| : \mathcal{Z} \text{ is shattered by } \mathcal{H}\}$$

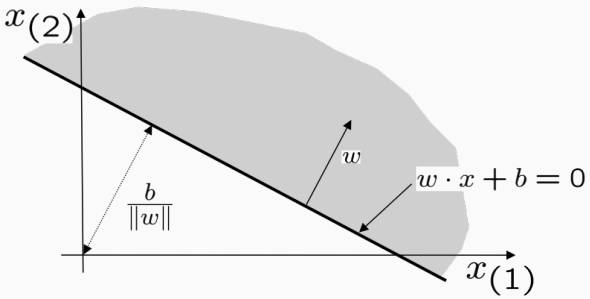
VC-dimension, an example

- Construct a parametric family of indicator functions defined in $\mathbb{R}^2$ by composing a threshold function and an affine function (separating hyperplanes in $\mathbb{R}^2$)

$$\mathcal{H} = \{h(x) : h(x) = \Theta(w \cdot x + b),\ w \in \mathbb{R}^2, b \in \mathbb{R}\}$$

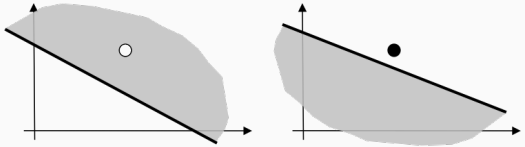
where

$$\Theta(z) = \begin{cases} -1 & \text{if} \quad z < 0 \\ 1 & \text{if} \quad z \geq 0 \end{cases}$$

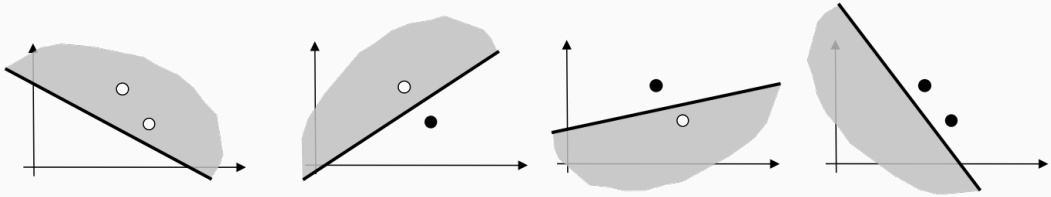


VC-dimension, an example (cont.)

- What is the VC-dimension of $\mathcal{H}$?
- Take a single point ($2^1 = 2$ dichotomies)

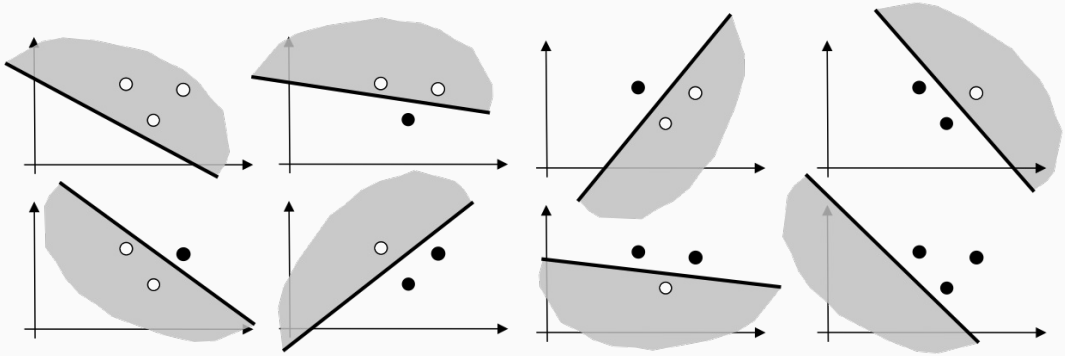


- Take two points ($2^2 = 4$ dichotomies)

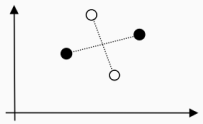


VC-dimension, an example (cont.)

- Take three point ($2^3 = 8$ dichotomies)



- There is no configuration of four points in $\mathbb{R}^2$ that can be shattered by planes $\implies VC(\mathcal{H}) = 3$



VC-dimension and number of parameters

In general, the VC-dimension of a set of separating hyperplanes in $\mathbb{R}^n$ is $n+1$.

The VC-dimension is, in general, different from the number of parameters. For example:

- for multilayer perceptrons with threshold activation function it is proportional to $n \ln(n)$;
- for multilayer perceptrons with sigmoidal activation function it is proportional to n^2;
- for functions like

  $$\mathcal{H} = \{h(x) : h(x) = \Theta[\sin(\alpha x)],\ \alpha \in \mathbb{R}\}$$

  it is infinite.

VC-dimension and cardinality of the hypothesis set

In general, the VC-dimension of a set of separating hyperplanes in $\mathbb{R}^n$ is $n + 1$.

The VC-dimension is related to the cardinality of the hypothesis set $\mathcal{H}$:

- the number of dichotomies of a set of m members is 2^m,
- then $\mathcal{H}$ contains at least $2^{VC(\mathcal{H})}$ different functions:

$$|\mathcal{H}| \geq 2^{VC(\mathcal{H})}.$$

  It follows that:

$$VC(\mathcal{H}) \leq \log_2 |\mathcal{H}|.$$

- This shows that the VC-dimension is related to the "richness" of the hypothesis set.

Consistency of ERM for the classification problem

- A necessary and sufficient condition for the consistency of the ERM independent of the distribution $F(x, y)$ is the finiteness of the VC-dimension of the hypothesis set:

$$d = VC(\mathcal{H}) < \infty \quad \Longleftrightarrow \quad \text{ERM consistent on } \mathcal{H}, \forall F(x, y)$$

- Whenever this condition holds, independent of the distribution, for $l \to \infty$, the empirical risk minimization leads to the best function of the hypothesis set $\mathcal{H}$.
- For the regression problem, an analogous condition holds, but related to the so called V_γ-dimension of real-valued functions (Vapnik, 2000).

Non-asymptotic bounds

What can be said for a finite number of observations?

Based on the VC-dimension, it is possible to formulate non–asymptotic, distribution-independent bounds for the risk, such as:

Non-asymptotic, distribution independent bound

With probability $\geq 1 - \delta$, $\forall h \in \mathcal{H}$ the following holds:

$$R(h) \leq R_{emp}(h) + \phi\left(d, l, \ln(\delta)\right)$$

where ϕ is the *confidence* and is an increasing function of d and a decreasing function of l and δ.

Notice that the confidence depends on the class of functions, on the number of observations and on the probability $1 - \delta$.

Notice that the bound holds for any $h \in \mathcal{H}$ thus may be applied to learning machines that do not implement ERM.

Roughly speaking, the confidence term depends on the "worst" h, meaning the most deceptive (i.e. a function that may exhibit the least empirical risk over a training set of l observations but has actually the highest risk).

Non-asymptotic bounds: example

A well-known non-asymptotic bound is:

$$R(h) \leq R_{emp}(h) + \sqrt{\frac{d\left(\ln \frac{2l}{d} + 1\right) - \ln \frac{\delta}{4}}{l}}$$

It holds, with probability $\geq 1 - \delta$, for indicator functions (binary classification problem) and 0/1-loss.

For l (number of observations) fixed, the confidence is a monotonically decreasing function of the VC-dimension d of the hypothesis set.

The bound is independent of

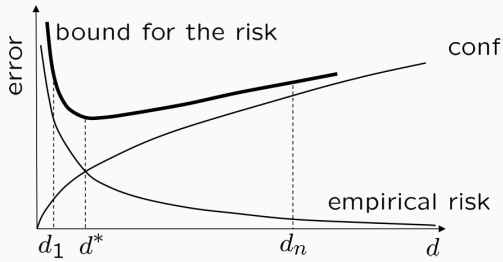
- the distribution that generates the data;
- the dimension of $\mathcal{X}$.

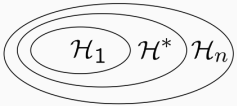
Observations

$$R(h) \leq R_{emp}(h) + \sqrt{\frac{d\left(\ln \frac{2l}{d} + 1\right) - \ln \frac{\delta}{4}}{l}}$$

- It holds $\forall h \in \mathcal{H}$ and thus it is valid for every algorithm, even for those algorithms that do not minimize the empirical risk;
- from a practical standpoint, it may be not very useful (the bound can be quite high);
- however, it is conceptually useful because it suggests a strategy different from ERM: **minimize the sum of empirical risk and the confidence**.

Structural Risk Minimization (SRM) principle





Why “structural”?

Because we give the hypothesis set a structure by considering nested sets

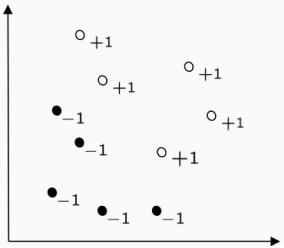
$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \cdots \subset \mathcal{H}_n \ldots$$

(thus the VC-dimension is non-decreasing) and this fact allows for minimizing the bound taking into account both the terms (empirical risk and confidence).

Binary classification

The binary classification problem

Given a training set $\mathcal{S} = \{(x_1, y_1), \ldots, (x_l, y_l)\}$ where each vector $x_i \in \mathcal{X} \subseteq \mathbb{R}^n$ is assigned to either one of the two classes: $y_i \in \mathcal{Y} = \{-1, +1\}$



The pairs (x_i, y_i) are random variables, independent and identically distributed according to an unknown distribution $F(x, y)$.

Find a decision function $h(x) : \mathcal{X} \to \mathcal{Y}$ able to correctly classify *new* vectors, i.e. vectors that do not belong to the training set

Linear decision function

Call (with slight abuse of notation) linear a decision function of the following type:

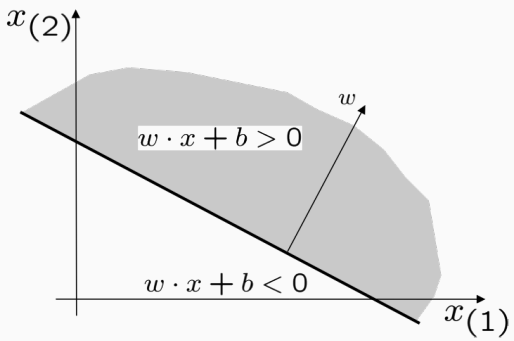
$$h(x) = \Theta(w \cdot x + b),\ w \in \mathbb{R}^n, b \in \mathbb{R}$$

where

$$\Theta(z) = \begin{cases} -1 & \text{if} \quad z < 0 \\ 1 & \text{if} \quad z \geq 0 \end{cases}$$

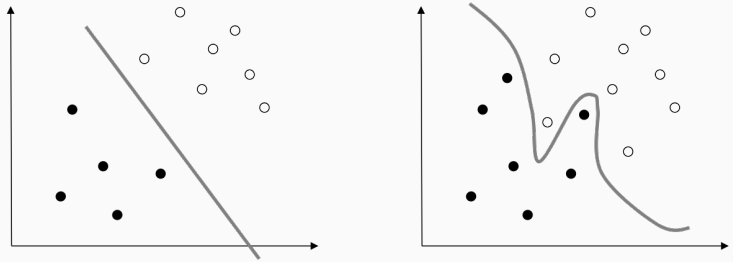
A decision function of this type, splits $\mathbb{R}^n$ into two regions by means of the separating hyperplane:

$$w \cdot x + b = 0$$



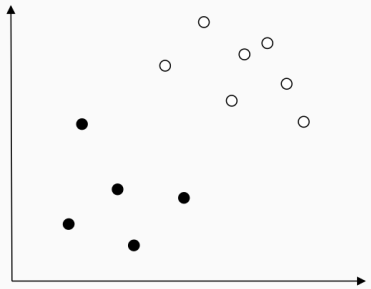
(Non) linearly separable training set

- A training set is said *linearly separable* if a hyperplane does exist that classifies correctly all the samples.
- Otherwise, the training set is said non linearly separable.



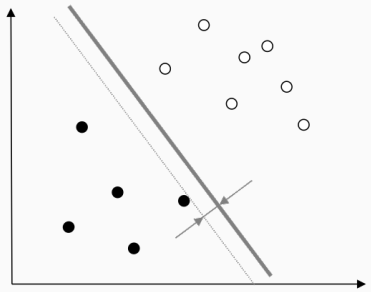
Maximal margin hyperplane (linearly separable case)

- Consider a linearly separable training set.
- Among all possible separating hyperplanes, there is a hyperplane that maximizes the *margin*, i.e. the distance from the closest example (*maximal margin hyperplane*).



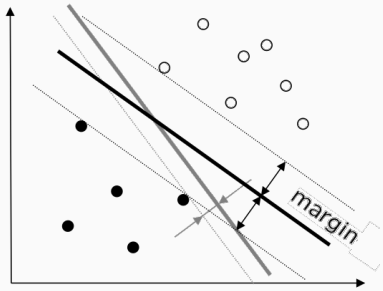
Maximal margin hyperplane (linearly separable case)

- Consider a linearly separable training set.
- Among all possible separating hyperplanes, there is a hyperplane that maximizes the *margin*, i.e. the distance from the closest example (*maximal margin hyperplane*).



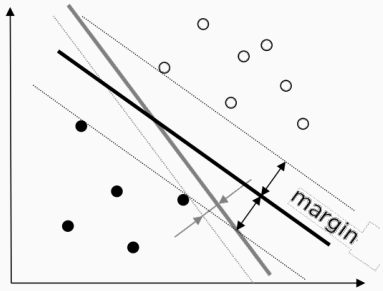
Maximal margin hyperplane (linearly separable case)

- Consider a linearly separable training set.
- Among all possible separating hyperplanes, there is a hyperplane that maximizes the *margin*, i.e. the distance from the closest example (*maximal margin hyperplane*).



Maximal margin hyperplane (linearly separable case)

- Consider a linearly separable training set.
- Among all possible separating hyperplanes, there is a hyperplane that maximizes the *margin*, i.e. the distance from the closest example (*maximal margin hyperplane*).

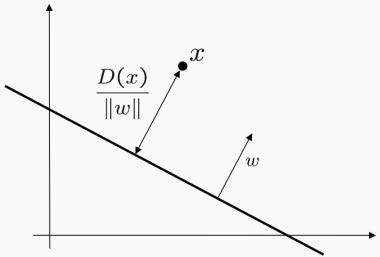


- How can we compute the maximal margin hyperplane?
- What are the properties of such a hyperplane?

Computing the maximal margin hyperplane

- Let $D(x) \doteq w \cdot x + b = 0$ be the equation of the hyperplane.
- Recall that the point to plane distance formula is $d = \dfrac{w \cdot x + b}{\|w\|}$.
- Thus, the (signed) distance between the hyperplane and x is $\dfrac{D(x)}{\|w\|}$.
- Any separating hyperplane satisfies the following:

$$\frac{y_i D(x_i)}{\|w\|} \geq M > 0, \ \forall i = 1, \ldots, l$$



- The maximal margin is then:

$$M^* = \max_{\|w\|=1, b} \min_i y_i D(x_i),$$

where the constraint $\|w\| = 1$ has been introduced for resolving the redundancy of representation.

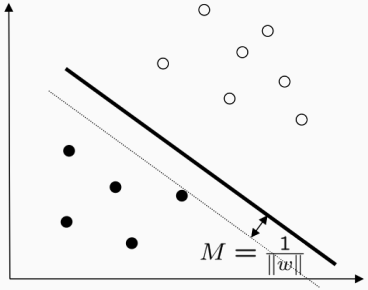
Computing the maximal margin hyperplane (cont.)

- Instead of imposing the constraint $\|w\| = 1$, given a hyperplane with margin M with respect to a training set, we can choose among its infinite representations the one satisfying:

$$M \|w\| = 1.$$

- Hyperplanes in this form are said *canonical* and are such that

$$\min_{i=1,\dots,l} y_i D(x_i) = 1$$



- If the optimization is restricted to canonical hyperplanes, maximizing the margin is equivalent to minimizing $\|w\|$.

Computing the maximal margin hyperplane (cont.)

- Then the problem becomes:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

s.t.

$$y_i(w \cdot x_i + b) \geq 1, \ \forall i = 1, \ldots, l$$

- The maximal margin is then

$$M^* = \frac{1}{\|w^*\|}$$

- It is a constrained **optimization problem**, with **quadratic convex cost** function and linear constraints

Computing the maximal margin hyperplane (cont.)

- Then the problem becomes:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

s.t.

$$y_i(w \cdot x_i + b) \geq 1, \ \forall i = 1, \ldots, l$$

- The maximal margin is then

$$M^* = \frac{1}{\|w^*\|}$$

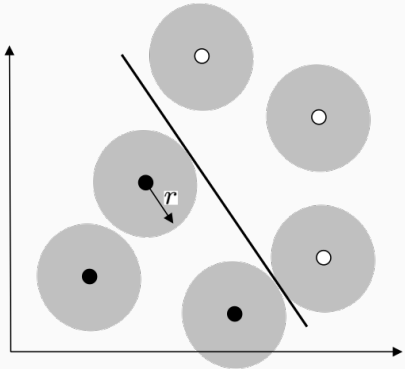
- It is a constrained **optimization problem**, with **quadratic convex cost** function and linear constraints

Quadratic Programming Problem

Robustness of the maximal margin hyperplane

Several robustness arguments justify the maximization of the margin.

In particular, the robustness to pattern noise and parameter noise.



Robustness of the maximal margin hyperplane

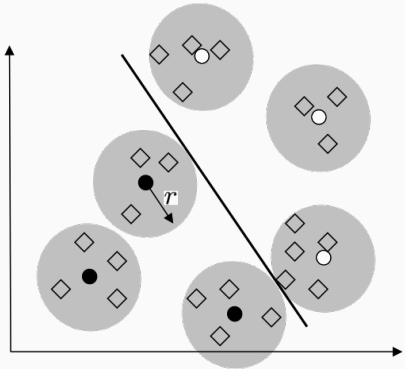
Several robustness arguments justify the maximization of the margin.

In particular, the robustness to pattern noise and parameter noise.

- Pattern noise: imagine that the test vectors ($\diamond$) are generated from the training vectors (x, y) according to

$$(x + \Delta x, y),$$

where $\|\Delta x\| \leq r$;



Robustness of the maximal margin hyperplane

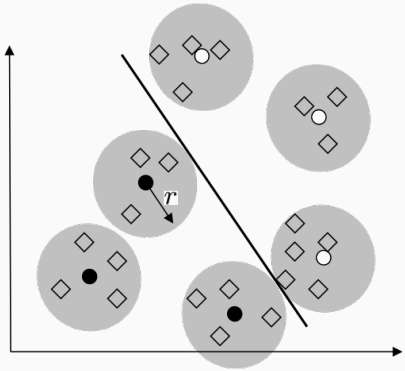
Several robustness arguments justify the maximization of the margin.

In particular, the robustness to pattern noise and parameter noise.

- Pattern noise: imagine that the test vectors ($\diamond$) are generated from the training vectors (x, y) according to

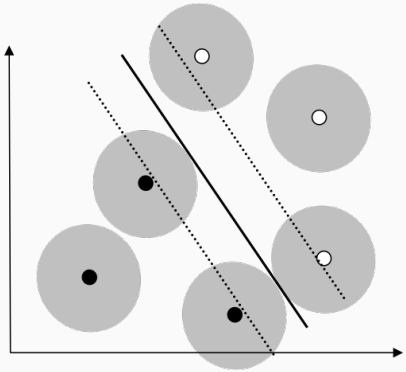
$$(x + \Delta x, y),$$

  where $\|\Delta x\| \leq r$;
- then it is clear that if the margin is greater than r, all the test vectors will be classified correctly.



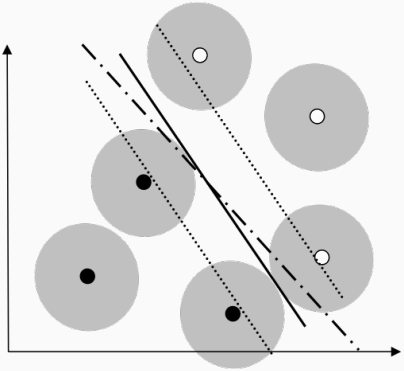
Robustness of the maximal margin hyperplane (cont.)

- Parameter noise: the maximal margin hyperplane maximizes the “thickness” of the stripe in figure.



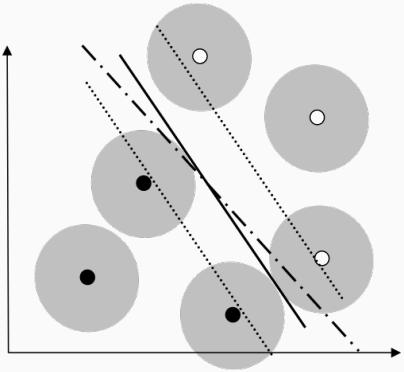
Robustness of the maximal margin hyperplane (cont.)

- Parameter noise: the maximal margin hyperplane maximizes the “thickness” of the stripe in figure.
- Suppose that the separating hyperplane parameters (w and b) are perturbed (due, for example, to numerical errors).



Robustness of the maximal margin hyperplane (cont.)

- Parameter noise: the maximal margin hyperplane maximizes the “thickness” of the stripe in figure.
- Suppose that the separating hyperplane parameters (w and b) are perturbed (due, for example, to numerical errors).
- One can expect that the correct classification will be less sensitive to perturbations of the parameters w and b compared to non-maximal margin hyperplanes.

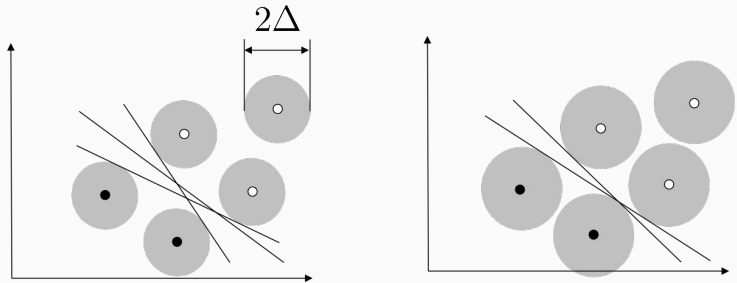


Margin and VC-dimension

- Given a linearly separable training set $\mathcal{S}$ let $\mathcal{H}_{\Delta,\mathcal{S}}$ be the set of the separating hyperplanes having margin such that

$$M \geq \Delta.$$

- As Δ increases, the set $\mathcal{H}_{\Delta,\mathcal{S}}$ becomes smaller.

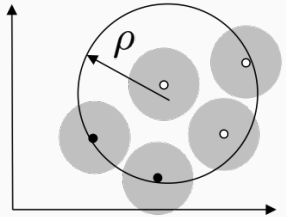


Margin and VC-dimension (cont.)

Theorem

Given $\mathcal{S} = \{(x_1, y_1), \ldots, (x_l, y_l)\}$, let ρ be the radius of the smallest sphere containing the vectors $x_i \in \mathcal{X} \subseteq R^n$. The following inequality holds:

$$VC(\mathcal{H}_{\Delta,\mathcal{S}}) \leq \min\left(\left[\frac{\rho^2}{\Delta^2}\right], n\right) + 1$$

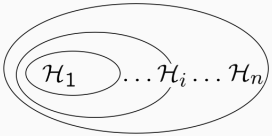


As a consequence, by means of the margin, one can control the VC-dimension and therefore apply the Structural Risk Minimization.

- Indeed, for a given TS, the sequence of canonical hyperplanes $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_i, \ldots\}$ such that

$$\mathcal{H}_i = \left\{ w \cdot x + b : \|w\|^2 \leq k_i \right\}, \; k_1 < k_2 \; \ldots$$

  has non-decreasing VC-dimension and therefore is equipped with a structure.



- Recalling the expression

$$R(h) \leq R_{emp}(h) + \phi\left(d, l, \delta\right)$$

  and observing that, being the data linearly separable, the first term of the second member can always be rendered zero, it is clear that maximizing the margin (i.e. minimizing d) results in minimizing the second member or, in other words, applying the SRM.

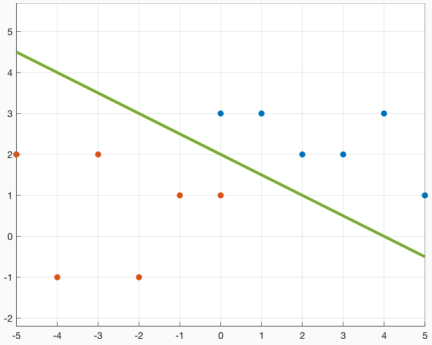
Matlab example

```
% maximal margin separating hyperplane in *primal* formulation
% training set
X=[0 1 2 3 4 5 0 -1 -2 -3 -4 -5;...   %points
   3 3 2 2 3 1 1  1 -1  2  -1 2]
Y=[1 1 1 1 1 1 -1 -1 -1 -1 -1 -1]   % labels

% problem to be solved:
% m i n 0.5*w'*w s.t. Y(i)(w'X(i)+b) >= 1
%  w,b
%
% X=QUADPROG(H,f,A,m)
% solves            min 0.5*x'*H*x + f'*x   subject to:  A*x <= b
%                       x

% Let x be the vector of unknowns: x = [w1, w2, b]'

% cost function
H=[eye(2) [0; 0]; 0 0 0];  f=[0 ; 0; 0];
% constraints
A=[]; b=[];
for ii=1:length(Y)
    b=[b;-1];
    A=[A;-Y(ii)*X(:,ii)' -Y(ii)];
end
% solve quadratic problem
[x,fval,exitflag]=quadprog(H,f,A,b)
% extract w e b
w=x(1:2)
b=x(3)
```

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

s.t.

$$y_i(w \cdot x_i + b) \geq 1,\ \forall i = 1, \ldots, l$$



Dual form

The problem of finding the maximal margin hyperplane admits dual formulation.

- Write the Lagrangian for the problem:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 \\ &\text{s.t.} \\ &y_i(w \cdot x_i + b) \geq 1, \ \forall i = 1, \ldots, l \end{aligned} \quad .$$

- Denoting with $alpha$ the vector of Lagrange multipliers, one gets

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i \left[y_i(w \cdot x_i + b) - 1 \right].$$

- The function $L(w, b, \alpha)$ has to be minimized with respect to w and b and maximized with respect to the multipliers $\alpha_i \geq 0$.

Dual form (cont.)

- Lagrangian:

$$L(w, b, \alpha) = \frac{1}{2}(w \cdot w) - \sum_{i=1}^{l} \alpha_i \left[y_i (w \cdot x_i + b) - 1 \right]$$

- Stationarity conditions

$$\frac{\partial L(w, b, \alpha)}{\partial b} = 0 \quad \Longrightarrow \quad \sum_{i=1}^{l} \alpha_i y_i = 0$$

$$\frac{\partial L(w, b, \alpha)}{\partial w} = 0 \quad \Longrightarrow \quad w = \sum_{i=1}^{l} y_i \alpha_i x_i$$

- Re-write the Lagrangian for substituting the stationarity conditions:

$$L(w, b, \alpha) = \frac{1}{2}(w \cdot w) - \sum_{i=1}^{l} \alpha_i y_i (w \cdot x_i) - b \sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i$$

Dual form (cont.)

- Lagrangian:

$$L(w, b, \alpha) = \frac{1}{2}(w \cdot w) - \sum_{i=1}^{l} \alpha_i \left[y_i (w \cdot x_i + b) - 1 \right]$$

- Stationarity conditions

$$\frac{\partial L(w, b, \alpha)}{\partial b} = 0 \quad \Longrightarrow \quad \sum_{i=1}^{l} \alpha_i y_i = 0$$

$$\frac{\partial L(w, b, \alpha)}{\partial w} = 0 \quad \Longrightarrow \quad w = \sum_{i=1}^{l} y_i \alpha_i x_i$$

- Re-write the Lagrangian for substituting the stationarity conditions:

$$L(w, b, \alpha) = \frac{1}{2}(w \cdot w) - \sum_{i=1}^{l} \alpha_i y_i (w \cdot x_i) - b \sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i$$

Dual form (cont.)

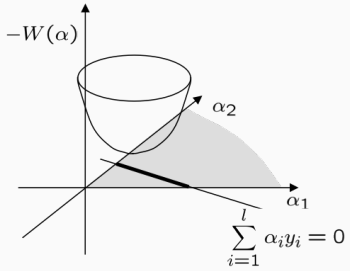
- The equivalent dual problem is

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

s.t. .

$\sum_{i=1}^{l} \alpha_i y_i = 0$

$\alpha_i \geq 0, \ \forall i = 1, \ldots, l$



- It's a **quadratic programming problem** (convex quadratic cost and linear constraints).

Indeed, the cost is clearly quadratic in the decision variables α_i. It is also convex since

$$\sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) = \alpha^\top X^\top X \alpha = \alpha^\top H \alpha$$

where $X = [y_1 x_1 \ldots y_l x_l]$. Thus H is positive semidefinite.

Observations

- Let α^* and w^* be the solutions of the dual problem and the primal problem, respectively. The second of the stationarity conditions states that

$$w^* = \sum_{i=1}^{l} y_i \alpha_i^* x_i$$

  therefore w^* is a linear combination of the vectors of the training set.

- Moreover, it is known (Luenberger and Ye, 2016), that the solution of the primal problem must satisfy the Karush-Kuhn-Tucker conditions:

$$\alpha_i^* \left[y_i (w^* \cdot x_i + b^*) - 1 \right] = 0, \ \forall i = 1, \ldots, l.$$

  Since $[y_i(w^* \cdot x_i + b^*) - 1] = 0$ only for the *active* constraints[1], it follows that the multipliers α_i^* corresponding to non-active constraints, are zero.

[1] constraints that are satisfied as equalities.

Observations

- Let α^* and w^* be the solutions of the dual problem and the primal problem, respectively. The second of the stationarity conditions states that

$$w^* = \sum_{i=1}^{l} y_i \alpha_i^* x_i$$

therefore w^* is a linear combination of the vectors of the training set.

- Moreover, it is known (Luenberger and Ye, 2016), that the solution of the primal problem must satisfy the Karush-Kuhn-Tucker conditions:

$$\alpha_i^* \left[y_i (w^* \cdot x_i + b^*) - 1 \right] = 0, \ \forall i = 1, \ldots, l.$$

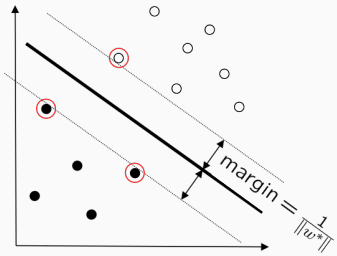
Since $[y_i(w^* \cdot x_i + b^*) - 1] = 0$ only for the *active* constraints[1], it follows that the multipliers α_i^* corresponding to non-active constraints, are zero.

The solution is **sparse**

[1] constraints that are satisfied as equalities.

Support Vectors

- The vectors corresponding to non-zero multipliers are called *support vectors*:
$$SV = \{x_i : \alpha_i^* > 0\}$$
- The vector that specifies (up to the additive term b^*) the maximal margin hyperplane is therefore a linear combination of the support vectors
$$w^* = \sum_{SV} y_i \alpha_i^* x_i$$



SVs correspond to the active constraints:
$$y_i(w^* \cdot x_i + b^*) = 1$$
hence their distance from the hyperplane is equal to the margin.

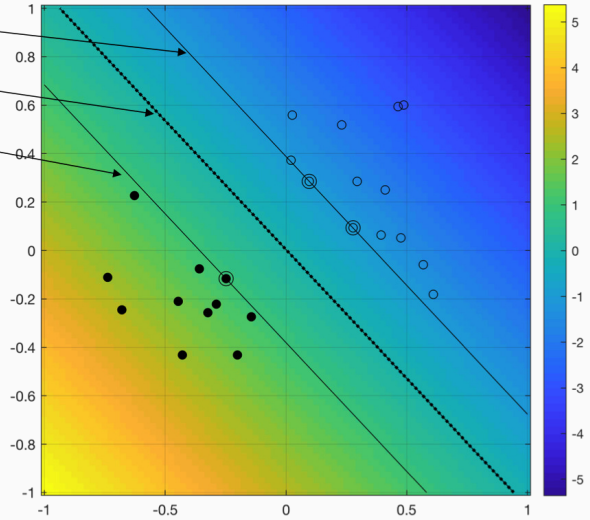
Notice that the solutions depends only on the support vectors. In other words, it does not change if we remove from the training set all the vectors that are not support vectors.

Example

$w^* \cdot x + b^* = -1$

$w^* \cdot x + b^* = 0$

$w^* \cdot x + b^* = 1$



Mechanical interpretation

- Imagine that each of the support vectors exerts a force of direction $y_i \frac{w^*}{\|w^*\|}$ and intensity α_i^*

$$SV = \{x_i : \alpha_i^* > 0\}$$

- Translational equilibrium is guaranteed by the constraints

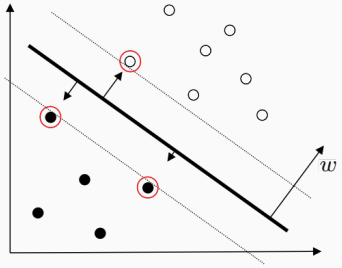
$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

- The total torque is

$$\sum_{SV} r_i \times F_i = \sum_{SV} \left(x_i - y_i \frac{w^*}{\|w^*\|} \right) \times \alpha_i^* y_i \frac{w^*}{\|w^*\|}$$

$$= \underbrace{\sum_{SV} \alpha_i^* y_i x_i}_{w^*} \times \frac{w^*}{\|w^*\|} = 0$$

therefore the rotational equilibrium is guaranteed too.



Decision function in primal and dual form

- The *decision function* allows the classification of a new vector.
- The decision function associated with the optimal hyperplane is

$$h^*(x) = \Theta(w^* \cdot x + b^*)$$

where b^* can be computed by exploiting the KKT conditions corresponding to any support vector x_i:

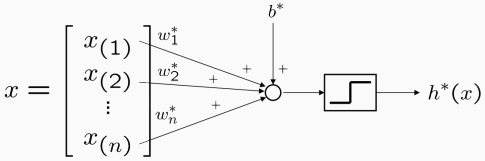
$$y_i(w^* \cdot x_i + b^*) = 1 \qquad \Longrightarrow \qquad b^* = y_i - \sum_{SV} y_j \alpha_i^* (x_j \cdot x_i)$$

- The decision function can also be expressed in terms of the support vectors (dual form). By substituting the expression for w^* in the previous form we obtain

$$\begin{aligned} h^*(x) &= \Theta[(\sum_{SV} y_i \alpha_i^* x_i) \cdot x + b^*] \\ &= \Theta[\sum_{SV} y_i \alpha_i^* (x_i \cdot x) + b^*] \end{aligned}$$

Comparison

The primal form of the SVM can be depicted as:

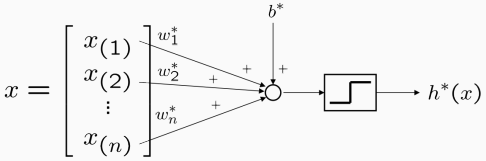


$$h^*(x) = \Theta[(w^* \cdot x) + b^*]$$

where n denotes the dimension of the input space $\mathcal{X}$.

Comparison

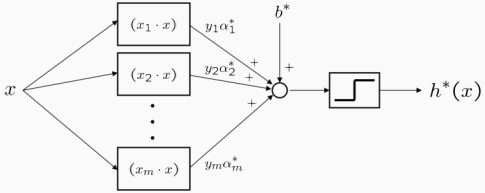
The primal form of the SVM can be depicted as:



$$h^*(x) = \Theta[(w^* \cdot x) + b^*]$$

where n denotes the dimension of the input space $\mathcal{X}$.

The dual form of the SVM can be depicted as:



$$h^*(x) = \Theta[\sum_{SV} y_i \alpha_i^* (x_i \cdot x) + b^*]$$

where m denotes the number of support vectors.

Inner product

- Consider the functional to be minimized (in the dual formulation):

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

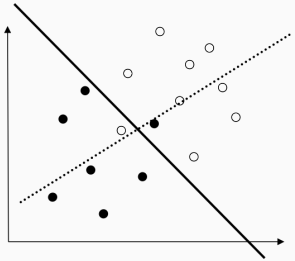
  and the decision function

$$h^*(x) = \Theta[\sum_{SV} y_i \alpha_i^* (x_i \cdot x) + b^*]$$

- It is worth noticing that, in both the functional and the decision function, the training vectors $x_i \in \mathcal{X}$ appear **inside an inner product only**.
- As we will show soon, **this fact is of fundamental importance for extending the methodology to the nonlinear case**.

Optimal hyperplane in the non-separable case

- If the training set is not linearly separable, there is no hyperplane able to classify correctly all the training vectors (i.e. make the empirical risk become zero):
- still, some hyperplanes are better than others.
- Reasonable idea: minimize the number of classification errors and, at the same time, maximize the margin for the points that are classified correctly.



Optimal hyperplane in the non-separable case (cont.)

- Allow constraints violation, by introducing slack variables $\xi_i \geq 0,\ \forall i = 1, \ldots, l$:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i,\ \forall i = 1, \ldots, l$$

- For σ sufficiently small, the number of violated constraints is close to:

$$\sum_{i=1}^{l} \xi_i^{\sigma}$$

- The problem can be expressed as follows ($C > 0$ is a constant):

$$\min \frac{1}{2}(w \cdot w) + C \sum_{i=1}^{l} \xi_i^{\sigma}$$

Optimal hyperplane in the non-separable case (cont.)

- In general, finding a hyperplane that minimizes the number of classification errors for a given training set is a difficult combinatorial problem.
- Therefore, to get computational tractability we choose $\sigma = 1$ and the problem becomes

$$\min \frac{1}{2}(w \cdot w) + C \sum_{i=1}^{l} \xi_i$$

- Thus we search for the hyperplane that minimizes **the sum of the deviations from the constraints** (rather than the number of violated constraints) maximizing, at the same time, the margin for the vectors that are classified correctly;
- such a hyperplane is called the *1-Norm Soft Margin Separating Hyperplane*.

Optimal hyperplane in the non-separable case (cont.)

- The whole optimization problems becomes:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i$$

s.t. .

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i,\ \forall i = 1, \ldots, l$$

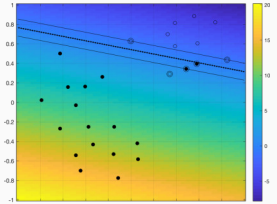
$$\xi_i \geq 0,\ \forall i = 1, \ldots, l$$

- C is a *regularization constant*, that controls the trade-off between the constraint violation and the capacity of the hypothesis set.

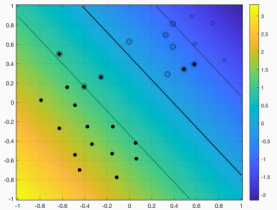
The role of C

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i$$

C big: the first term is less important hence we get solutions with **few classification errors but small margin** (because w can be relatively high)



C small: the first term is more important hence we get solutions with **high margin** (because w as to be relatively small) **but many classification errors**.



Dual formulation

- The dual formulation is

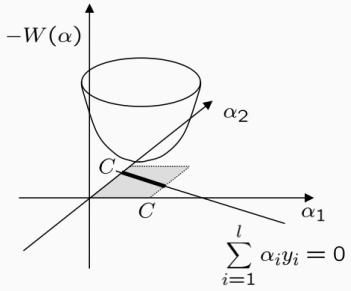
$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

s.t.

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, l$$

- Same as the separable case, except for the constraints on the multipliers (also called *box constraints*).



It's again a **quadratic programming problem** (convex quadratic cost and linear constraints).

Bonded and unbounded support vectors

- As in the separable case, the solution is sparse
- The support vectors are those whose multipliers are different from zero.
- Bounded support vectors: those whose multipliers satisfy

$$\alpha_i^* = C.$$

- Unbounded support vectors: those whose multipliers satisfy

$$0 < \alpha_i^* < C.$$

- The decision function is the same as in the separable case:

$$h^*(x) = \Theta[\sum_{SV} y_i \alpha_i^* (x_i \cdot x) + b^*]$$

- b^* is computed as in the separable case from any **unbounded** support vector.

Soft margin bound

Bounds exist that justify from a statistical point of view the Soft Margin Separating Hyperplane, for example

Soft Margin Bound (Schölkopf and Smola, 2001)

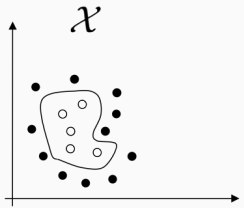
Given the set of functions $h(x) = \Theta(w \cdot x)$, where $\|w\| \leq \Delta^{-2}$, $\|x\| \leq \rho$.
Let ν be the fraction of members of the training set with margin less than $\lambda/\|w\|$. Then, independent of the distribution $F(x, y)$, with probability $1 - \delta$, the probability of misclassification of a test vector is less than

$$\nu + \sqrt{\frac{c}{l}\left(\frac{\rho^2}{\lambda^2\Delta^2}\ln^2 l + \ln(1/\delta)\right)}$$

where c is a constant.

Feature mapping

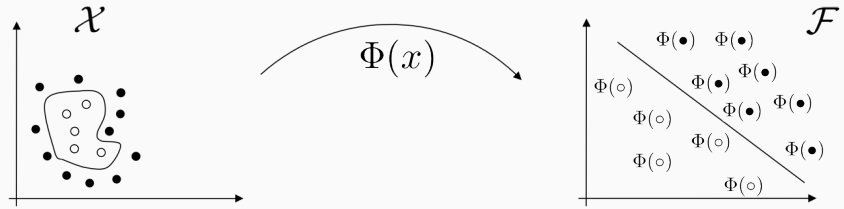
- In many cases, being the dependence inherently nonlinear, the soft margin is not adequate. In such cases, the regularity of the data cannot be captured by any linear function, as in the following example:



- It may exist, however, a nonlinear transformation $\Phi(x) : \mathcal{X} \longrightarrow \mathcal{F}$ that renders the transformed data linearly separable.

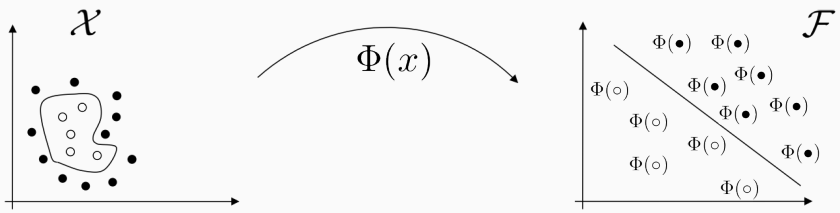
Feature mapping

- In many cases, being the dependence inherently nonlinear, the soft margin is not adequate. In such cases, the regularity of the data cannot be captured by any linear function, as in the following example:



- It may exist, however, a nonlinear transformation $\Phi(x) : \mathcal{X} \longrightarrow \mathcal{F}$ that renders the transformed data linearly separable.
- Φ is called *feature map*, $\mathcal{X}$ is the *input space* (or *attribute space*) and $\mathcal{F}$ is the *feature space*.

Feature mapping (cont.)



Therefore one can conceive a two-steps algorithm:

- Nonlinear transformation (feature mapping)
- Linear classification in the feature space

In other words, find a separating hyperplane in the feature space starting from the transformed training set:

$$\mathcal{S} = \{(\Phi(x_1), y_1), \ldots, (\Phi(x_l), y_l)\}$$

where the pair $(\Phi(x_i), y_i) \in \mathcal{F} \times \mathcal{Y}$.

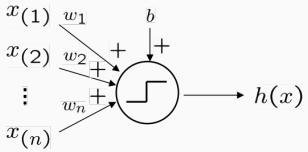
The Rosenblatt's perceptron

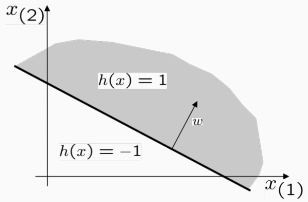
- The perceptron was proposed by Rosenblatt in the end of thh 50s (Rosenblatt, 1958).
- It is based on the neuron model of Mc-Culloch and Pitts (McCulloch and Pitts, 1943), which is activated ("fires") if the weighted sum of its inputs is above a threshold:

$$h(x) = \Theta(w \cdot x + b), \ w \in \mathbb{R}^n, b \in \mathbb{R}.$$

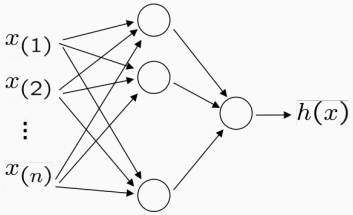
- A decision function of this type splits $\mathbb{R}^n$ into two regions divided by the hyperplane of equation

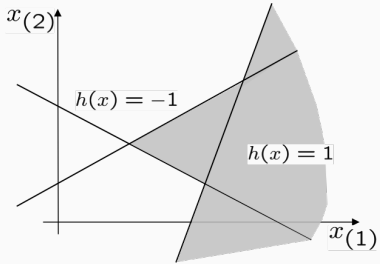
$$w \cdot x + b = 0.$$





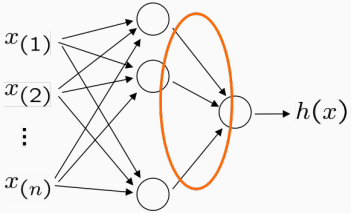
The Rosenblatt's perceptron (cont.)





- The perceptron consists of many neurons connected and organized in several layers.
- The last layer consists of a single neuron.
- The perceptron splits the input space into two regions (not connected, in general) separated by a piecewise linear surface

The Rosenblatt's perceptron (cont.)



At that time it was not clear how to choose the weights of the hidden layers. Solution proposed:

- fix randomly the weights of the hidden layers;
- modify during training the weights of the **last neuron only**.

That is equivalent to:

- defining a (nonlinear) map $z = \Phi(x)$;
- training a single neuron over a new training set, consisting of the transformed vectors $\{(z_1, y_1), \ldots, (z_l, y_l)\}$.

Training algorithm

Algorithm Perceptron's training algorithm (for $b = 0$)

Input: A training set $\{z_i, y_i\}$, $i = 1 \dots l$
Output: The weight vector w^*

Initialize the weight vector $w \leftarrow 0$.
repeat
    for all pairs z, y in the training set **do**
        if $y(w \cdot z) \geq 0$ i.e. the sample is correctly classified **then**
            leave the weight vector unchanged.
        else
            update the weight vector as $w \leftarrow w + \eta y z, \qquad \eta > 0.$
        end if
    end for
until all the examples are correctly classified.
Return w

If the training set is linearly separable, the algorithm provides a separating hyperplane in a finite number of steps (Novikoff, 1962).

Theorem (First theorem about the perceptrons (Novikoff, 1962))

If the following conditions hold:

1. *the norm of the training vectors z is bounded by some constant R:*

$$\|z\| \leq R;$$

2. *the training set is separable with margin ρ:*

$$\sup_{w} \min_{i} y_i(z_i \cdot w) \geq \rho;$$

3. *the training sequence is presented to the perceptron a sufficient number of times,*

then after at most

$$N \leq \left[\frac{R^2}{\rho^2}\right]$$

corrections, a hyperplane that separates the training data will be constructed.

Training algorithm (cont.)

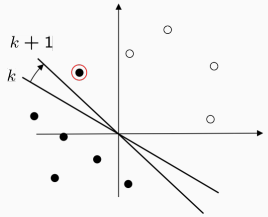
The update rule can be explained as follows.

Let denote (with a slight abuse of notation) with y_k, z_k the pair presented to the perceptron at the k-th iteration. Let also denote by $w(k)$ the weight vector at the k-th iteration.

When presented **the same sample** right after the k-th iteration, the perceptron provides the output:

$$\begin{aligned} y_k(w(k+1) \cdot z_k) &= y_k[(w(k) + \eta y_k z_k) \cdot z_k] \\ &= y_k(w(k) \cdot z_k) + \eta z_k^2 \end{aligned}$$

Notice that the term ηz_k^2 is strictly positive, thus the update rule "pushes" the hyperplane toward the right direction.



Observations

- The hyperplane found by the perceptron **does not maximize any margin**;
- **the solution depends on the order** the vectors are shown to the perceptron;
- **when the training set is not linearly separable, the algorithm does not converge**, but enters an endless loop in which the weights are modified infinitely many times;
- the update rule

$$w(k+1) = w(k) + \eta y_k z_k$$

  is such that **the solution (the weight vector) is a linear combination of the training vectors**:

$$w^* = \sum_{i=1}^{l} \alpha_i y_i z_i;$$

- moreover, α_i is **proportional to the number of corrections** performed during training, as a consequence of observing a misclassified z_i.

Dual formulation

- The decision function has a dual expression:

$$h(z) = \Theta\left(\sum_{i=1}^{l} \alpha_i y_i (z_i \cdot z)\right),$$

  and there exists a dual formulation of the training algorithm.

Dual formulation (cont.)

Algorithm Perceptron's training algorithm, dual form (for $b = 0$)

Input: A training set $\{z_i, y_i\}$, $i = 1 \dots l$

Output: The weight vector α^*

1: Initialize the weight vector $\alpha \leftarrow 0$.
2: **repeat**
3: **for all** pairs z_k, y_k , $k = 1 \dots l$ **do**
4: **if** $y_k \left(\sum_{i=1}^{l} \alpha_i y_i (z_i \cdot z_k) \right) \geq 0$ i.e. the k-th example is classified correctly **then**
5: leave the weight vector unchanged.
6: **else**
7: update the k-th component of the weight vector as $\alpha_k \leftarrow \alpha_k + 1$.
8: **end if**
9: **end for**
10: **until** all the examples are correctly classified.
11: Return α

Potential functions

In 1964 Aizerman, Braverman e Rozonoer (Aizerman et al., 1964) proposed the potential functions method (and studied its convergence properties):

- assume that a separating surface exists, corresponding to the zero-level surface of:

$$\Psi(x) = \sum_{1}^{N} c_i \phi_i(x)$$

  where $\phi_i : \mathbb{R}^n \longrightarrow \mathbb{R}$.

- Define a potential function

$$K(x, x') = \sum_{1}^{N} \phi_i(x)\phi_i(x');$$

- use a decision function and a training algorithm corresponding to that of the perceptron in dual formulation.

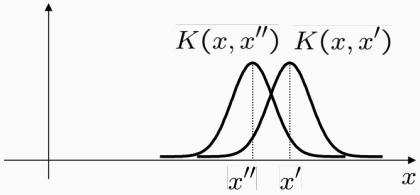
The potential function may be interpreted as an inner product in a feature space of dimension N implied by the feature mapping $\Phi(x) = [\phi_1(x), \ldots \phi_N(x)]^\top$.

Potential functions (cont.)

- In particular, the decision function is

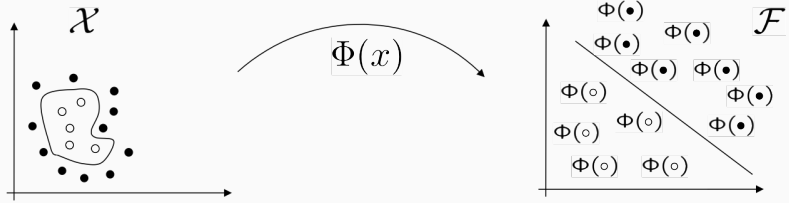
$$h(x) = \Theta\left(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x)\right).$$

- The training algorithm is that of the perceptron (in its dual formulation) where the inner product $(x_i \cdot x_j)$ is replaced by the potential $K(x_i, x_j)$.



Potential functions (cont.)

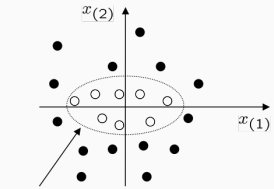
- The potential functions method can be considered **a nonlinear generalization of the single layer perceptron** because using the same algorithm it allows constructing a decision function **in the feature space**.



- The subsequent step would have been endowing this method with the maximal margin, that Vapnik and Chervonenkis had discovered roughly in the same period.
- That, however, was done only in 1992 (Boser et al., 1992).

Example of feature mapping

In the example below, there exists a separating curve between the points of the two classes. Precisely, the curve is an ellipse.

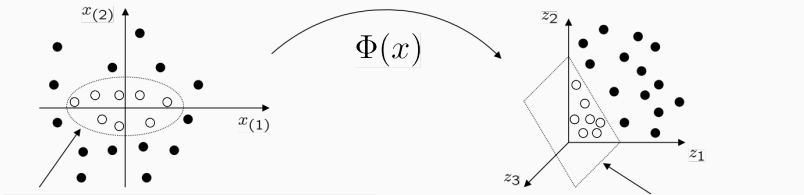


$$ax_{(1)}^2 + bx_{(2)}^2 + cx_{(1)}x_{(2)} = k$$

- take the map $\Phi(x) = \Phi(x_{(1)}, x_{(2)}) = (z_1, z_2, z_3) = (x_{(1)}^2, x_{(2)}^2, x_{(1)}x_{(2)})$

Example of feature mapping

In the example below, there exists a separating curve between the points of the two classes. Precisely, the curve is an ellipse.



- take the map $\Phi(x) = \Phi(x_{(1)}, x_{(2)}) = (z_1, z_2, z_3) = (x_{(1)}^2, x_{(2)}^2, x_{(1)}x_{(2)})$
- The ellipse in two dimension is mapped to a plane in three dimension.
- The coefficient c in this case equals zero because the ellipse axes are parallel to the coordinate axes. Therefore the plane is orthogonal to the plane (z_1, z_2)

- In the previous example, the features are all the monomials of second degree that can be obtained by multiplying the components of the input vector:

$$\Phi(x) = \Phi(x_{(1)}, x_{(2)}) = (z_1, z_2, z_3) = (x_{(1)}^2, x_{(2)}^2, x_{(1)}x_{(2)}).$$

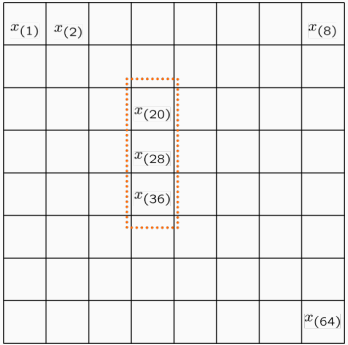
- A hyperplane in the feature space corresponds, in the input space, to a level surface of a polynomial:

$$az_1 + bz_2 + cz_3 = k \qquad \longleftrightarrow \qquad ax_{(1)}^2 + bx_{(2)}^2 + cx_{(1)}x_{(2)} = k.$$

- Such a machine is called *polynomial classifier*.

Polynomial machines (cont.)

- Why using monomials as features?
- Because a product between two or more components of the input vector can be considered a logical AND.



- In the 8*8 image on the left, the monomial $x_{(20)}x_{(28)}x_{(36)}$ will have a high value only if each of the three pixels has high intensity. Therefore, this feature codifies the condition “each of the pixels 20,28,36 has high intensity.”

Polynomial machines (cont.)

- If $x \in \mathbb{R}^n$, the number of monomials of degree d obtained by multiplying the components of x is:

$$\binom{n+d-1}{d}.$$

- The number of monomials of degree $\leq d$ is:

$$\binom{n+d}{d}.$$

- Realistic example: 16*16 image, monomial of fifth degree:

$$\binom{256+5-1}{5} \approx 10^{10},$$

thus, serious computational difficulties arise in solving an optimization problem is such a huge space.

Two requirements

Using a nonlinear mapping (feature mapping) can be effective if we can guarantee:

- good generalization capability;
- computational tractability.

The former is obtained by **maximizing the margin** (the bound for the risk does not depend on the dimensionality of the space).

The latter, by means of an **implicit transformation** (“kernel trick”).

Implicit transformation

- Due to the transformation, the decision function takes the form

$$h(x) = \Theta(w \cdot \Phi(x) + b) = \Theta(\sum_{i=1}^{N} w_i \Phi_i(x) + b)$$

  where N is is the dimensionality of the feature space ($w \in \mathbb{R}^N$).

- The primal problem is an optimization problem in a space of dimension N

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \xi_i \\ &\text{s.t.} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad . \\ &y_i(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i, \ \forall i = 1, \ldots, l \\ &\xi_i \geq 0, \ \forall i = 1, \ldots, l \end{aligned}$$

- For large N, computational difficulties arise, both for solving the optimization problem and for evaluating the decision function.

Implicit transformation (cont.)

- On the other hand, the dual formulation is:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j))$$

s.t.

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, l$$

- The decision function is:

$$h(x) = \Theta[\sum_{i=1}^{l} y_i \alpha_i (\Phi(x_i) \cdot \Phi(x)) + b]$$

Implicit transformation (cont.)

- On the other hand, the dual formulation is:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j))$$

s.t.

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, l$$

- The decision function is:

$$h(x) = \Theta[\sum_{i=1}^{l} y_i \alpha_i (\Phi(x_i) \cdot \Phi(x)) + b]$$

- Observe that the input vectors transformed according to the map $\Phi(\cdot)$ appear **in dot products only**.
- Should these products can be computed directly (without computing vectors of $\mathcal{F}$) then the computational difficulties can be avoided

Kernels

- A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that

$$k(x, x') = \Phi(x) \cdot \Phi(x'), \quad \forall x, x' \in \mathcal{X}$$

  where Φ is a map from $\mathcal{X}$ to a space $\mathcal{F}$ (dot-product space).
- The kernel defines **implicitly** the transformation Φ and **is all we need to find the optimal hyperplane** in $\mathcal{F}$.
- Thanks to the kernel, we avoid the explicit computation of the features (i.e. the vectors $\Phi(x)$).
- Sometimes that techniques is called the "kernel trick".

Kernels (cont.)

A classical example:

- Take $\mathcal{X} = \mathbb{R}^2$
- Let $x = (x_{(1)}, x_{(2)})$ and $z = (z_{(1)}, z_{(2)})$ and define a kernel as

$$\begin{aligned}(x \cdot z)^2 &= [(x_{(1)}, x_{(2)}) \cdot (z_{(1)}, z_{(2)})]^2 \\ &= [x_{(1)} z_{(1)} + x_{(2)} z_{(2)}]^2 \\ &= x_{(1)}^2 z_{(1)}^2 + 2 x_{(1)} x_{(2)} z_{(1)} z_{(2)} + x_{(2)}^2 z_{(2)}^2 \\ &= (x_{(1)}^2, \sqrt{2} x_{(1)} x_{(2)}, x_{(2)}^2) \cdot (z_{(1)}^2, \sqrt{2} z_{(1)} z_{(2)}, z_{(2)}^2) \\ &\doteq \Phi(x) \cdot \Phi(z)\end{aligned}$$

- Thus the map $\Phi : \mathbb{R}^2 \longrightarrow \mathbb{R}^3$ computes **monomial features of degree two**:

$$\Phi : (x_{(1)}, x_{(2)}) \quad \longrightarrow \quad (x_{(1)}^2, \sqrt{2} x_{(1)} x_{(2)}, x_{(2)}^2)$$

- Clearly in that case $\mathcal{F} = \mathbb{R}^3$.

Kernels (cont.)

We can thus formulate the optimization problem (whose solution is the optimal hyperplane in the feature space) in terms of the kernel $k(x, x')$:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

s.t.

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, l$$

.

The decision function is

$$h(x) = \Theta[\sum_{i=1}^{l} y_i \alpha_i k(x_i, x) + b]$$

Note: the matrix $[K_{ij}] = [k(x_i, x_j)], \ i, j = 1, \ldots, l$ is called the *Gram matrix* and is associated to the quadratic form appearing in the cost functional.

Characterization

Under what conditions a given $k(\cdot, \cdot)$ is a kernel? Roughly speaking, the function must be a dot product in some feature space.

Clearly, since the dot product is commutative, a necessary condition is $k(\cdot, \cdot)$ being symmetric:

$$k(x, z) = k(z, x), \ \forall x, z \in \mathcal{X}$$

A necessary and sufficient condition, for **finite input space** $\mathcal{X}$ can be stated as follows.

- Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$.
- Given the symmetric function $k(\cdot, \cdot)$, we can compute the (symmetric) Gram matrix

$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ \vdots & \vdots & \vdots & \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{bmatrix} \in \mathbb{R}^{n \times n}$$

- It can be shown that $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \longrightarrow \mathcal{F}$ is a kernel if and only if K is positive semidefinite.

Characterization (cont.)

Indeed, being K symmetric, it can be factorized as:

$$K = V^\top \Lambda V,$$

where $\Lambda = \text{diag}\{\lambda_i\}$. Thus

$$k(x_i, x_j) = K_{ij} = \sum_{p=1}^{n} (V^T)_{ip} \Lambda_{pp} V_{pj} = \sum_{p=1}^{n} V_{pi} \lambda_p V_{pj}.$$

If K is positive semidefinite (i.e. $\lambda_i \geq 0, \forall i$) the latter can be written as

$$\sum_{p=1}^{n} V_{pi} \lambda_p V_{pj} = (\sqrt{\lambda_1} V_{1i}, \sqrt{\lambda_2} V_{2i}, \ldots, \sqrt{\lambda_n} V_{ni}) \cdot (\sqrt{\lambda_1} V_{1j}, \sqrt{\lambda_2} V_{2j}, \ldots, \sqrt{\lambda_n} V_{nj})$$

which is a dot product between features $\Phi(x_i) \cdot \Phi(x_j)$, provided that the feature map is defined as:

$$\Phi(x_i) = (\sqrt{\lambda_1} V_{1i}, \sqrt{\lambda_2} V_{2i}, \ldots, \sqrt{\lambda_n} V_{ni}) \qquad i = 1, \cdots, n.$$

Theorem (Mercer's theorem)

To guarantee that the symmetric function $k(x, z) : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$, $k(x, z) \in \mathcal{L}_2$ *has an expansion*

$$k(x, z) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i(z)$$

with positive coefficients $\lambda_i > 0$ *(i.e.* $k(x, z)$ *describes an inner product in some feature space) it is necessary and sufficient that the condition*

$$\iint_{\mathcal{X} \times \mathcal{X}} k(x, z) g(x) g(z) dx dz > 0$$

be satisfied $\forall g : g \in \mathcal{L}_2,\ g \neq 0.$

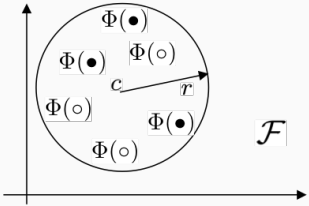
It can be shown that the condition of the Mercer's Theorem is equivalent to the Gram matrix formed from any finite subset $\{x_1, x_2, \ldots, x_p\}$ of the set $\mathcal{X}$ being positive semidefinite (which, by the way, guarantees the convexity of the optimization problem.)

Computing the radius of smallest sphere containing the features

We have seen that the radius of smallest sphere containing the features is related to the VC-dimension. It can be computed as follows.

- Write the Lagrangian for the problem:

$$\min_{r,c} \; r^2$$
$$\text{s.t.} \quad .$$
$$r^2 - \|\Phi(x_i) - c\|_2^2 \geq 0, \; \forall i = 1, \ldots, l$$



- Denoting with $\lambda_i \geq 0$ the Lagrange multipliers, one gets

$$L(r, c, \lambda) = r^2 - \sum_{i=1}^{l} \lambda_i (r^2 - \|\Phi(x_i) - c\|_2^2)$$

- The function $L(r, c, \lambda)$ has to be minimized with respect to r and c and maximized with respect to the multipliers $\lambda_i \geq 0$.

Computing the radius of smallest sphere containing the features (cont.)

- Lagrangian:

$$L(r, c, \lambda) = r^2 - \sum_{i=1}^{l} \lambda_i (r^2 - \|\Phi(x_i) - c\|_2^2)$$

- Stationarity conditions

$$\frac{\partial L(r, c, \lambda)}{\partial r} = 0 \quad \Longrightarrow \quad \sum_{i=1}^{l} \lambda_i = 1$$

$$\frac{\partial L(r, c, \lambda)}{\partial c} = 0 \quad \Longrightarrow \quad c = \sum_{i=1}^{l} \lambda_i \Phi(x_i)$$

- Re-write the Lagrangian for substituting the stationarity conditions:

$$L(r, c, \lambda) = r^2 \left(1 - \sum_{i=1}^{l} \lambda_i\right) + \sum_{i=1}^{l} \lambda_i [(\Phi(x_i) - c) \cdot (\Phi(x_i) - c)]$$

Computing the radius of smallest sphere containing the features (cont.)

- Lagrangian:

$$L(r, c, \lambda) = r^2 - \sum_{i=1}^{l} \lambda_i (r^2 - \|\Phi(x_i) - c\|_2^2)$$

- Stationarity conditions

$$\frac{\partial L(r, c, \lambda)}{\partial r} = 0 \implies \sum_{i=1}^{l} \lambda_i = 1$$

$$\frac{\partial L(r, c, \lambda)}{\partial c} = 0 \implies c = \sum_{i=1}^{l} \lambda_i \Phi(x_i)$$

- Re-write the Lagrangian for substituting the stationarity conditions:

$$L(r, c, \lambda) = r^2 \left(1 - \sum_{i=1}^{l} \lambda_i\right) + \sum_{i=1}^{l} \lambda_i [(\Phi(x_i) - c)\dot{(}\Phi(x_i) - c)]$$

Computing the radius of smallest sphere containing the features (cont.)

- The equivalent dual problem is

$$\max_{\lambda} W(\lambda) = \sum_{i=1}^{l} \lambda_i (\Phi(x_i) \cdot \Phi(x_i)) - \sum_{i,j} \lambda_i \lambda_j (\Phi(x_i) \cdot \Phi(x_j))$$

s.t.

$$\sum_{i=1}^{l} \lambda_i = 1$$

$$\lambda_i \geq 0, \ \forall i = 1, \ldots, l$$

  and can be solved using the kernel trick.

- Then, the radius can be computed by choosing an active constraint (say i):

$$\begin{aligned} r^2 &= (\Phi(x_i) - c) \cdot (\Phi(x_i) - c) \\ &= \Phi(x_i) \cdot \Phi(x_i) - 2c \cdot \Phi(x_i) + c \cdot c \\ &= k(x_i, x_i) - 2 \sum_{j=1}^{l} \lambda_j k(x_j, x_i) + \sum_{i,j=1}^{l} \lambda_i \lambda_j k(x_i, x_j) \end{aligned}$$

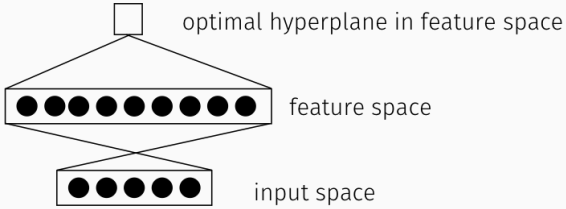
Matlab example

The computation may be performed with a few lines of Matlab code:

```
% compute the radius of smallest sphere enclosing the features

load X % load the training set

l=size(X,2) % number of vectors of the training set
n=size(X,1) % dimension of input space

% prepare the matrices for the optimization
k=kernel(X,X); %compute the Gram matrix
f=-diag(k);
H=2*k;
A=-eye(l)
b=zeros(l,1)
Aeq=ones(1,l);
beq=1;

[lambda,h] = quadprog(H,f,A,b,Aeq,beq)

% get active constraints (corresponding to non zero Lagrange multipliers)
I=find(lambda>1e-5);

% compute r (a numerically preferable solution  would be to take the
% average value across all the active
% constraints)
r=sqrt(k(I(1),I(1))-2*k(I(1),:)*lambda + lambda'*k*lambda)
```

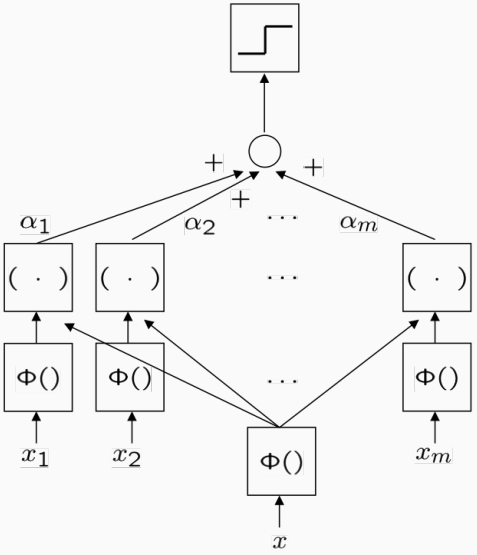
Support Vector Machines

- SVMs map input vectors x into a high dimensional space $\mathcal{F}$;
- the map is fixed and implicitly defined by the kernel;
- then, they found the optimal separating hyperplane in that space.

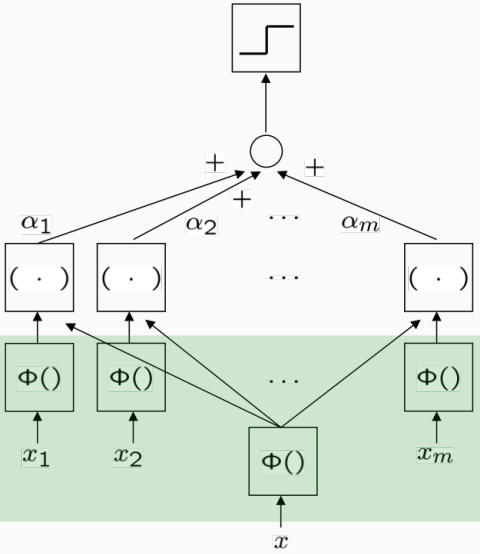


- The **generalization capability** is guaranteed by maximizing the margin;
- the **computational tractability** guaranteed by the kernel trick.

Architecture



Architecture



Transformation performed implicitly by means of the kernel

Generalization and number of support vectors

A common technique for estimating the generalization capability of a learning machine as a function of its hyperparameters is the “leave-one-out” method:

- remove x_i from the training set;
- perform the training;
- test the obtained classifier on the removed pattern;
- repeat $\forall i$;
- the expected generalization error is the average of the obtained errors.

For the particular structure of the SVM, **the outcome of the leave-one-out is related to the number of support vectors**.

Generalization and number of support vectors (cont.)

In an SVM, when a training pattern is removed the following can happen:

1. the pattern was not a SV of the "complete" classifier: in this case the solution (the separating hyperplane) does not change and the pattern is classified correctly (error = 0);
2. the pattern was a SV but the solution does not change (error = 0);
3. the pattern was a SV and the solution changes; in that case it may happen that the pattern is misclassified (error = 1).

In the worst case, every removed SV leads to a misclassification. As a consequence, $\frac{\#SV}{l}$ is an index of the generalization capability. More precisely:

$$\mathbb{E}(P_{err}) \leq \mathbb{E}\left(\frac{\#SV}{l}\right).$$

Examples of kernels

- *Homogeneous polynomial*:

$$k(x, x') = (x \cdot x')^d,$$

in which the features are all the monomials of degree d obtained by multiplying the (weighted) components of the input vector.

- *Non-homogeneous polynomial*:

$$k(x, x') = (x \cdot x' + c)^d,$$

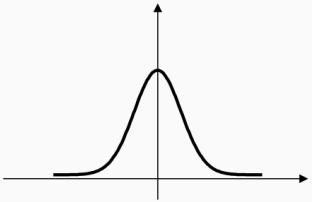
in which the features are all the monomials of degree $\leq d$ obtained by multiplying the (weighted) components of the input vector.

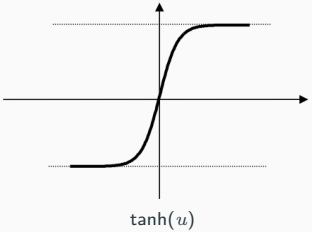
- *Gaussian*:

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{c}\right).$$

- *Sigmoidal* (it does not satisfies the Mercer's conditions for all the values of γ and θ)

$$k(x, x') = \tanh(\gamma(x \cdot x') + \theta)$$





Radial basis kernels

- The Gaussian kernel is an example of *radial basis kernel*:

$$k(x, x') = f\left(d(x, x')\right)$$

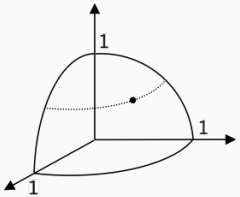
  where $d(x, x')$ is a metric and $f : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$.

- Radial basis kernels are translation invariant:

$$k(x - x_0, x' - x_0) = k(x, x')$$

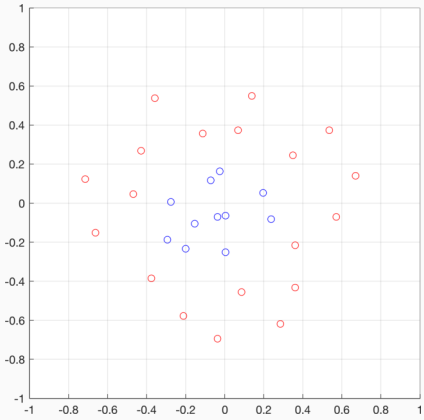
- In addition, the Gaussian kernel enjoys the properties:
  - image vectors have unitary length, i.e. $k(x, x) = 1 \; \forall x$;
  - they are located in the same orthant, i.e. $k(x, x') > 0 \; \forall x, x'$;
  - the feature space has infinite dimension:

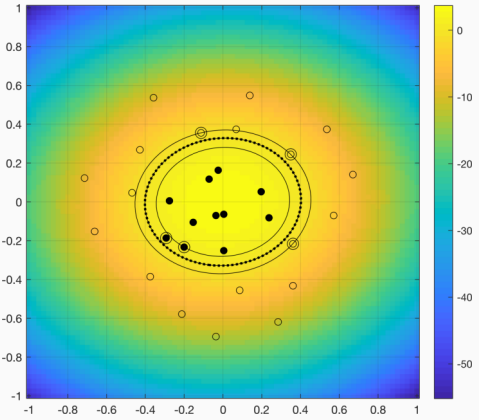
$$[k(x_i, x_j)] = [\Phi(x_i) \cdot \Phi(x_j)] \text{ is full rank } \forall \{x_1, x_2, \ldots, x_m\}, \; \forall m.$$



Example 1

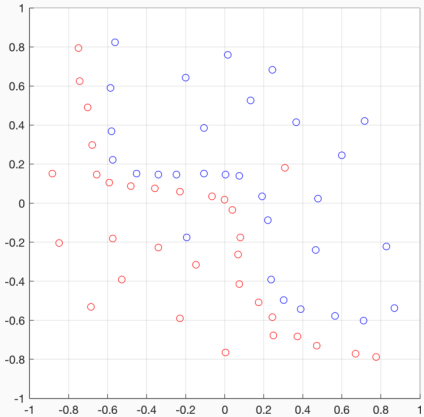
Homogeneous polynomial kernel of second degree: $k(x, x') = (x \cdot x')^2$

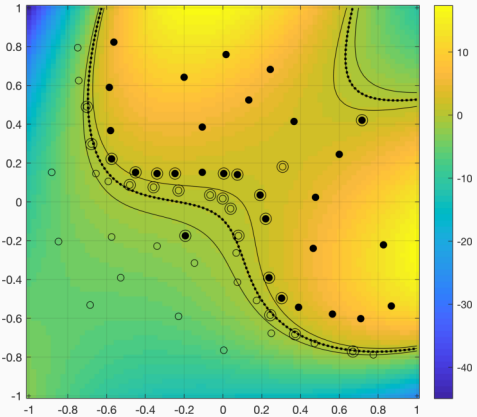




Example 2

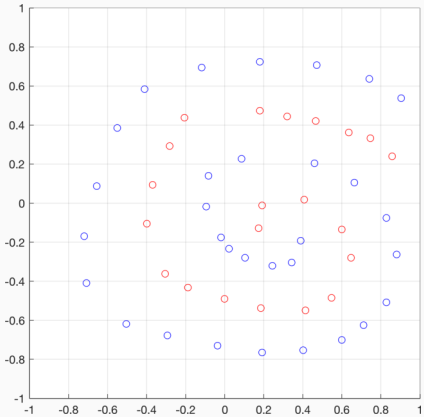
Non-homogeneous polynomial kernel of fourth degree: $k(x, x') = (x \cdot x' + 1)^4$

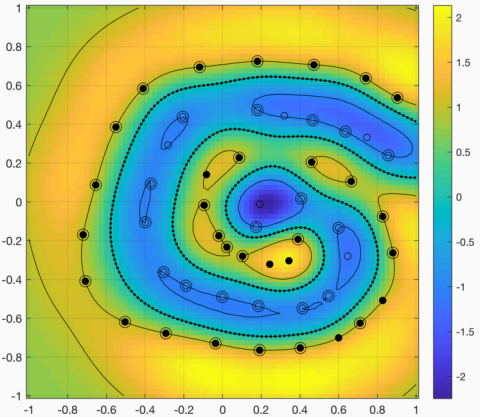




Example 3

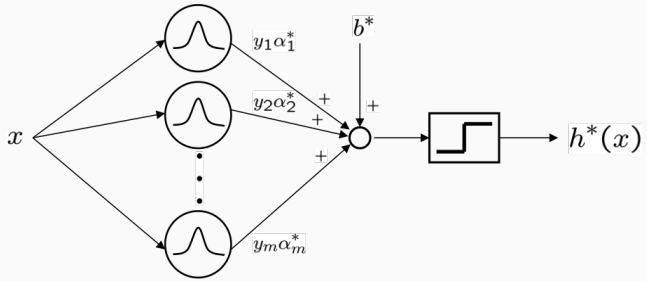
Gaussian kernel: $k(x, x') = \exp\left(-\frac{\|x-x'\|^2}{\sigma^2}\right)$





A support vector machine with Gaussian kernel is equivalent to a radial basis function network with Gaussian activation function. However:

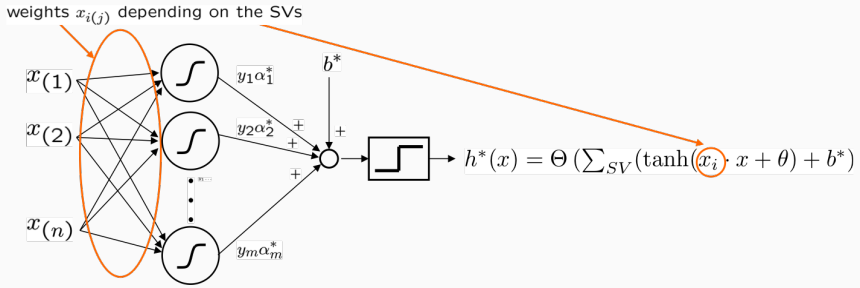
- The actual number of neurons (the SVs) is automatically chosen by the algorithm.
- The positioning of the “centers” is automatically performed by the algorithm.



SVMs and sigmoidal neural networks

A support vector machine with sigmoidal kernel is equivalent to a single-layer sigmoidal neural network. However:

- The actual number of neurons (the SVs) is automatically chosen by the algorithm.
- The training is a convex problem.

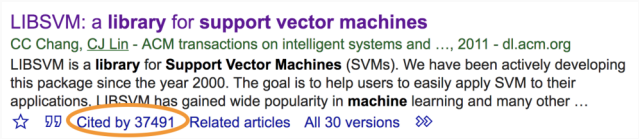


Extensions and software library

There exist many extensions (Abe, 2010) of the basic methods:

- different norms for penalizing the constraint violation;
- weighted versions for dealing with unbalanced data;
- different and more significant parameterizations for the regularization (ν-SVM);
- formulations that lead to linear programming problems;
- various extensions to the multi-class problem.

A well established software tool is LIBSVM (Chang and Lin, 2011).

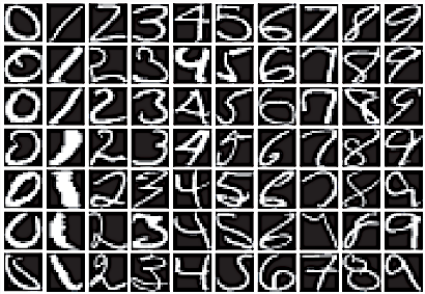


Handwritten digits recognition example

Example from (Vapnik, 2000).

- Database: U.S. Postal Service
- 7300 training examples
- 2000 test examples
- Each example is 16×16 image, thus the input space dimension is 256
- It's a ten-class problem faced by constructing ten classifiers, each one separating one class from the rest (*one-against-all* approach).
- Polynomial kernel, experiments with different values for degree d:

$$k(x, x_i) = \left(\frac{x \cdot x_i}{256} \right)^d, \quad d = 1, \ldots, 7$$



Handwritten digits recognition example (cont.)

Degree of polynomial	Dimensionality of feature space (n)	Support vectors (average value over 10 classifiers)	Raw error (%)
1	256	282	8.9
2	≈ 33000	227	4.7
3	$\approx 10^6$	274	4.0
4	$\approx 10^9$	321	4.2
5	$\approx 10^{12}$	374	4.3
6	$\approx 10^{14}$	377	4.5
7	$\approx 10^{16}$	422	4.5

Choice of the degree

- For $d \geq 3$, the training set turns out to be separable in the feature space, thus the empirical risk is zero;
- as a consequence, the structural risk (the sum of the empirical risk and the confidence) can be minimized by minimizing the VC-dimension;
- the VC-dimension (an upper bound of) can be estimated by using:

$$VC \leq \min\left(\left[\frac{\rho^2}{\Delta^2}\right], n\right) + 1$$

  where ρ is the radius of the minimum enclosing sphere, Δ is the margin and n is the dimensionality of the input space. Notice that all those quantities are known after the training of the SVM.
- Choice of the degree:
  1. Train different SVMs using kernels of degree $d = 1, \ldots, 7$;
  2. estimate the VC-dimension for each SVM;
  3. choose the one with the smallest estimate of the VC-dimension.
- Another possibility, widely used in practice, is to choose the degree (or, in general, the hyperparameters), using cross-validation.

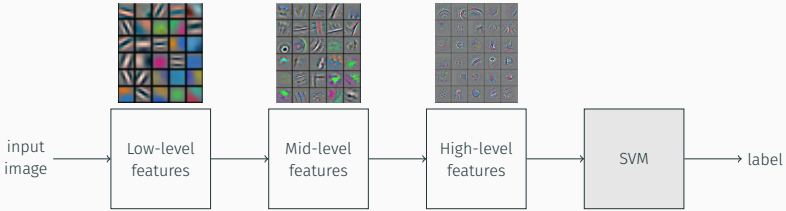
Choice of the degree (cont.)

Notice that choosing the degree using the VC-dimension does not necessarily imply the minimization of the test errors, as shown by the table.

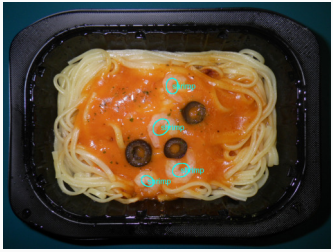
	Chosen classifier			Number of test errors						
digit	d	n	VC	1	2	3	4	5	6	7
0	3	$\approx 10^6$	530	36	14	11	11	11	12	17
1	7	$\approx 10^{16}$	101	17	15	14	11	10	10	10
2	3	$\approx 10^6$	842	53	32	28	26	28	27	32
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

SVMs for transfer learning

- SVMs are suitable for *transfer learning*, i.e. exploiting specialized features extractors, for instance Convolutional Neural Networks.
- Idea: take a pre-trained network, replace the last layer with an SVM and train the SVM only.



Example: transfer learning with VGG



- Transfer learning based on VGG net (Simonyan and Zisserman, 2014);
- took the output of the last hidden layer (4096 elements);
- trained a **linear** SVM.
- (detection of candidate regions based on scale-space approach).

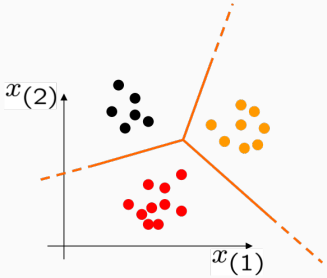
Multi-class classification

Multiclass SVM

Support Vector Machine are inherently a binary classification tool.

However, various approaches exist to combine many binary SVMs to obtain a multiclass classifier:

- One-against-rest SVM
- Pairwise SVM
- All-at-once SVM
- ...



One-against-rest

The simplest approach is the following.

- For n classes, train n classifiers, each separating a class from the remaining.
- n decision function are thus obtained, whose primal form is

$$h_i(x) = \Theta(w_i \cdot \Phi(x) + b_i),$$

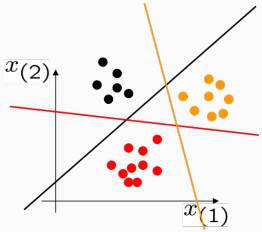
where h_i is the decision function corresponding to the SVM that separates class i from the remaining.

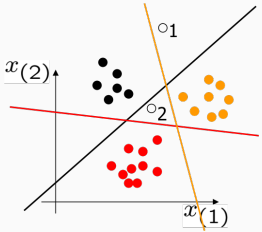
There may exist some unclassifiable regions, for instance:

- point 1 in the figure belongs both to the orange and the black class;
- point 2 does not belong to any class.

To overcome the problem we can take into account the distance from the separating hyperplane:

$$\text{winning class} = \arg \max_{i=1,\ldots,n} w_i \cdot \Phi(x) + b_i$$





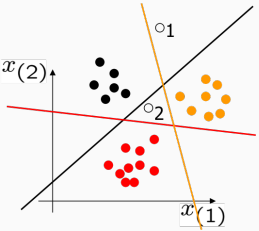
One-against-rest (cont.)

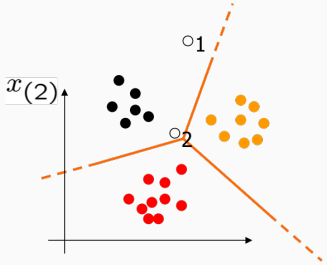
$$\text{winning class} = \arg \max_{i=1,\ldots,n} w_i \cdot \Phi(x) + b_i$$

By taking into account the distance from the hyperplane:

- the point 1 is closer to the boundary of the orange region than to the boundary of the black one. Thus it is classified as black.
- the point 2 is closer to the black region boundary than to the other two region boundaries thus it is classified as black.

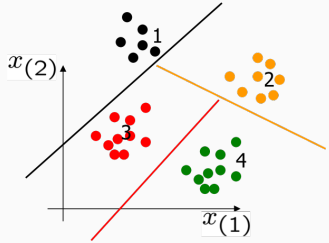
The resulting decision regions are shown in the bottom figure.

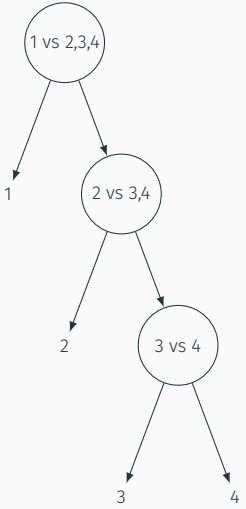




One-against-rest with decision tree

- Train $n-1$ binary classifiers.
- In the simplest case the i-th classifier separates class i from classes $i+1, \ldots, n$.
- The classification is performed by evaluating the binary classifiers in sequence.
- As soon as the i-th classifier produces a positive output, the pattern is classified into class i.
- Thus, at most $n-1$ classifiers need to be evaluated.

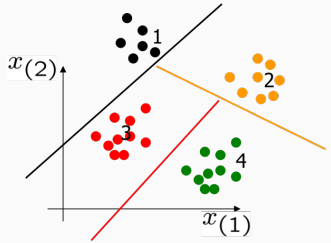


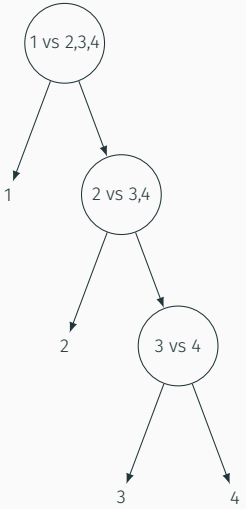


One-against-rest with decision tree (cont.)

- The lower index classes have larger decision regions;
- the obtained classifier depends on the order of the classes;
- it is important to carefully choose the order of the classes to improve the generalization capacity

- Guiding principle: **classes that are easily separable from the rest should be in the upper part of the tree**.
- Indeed, the more data are misclassified at the upper node of the decision tree, the worse the classification performance becomes.
- See (Abe, 2010) for details and strategies for building decision trees.





Pairwise SVM

- For n classes, train $n(n-1)/2$ classifiers, each separating a class from one of the remaining classes.
- $n(n-1)/2$ decision function are thus obtained, whose primal form is

$$h_{ij}(x) = \Theta(w_{ij} \cdot \Phi(x) + b_{ij}),$$

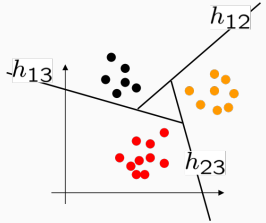
  where h_{ij} is the decision function corresponding to the SVM that separates class i from class j.

- Define the disjoint regions

$$R_i = \{x | h_{ij}(x) > 0, \ \ \forall j \neq i\};$$

  (the region R_i is the region where all the classifiers involving class i are in agreement).
- If $x \in R_i$, it is classified into class i; otherwise, the class is decided by voting:

$$\text{winning class} = \arg \max_{i=1,\ldots,n} \sum_{j \neq i,\ j=1}^{n} h_{ij}(x).$$



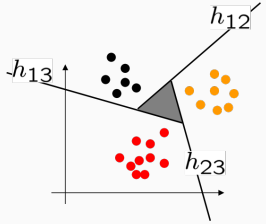
Pairwise SVM (cont.)

- Unfortunately, there are still unclassifiable regions.
- For instance, in the example in the figure, the points belonging to the gray area all have a total of zero votes (one positive and one negative):

$$h_1(x) = h_2(x) = h_3(x) = 0$$

- However, the region of indecision is typically much smaller than that which would be obtained with the conventional One-against-rest approach.
- Possible solution: adopt a criterion based on the (signed) distance from the decision area.

$$\text{winning class} k = \arg \max_{i=1,\ldots,n} \left(\min_{j=1,\ldots,n,\ j \neq i} w_{ij} \cdot \Phi(x) + b_{ij} \right).$$



Pairwise SVM (cont.)

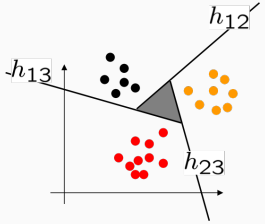
- Unfortunately, there are still unclassifiable regions.
- For instance, in the example in the figure, the points belonging to the gray area all have a total of zero votes (one positive and one negative):

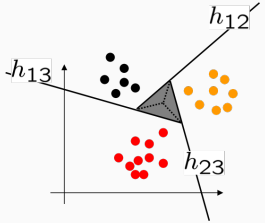
$$h_1(x) = h_2(x) = h_3(x) = 0$$

- However, the region of indecision is typically much smaller than that which would be obtained with the conventional One-against-rest approach.
- Possible solution: adopt a criterion based on the (signed) distance from the decision area.

$$\text{winning class} k = \arg \max_{i=1,\ldots,n} \left(\min_{j=1,\ldots,n,\ j \neq i} w_{ij} \cdot \Phi(x) + b_{ij} \right).$$

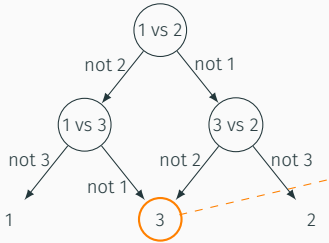
- The indecision is thus resolved as in figure.

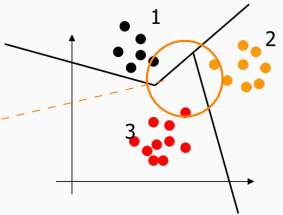


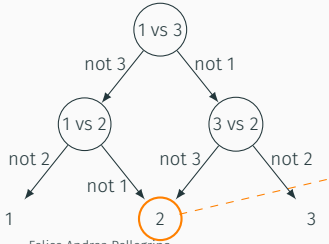


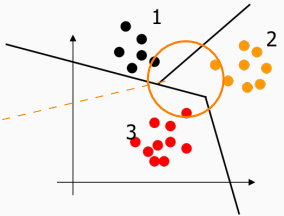
Pairwise SVM with decision tree

- DAG: Direct Acyclic Graph (Platt et al., 2000).
- It is sufficient to evaluate $n - 1$ classifiers.
- The result, however, depends on the order of evaluation.









Pairwise SVM with decision tree (cont.)

- It is reasonable to demand that regions of indecision be assigned to classes that are difficult to separate from the others.
- In fact, assigning them to classes that are easy to separate is most likely an error.
- This result is obtained by placing in the highest nodes of the tree the pairs of classes that are more easily separable (those pairs for which the generalization capability[2] is high).

Algorithm DAG structure for n-class problem

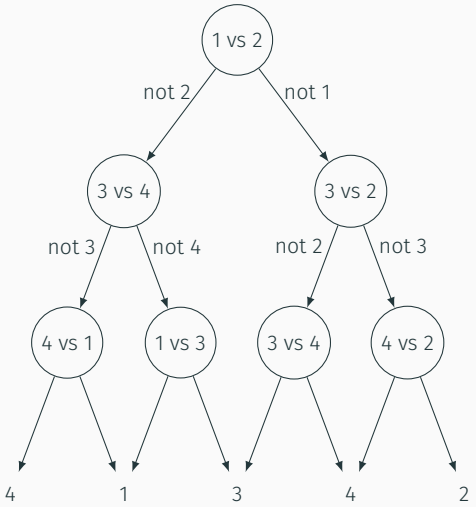
1: Generate the initial list: $\{1, \ldots, n\}$.
2: **while** there is at least one list remaining of more that two elements **do**
3: Choose a list and select the class pair (i, j) from the list with the highest generalization capability.
4: **if** the chosen list of step 3 has more than two elements **then**
5: Generate two lists deleting i and j respectively from the list.
6: **end if**
7: **end while**

[2] estimated for example through the number of support vectors.

Pairwise SVM with decision tree (cont.)

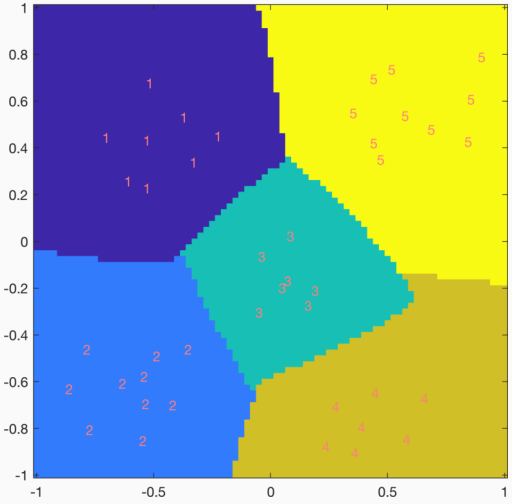
Example (four classes):

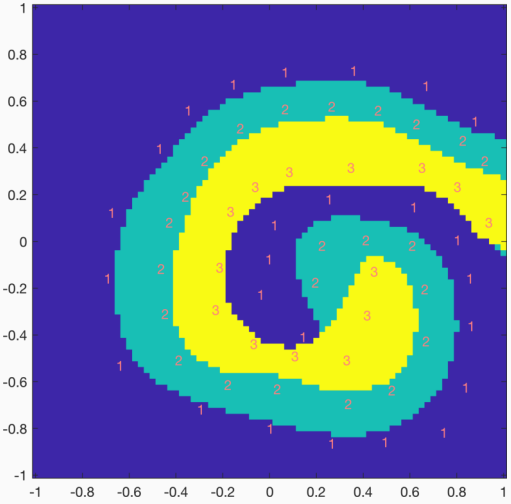
- initial list: $\{1, 2, 3, 4\}$;
- let the pair of classes with the greatest generalization capacity be $\{1, 2\}$;
- we get the two lists: $\{1, 3, 4\}$ and $\{2, 3, 4\}$;
- let the pair of classes among $\{1, 3, 4\}$ with the greatest generalization capacity be $\{3, 4\}$;
- let the pair of classes among $\{2, 3, 4\}$ with the greatest generalization capacity be $\{2, 3\}$;
- we obtain the lists $\{1, 4\}$, $\{1, 3\}$, $\{3, 4\}$ and $\{2, 4\}$ which are the lowest level nodes.



Pairwise SVM with decision tree (cont.) (cont.)

Examples





All-at-once SVM

- In the all-at-once approcah (Weston and Watkins, 1998) all the decision functions are **simultaneously determined**.
- For each of the classes a hyperplane in the feature space is generated, of equation:

$$w_i \cdot \Phi(x) + b_i = 0.$$

- For x belonging to class i, the following inequalities are imposed;

$$w_i \cdot \Phi(x) + b_i > w_j \cdot \Phi(x) + b_j, \quad j \neq i, j = 1, \ldots, n.$$

- The inequalities basically say that **the signed distance from the hyperplanes should be maximal for hyperplane of the true class**.

All-at-once SVM (cont.)

- By setting $W = [w_1 \ \dots \ w_n]$ and $b = [b_1 \ \dots \ b_n]^\top$, the problem can be formulated (in the primal form) as follows:

$$\begin{aligned} &\min_{W,b,\xi} \frac{1}{2} \sum_{1}^{n} (w_i \cdot w_i) + C \sum_{i=1}^{l} \sum_{j \in \mathcal{I} \setminus y_i} \xi_{ij} \\ &\text{s.t.} \\ &(w_{y_i} - w_j) \cdot \Phi(x_i) + b_{y_i} - b_j \geq 1 - \xi_{ij}, \ \forall j \neq y_i, \ i = 1, \dots, l \\ &\xi_{ij} \geq 0, \ \forall i, j \end{aligned} \tag{1}$$

where $\mathcal{I} = \{1, \dots, n\}$ and ξ_{ij} are slack variables associated to constraint violations (thus a soft margin is employed).

All-at-once SVM (cont.)

- The corresponding dual formulation is

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{l} \sum_{j \in \mathcal{I} \setminus y_i} \alpha_{ij} - \frac{1}{2} \sum_{i,h=1}^{l} \sum_{j=1}^{n} z_{ij} z_{hj} k(x_i, x_j)$$

s.t.

$$\sum_{i=1}^{l} z_{ij} = 0, \ \ \forall j \neq y_i$$

$$0 \leq \alpha_{ij} \leq C, \ \forall i, j \neq y_i$$

,

where

$$z_{ij} = \begin{cases} \sum_{h \in \mathcal{I} \setminus y_i} \alpha_{ih} & \text{if } \ j = y_i \\ -\alpha_{ij} & \text{otherwise} \end{cases}$$

Again, it is a convex quadratic programming problem.

All-at-once SVM (cont.)

- The decision functions are:

$$h_i(x) = \sum_{j=1}^{l} z_{ij} k(x_j, x) + b_i.$$

- The classification rule is:

$$\text{winning class} = \arg \max_{i=1,\ldots,n} h_i(x).$$

- It can be shown that the solution is sparse, since many of the z_{ij} are zero.
- However, the optimization problem is large, since there are $l \times n$ slack variables.

All-at-once SVM (cont.)

Observation:

- define the *score* for the class j

$$s_j \doteq w_j \Phi(x) + b_j;$$

- define a *loss* L_i associated to the pair x_i, y_i as

$$L_i \doteq \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1);$$

- then, the primal problem (1) is equivalent to

$$\min_{W,b} \ \lambda \| W \|_F^2 + \frac{1}{l} \sum_{i=1}^{l} L_i$$

  for a proper choice of λ.

Thus, it is basically a loss minimization with L^2 parameter regularization[3].

[3] also known as *ridge regression* or *Tikhnov regularization*.

Incorporating a priori knowledge

Incorporating a priori knowldege

Some methods to introduce a priori knowledge (for example the **invariance** with respect to certain transformations of input vectors) are

- virtual support vectors;
- kernel jittering;
- (ad hoc kernel design).

For a detailed discussion see (Schölkopf and Smola, 2001).

Virtual support vectors

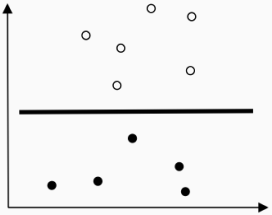
- The most basic method of “forcing” the invariance to a transformation, for a generic learning machine, is *data augmentation*.
- By data augmentation we create and enriched training set, adding transformed copies of the vectors of the original training set.
- Problem: in this way the training set may become huge and therefore computational problems could arise.

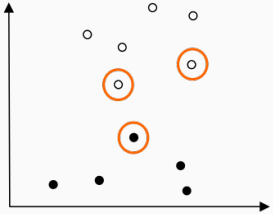
In the SVM case, it is possible to enrich the training set by **transforming the support vectors only**, which are the only significant ones for discrimination purposes.

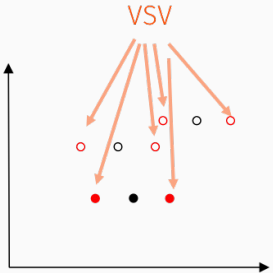
The method is called the *Virtual Support Vectors (VSV) method*.

Virtual support vectors (cont.)

Example: enforcing invariance to horizontal translation.



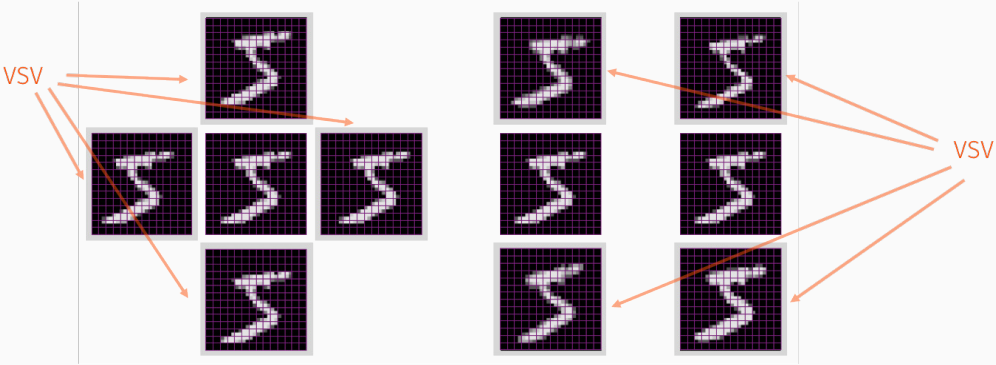




Virtual support vectors (cont.)

Optical character recognition (Schölkopf and Smola, 2001).

Incorporating invariance to translation (left), rotation (center) and thickness (right).



Virtual support vectors (cont.)

Summarizing:

- Train an SVM and identify the support vectors (SV);
- generate artificial examples by transforming the support vectors according to the transformations with respect to which we want to impose invariance. Artificial examples are called virtual support vectors (VSV);
- train a new SVM with a TS consisting of SV and VSV.

Notice that the procedure can be iterated. However, attention must be payed to avoid introducing unwanted invariances (for example a “6” could be reversed, becoming a “9” but obviously that’s not desirable).

Kernel jittering

- The transformations of the training set vectors do not take place before training but **at the time of kernel computation**.
- Instead of the Hessian matrix

$$K_{ij} = k(x_i, x_j),$$

we use a modified version

$$K_{ij}^J = k^J(x_i, x_j)$$

defined as follows.

- Among all the transformed versions of x_i (including x_i itself) the "closest" to x_j is chosen (say, x_q) and the new kernel becomes

$$K_{ij}^J = k^J(x_i, x_j) = k(x_q, x_j).$$

Kernel jittering (cont.)

Indeed:

- the kernel is a **similarity measure** between patterns;
- the goal is to discriminate between different patterns.
- As a consequence, when enriching the training set, **it is sufficient to take into account the most similar, but different, patterns**.

Kernel jittering (cont.)

- Clearly, the distance must be computed in the feature space.
- Recalling the definition of Euclidean distance in terms of dot product:

$$\|x - y\|^2 = x \cdot x - 2x \cdot y + y \cdot y$$

the distance in the feature space may be written as

$$\begin{aligned} ||\Phi(x_i) - \Phi(x_j)||^2 &= \Phi(x_i) \cdot \Phi(x_i) - 2\Phi(x_i) \cdot \Phi(x_j) + \Phi(x_j) \cdot \Phi(x_j) \\ &= k(x_i, x_i) - 2k(x_i, x_j) + k(x_j, x_j) \end{aligned} \tag{2}$$

thus it **can be computed implicitly by means of the kernel**.

- For a given x_j, the closest pattern is the one minimizing (2).

Kernel jittering (cont.)

Advantage of the jitter method (JSV) e compared to the virtual support vector method (VSV):

- **the VSV method scales quadratically** with respect to the number N_J of transformations generated because the SVM algorithm scales quadratically with respect to the size of the TS and the VSV method expands the TS by a factor N_J;
- the JSV method instead keeps the size of the TS constant, although there is an additional computational load (it requires for each element a minimization among N_J generalized scalar products).
- Thus, **the JSV method scales linearly** with respect to N_J.

References

References

Abe, S. (2010). *Support Vector Machines for Pattern Classification (Advances in Pattern Recognition)*. Springer-Verlag, London.

Aizerman, M., Braverman, E. M., and Rozonoer, L. (1964). Theoretical foundations of potential function method in pattern recognition. *Automation and Remote Control*, 25(6):917–936.

Boser, B., Guyon, I., and Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, volume 54, pages 144–152. ACM.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27.

Luenberger, D. G. and Ye, Y. (2016). *Linear and Nonlinear Programming*. Springer International Publishing.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Novikoff, A. (1962). On convergence proofs for perceptrons. In *Proceedings of the Symposium of the Mathematical Theory of Automata*, volume 12, pages 615–622.

Platt, J. C., Cristianini, N., and Shawe-Taylor, J. (2000). Large Margin DAGs for Multiclass Classification. In *Advances in neural information processing systems*, pages 547–553.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

References (cont.)

Schölkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

Vapnik, V. (2000). *The nature of statistical learning theory, Second Edition*. Springer-Verlag New York.

Weston, J. and Watkins, C. (1998). Multi-class Support Vector Machines. Technical report, Royal Holloway University of London, London.

554SM –Fall 2018

Lecture 7
Support vector machines for classification

END