Machine Learning Methods
for Agricultural & Food Data Management

— Part 3 —

– Statistical Learning Theory and Support Vector Machines –

Paul HONEINE

LITIS Lab
paul.honeine@univ-rouen.fr

— Spring 2021 —
version 2.01

Outline

Prologue

Machine Learning Methods: Outline

Outline:

Part 1: Introduction to Machine Learning

Part 2: (“Primal”) Machine Learning Algorithms

Part 3: **Statistical Learning Theory and Support Vector Machines** (this file)

Part 4: Multiclass and Regression

Outline

Statistical Machine Learning

Introductory example: detection/classification problem

A detection/classification problem can be written as:

$$\begin{cases} \omega_0 : \boldsymbol{x} = \boldsymbol{b} & \text{Hypothesis "only noise"} \\ \omega_1 : \boldsymbol{x} = \boldsymbol{b} + \boldsymbol{s} & \text{Hypothesis "signal and noise"} \end{cases}$$

One needs to determine a detector/classifier $\psi(\cdot)$, with the minimal probability error for instance, namely

$$P_e(\psi) = p(\psi(\boldsymbol{x}) \neq y),$$

where $\boldsymbol{x}$ is the observation and y the associated hypothesis.

The strategy to adopt to solve this problem depends on the nature of the information and knowledge available on $(\boldsymbol{x}, y)$.

Detection/classification problem

Modes of resolution

Free-structure detection/classification

By restricting ourselves to simple assumptions, the application of a decision rule such as that of Bayes leads to

$$\psi^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad p(\boldsymbol{x}|\omega_1)/p(\boldsymbol{x}|\omega_0) \geq \lambda_0 \\ 0 & \text{otherwise,} \end{cases}$$

subject to knowing at least $p(\boldsymbol{x}|\omega_0)$ and $p(\boldsymbol{x}|\omega_1)$. The threshold λ_0 is the only parameter that depends on the chosen rule.
Thus, the detector/classifier is not subject to any structural constraint, but results from the choice of a criterion.

Detection/classification with imposed structure

Ignorance of the statistical properties of the sample requires the implementation of an alternative strategy, which can be

1. define a class of detectors/classifiers $\mathcal{H} = \{\psi(\boldsymbol{x}, \theta) \,:\, \theta \in \Theta\}$
2. select the "best" element of $\mathcal{H}$

Simple in appearance, this approach assumes that the following questions are satisfactorily answered:

1. How to choose the detector/classifier class $\mathcal{H}$?
2. What are the relevant risk functionals for the problem being addressed ?
3. Which optimization procedure to adopt ?

Learning Problem

Problem of learning a decision rule

The knowledge of a probabilistic model is replaced by that of a set of data (*i.e.*, training dataset) $\mathcal{A}_n$:

$$\mathcal{A}_n = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}.$$

We seek a decision rule that consists in finding a partition of the space of observations $\mathcal{X}$ that is optimal in the sense of the chosen performance criterion.

There are two main approaches that can be distinguished:

1. Direct use of the learning dataset for decision making (*e.g.* k-nearest neighbors rule)
2. Choice of the structure of the decision rule, then optimization of its characteristic parameters according to the chosen criterion

Learning Problem

k-nearest neighbors rule

The k-nearest neighbors rule

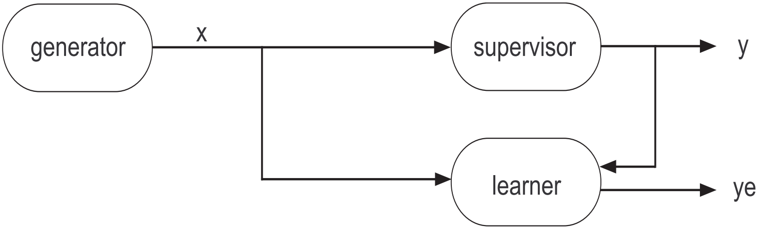
...

...

...

Learning Problem

Learning a decision function



1. Generator: $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$, as random vectors i.i.d.
2. Supervisor: $y \in \mathcal{Y} \subset \mathbb{R}$, as random variables
3. Learner: represented by $\psi(\boldsymbol{x}; \theta) \in \mathcal{H}$

Learning Problem

Exemples of learners

– **Polynomials of degree p**

$$\psi(\boldsymbol{x};\boldsymbol{a}) = \sum_{\substack{i_1,\ldots,i_d \in \mathbf{N} \\ i_1+\ldots+i_l \leq p}} a_{i_1,\ldots,i_d}\, x[1]^{i_1} \ldots x[d]^{i_d}$$

..., and other decompositions on a basis, such as Fourier series, Haar, ...

– **Splines**

$$\psi(\boldsymbol{x};c) \in \mathcal{L}^2(\mathbb{R}^d) \text{ such that } \psi' \in \mathcal{L}^2(\mathbb{R}^d), \|\psi'\|^2 \leq c$$

– **Nadaraya-Watson**

$$\psi(\boldsymbol{x};\sigma) = \frac{\sum_{i=1}^n y_i\, K_\sigma(\boldsymbol{x},\boldsymbol{x}_i)}{\sum_{i=1}^n K_\sigma(\boldsymbol{x},\boldsymbol{x}_i)}$$

– **MLP, RBF, ...**

$$\psi(\boldsymbol{x};\boldsymbol{a},\boldsymbol{\theta}) = \sum_k a_k\, g_k(\boldsymbol{x};\boldsymbol{\theta}_k)$$

Learning Problem

Risk minimization

Objective

To find in $\mathcal{H} = \{\psi(\boldsymbol{x}, \theta) \,:\, \theta \in \Theta\}$ a function realizing the best approximation of y in the sense of a risk functional of the form

$$J(\psi) = \int Q(\psi(\boldsymbol{x}, \theta), y)\, p(\boldsymbol{x}, y)\, d\boldsymbol{x}\, dy,$$

where Q represents a cost associated to each pair $(\boldsymbol{x}, y)$.

Example of cost function: probability error

When it comes to developing a minimum error probability decision structure, the risk is expressed as

$$P_e(\psi) = \int \mathbb{1}_{(\boldsymbol{x},\theta)\neq y}\, p(\boldsymbol{x}, y)\, d\boldsymbol{x}\, dy,$$

where $\mathbb{1}$ denotes the indicator function.

Problem of learning

Other examples of cost functions

- **Quadratic cost**

$$Q(\boldsymbol{x}, y) = (y - \psi(\boldsymbol{x}; \theta))^2 \quad \rightarrow \quad \psi^*(\boldsymbol{x}; \theta) = \mathrm{E}(y \,|\, \boldsymbol{x})$$

- **Absolute cost**

$$Q(\boldsymbol{x}, y) = |y - \psi(\boldsymbol{x}; \theta)|$$

- **Cross Entropy**

$$Q(\boldsymbol{x}, y) = -y \log(\psi(\boldsymbol{x}; \theta)) - (1 - y) \log(1 - \psi(\boldsymbol{x}; \theta)) \quad \rightarrow \quad \psi^*(\boldsymbol{x}; \theta) = \mathrm{P}(y = 1 \,|\, \boldsymbol{x})$$

Learning Problem

Minimization of the empirical risk

It's about minimizing the risk functional

$$J(\psi) = \int Q(\psi(\boldsymbol{x};\theta), y)\, p(\boldsymbol{x}, y)\, \psi\boldsymbol{x}\, dy,$$

the probability density $p(\boldsymbol{x}, y)$ being unknown.

Minimization of the empirical risk (MER)

The minimization of $J(\psi)$ translates into that of the empirical risk

$$J_{emp}(\psi) = \frac{1}{n}\sum_{k=1}^{n} Q(\psi(\boldsymbol{x}_k;\theta), y_k),$$

which can be evaluated using the training dataset $\mathcal{A}_n$.

Empirical probability error

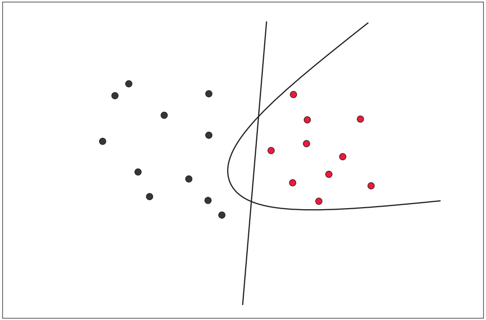
The empirical risk associated to the probability of error is the number of assignment errors committed by $\psi(\boldsymbol{x};\theta)$ on $\mathcal{A}_n$, namely

$$P_{emp}(\psi) = \frac{1}{n}\sum_{k=1}^{n} \mathbb{1}_{\psi(\boldsymbol{x}_k;\theta)\neq y_k}.$$

Learning Problem

Example of implementation issue

Problem. Two Gaussian families ω_0 and ω_1 in $\mathbb{R}^2$, of distinct means and covariance matrices, each made up by 10 samples.



Which frontier to choose ?

What happens if $\hat{P}_e(\text{linear}) = 5\%$ while $\hat{P}_e(\text{quadratic}) = 9\%$?

Learning Problem

Approximation error and estimation error

Let $\psi^* = \arg\min J(\psi)$ be the rule of minimal risk, and $\psi_n^* = \arg\min_{\psi \in \mathcal{H}} J_{emp}(\psi)$ the one that minimizes the empirical risk on $\mathcal{H}$ using the training dataset $\mathcal{A}_n$.

Definition (Estimation error)

It is the difference in performance between the best rule in $\mathcal{H}$ and the one obtained from learning:

$$J_{estim} = J_e(\psi_n^*) - \inf_{\psi \in \mathcal{H}} J_e(\psi)$$

$\triangleright$ *relevance of the empirical criterion and performance of the algorithm*

Definition (Approximation error)

It is given by the difference in performance between the optimal rule ψ^* and the best one within $\mathcal{H}$:

$$J_{approx} = \inf_{\psi \in \mathcal{H}} J_e(\psi) - J_e(\psi^*)$$

$\triangleright$ *choice of the class $\mathcal{H}$*

Learning Problem

Modeling error

Learning

The goal of learning is to minimize the modeling error, defined by:

$$J_{mod}(\psi_n^*) = J_e(\psi_n^*) - J_e(\psi^*).$$

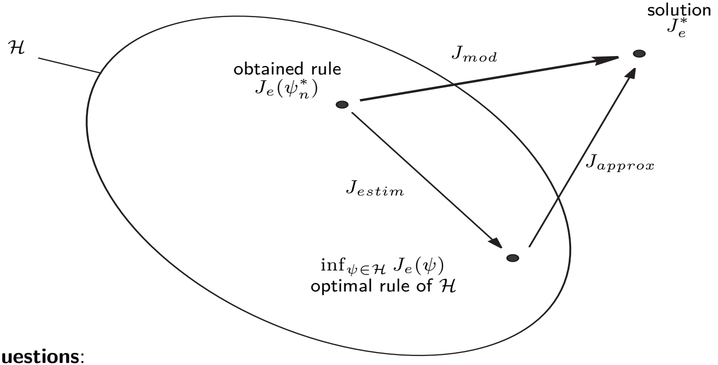
There are two contributions of different natures in this error:

$$J_{mod}(\psi_n^*) = \underbrace{\left(J_e(\psi_n^*) - \inf_{\psi \in \mathcal{H}} J_e(\psi)\right)}_{J_{estim}} + \underbrace{\left(\inf_{\psi \in \mathcal{H}} J_e(\psi) - J_e(\psi^*)\right)}_{J_{approx}}.$$

The minimization of J_{mod} is based on the search for a tradeoff between these two antagonistic terms: the increase in the number of tests in $\mathcal{H}$ leads to an increase of J_{estim} while J_{approx} decreases, and vice versa.

Learning Problem

Approximation error, estimation error, and modeling error



Questions:

1. *Is the objective feasible ?*

   $\rightarrow$ *Consistency of the decision rule*

   $\rightarrow$ *Consistency of the induction principle*

   $\rightarrow$ *Convergence rate*

2.: *If yes, how in practice ?*

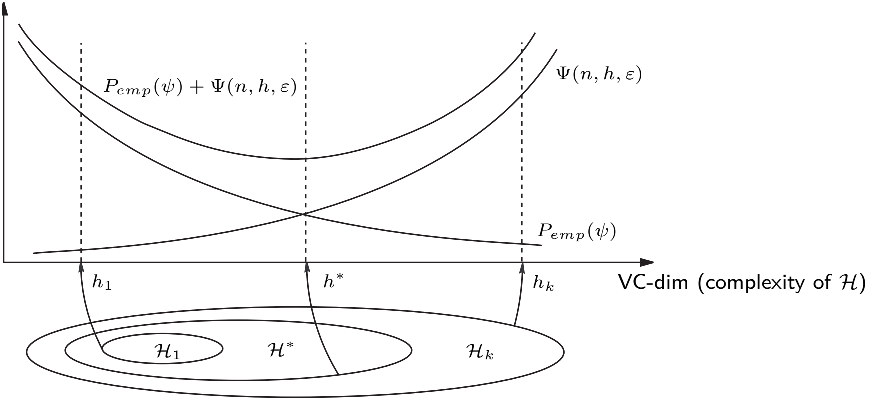
Learning Problem

Consistency and convergence rate from P_{emp} to P_e

Precursor work by Vapnik and Chervonenkis (1971) provided quantitative instructions on the convergence rate of P_{emp} to P_e.

Inequality of Vapnik-Chervonenkis:

With a probability of at least $1-\varepsilon$, we have: $P_e(\psi) \leq P_{emp}(\psi) + \sqrt{\frac{h\ \ln(\frac{2en}{h}) - \ln\frac{\varepsilon}{4}}{n}}$.



Learning Problem

Principle of the structural risk minimization

The *structural risk minimization* principe advocated by Vapnik implies the construction, within the class $\mathcal{H}$, a sequence of nested subsets $\mathcal{H}_k$

$$\mathcal{H}_1 \subset \ldots \subset \mathcal{H}_k \subset \ldots \subset \mathcal{H}.$$

With this structure established, the learning phase is conducted in two stages:

1. Search for the detector/classifier with minimum empirical error in each subset $\mathcal{H}_k$:
$$\psi^*_{n,k} = \arg \min_{\psi \in \mathcal{H}_k} P_{emp}(\psi).$$
2. Select the detector/classifier with the best guaranteed error $P_{emp}(\psi^*_{n,k}) + \Psi(n, h_k, \varepsilon)$:

$$\psi^*_n = \arg\min_{k \geq 1} \{P_{emp}(\psi^*_{n,k}) + \Psi(n, h_k, \varepsilon)\}.$$

Outline

Statistical Machine Learning

Ill-posed Problems and Regularization

Ill-posed problems and regularization

Ill-posed Problem

Introduction

Learning Problem:

We seek a function from a space $\mathcal{H}$ of candidate functions defined from $\mathcal{X}$ to $\mathcal{Y}$, such that, for any x, predicts the corresponding label y, namely

$$y = \psi(\boldsymbol{x})$$

We have a training dataset $\mathcal{A}_n = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$

▷ *Goal: empirical risk minimization AND generalization !*

Ill-posed Problem

Definition of ill-posedness

Definition (Well-posed problem / ill-posed problem (Hadamard))

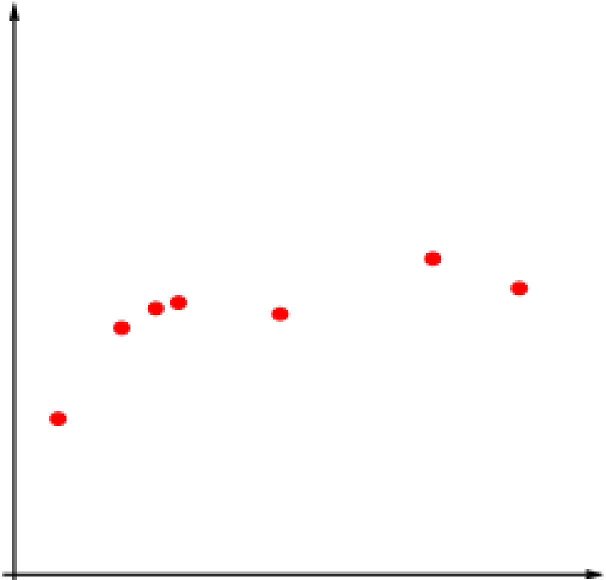
A problem is said *well-posed* if

- the solution exists
- the solution is unique
- the solution is a continuous function of the data (a small perturbation of the data leads to a small perturbation of the solution)

A problem is said *ill-posed* if it is not well-posed

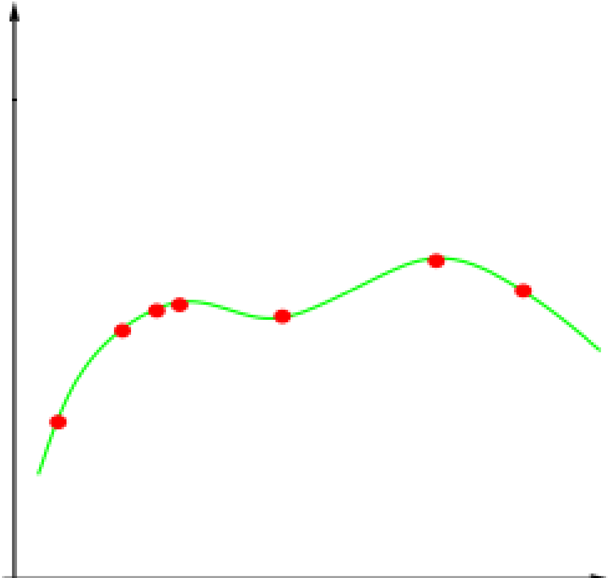
Ill-posed Problem

Unique solution !



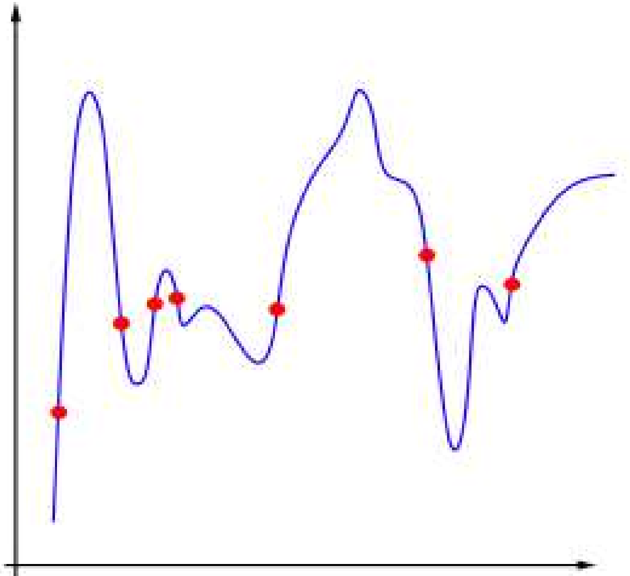
Ill-posed Problem

Unique solution !



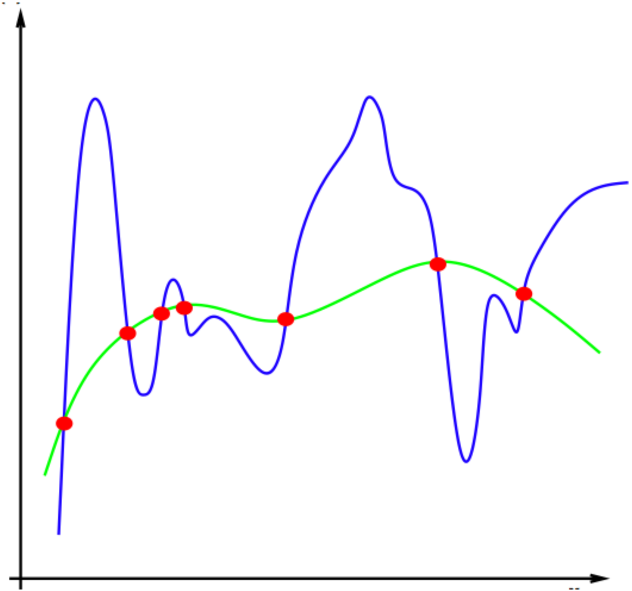
Ill-posed Problem

Unique solution !



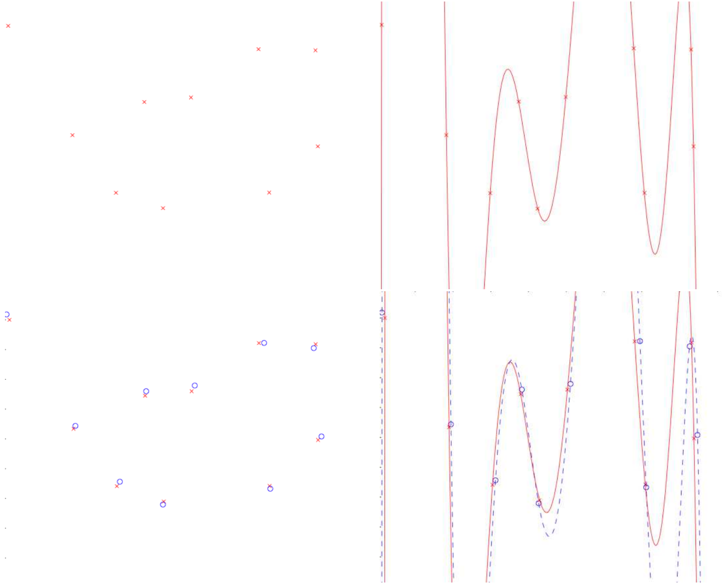
Ill-posed Problem

Unique solution !



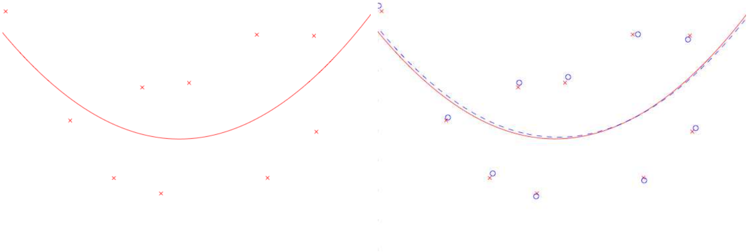
Ill-posed Problem

Continuity of the solution !



Ill-posed Problem

Continuity of the solution !



Ill-posed Problem

Minimization of the empirical risk

The minimization of the empirical risk

$$J_{emp}(\psi) = \frac{1}{n}\sum_{k=1}^{n} Q(\psi(\boldsymbol{x}_k), y_k),$$

is an ill-posed problem.

Solution: **Regularization**

Ill-posed Problem

Ivanov regularization

Régularisation d'Ivanov

Determine the function ψ that minimizes

$$\frac{1}{n}\sum_{k=1}^{n} Q(\psi(\boldsymbol{x}_k), y_k),$$

subject to

$$\|\psi\|^2 \leq A$$

Ill-posed Problem

Tikhonov regularization

Penalized empirical risk:

$$\mathsf{RisqEmp}(\psi) + \eta\ \mathsf{Pen}(\psi),$$

where η is a positive parameter that controls the tradeoff of the two terms.

▷ *The penalty terme allows to incorporate a smoothing effect*

Tikhonov regularization

Determine the function ψ of a space $\mathcal{H}$ of candidate functions, minimizing

$$\frac{1}{n}\sum_{k=1}^{n} Q(\psi(\boldsymbol{x}_k), y_k) + \eta\|\psi\|^2_{\mathcal{H}},$$

for a parameter $\eta > 0$, and where $\|\psi\|_{\mathcal{H}}$ is a functional norm in the space $\mathcal{H}$.

This problem is well-posed.

Outline

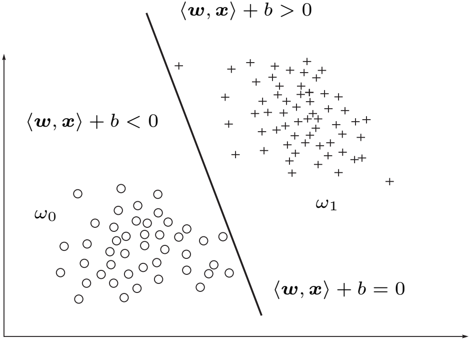
Algorithms: Support Vector Machines

The Percetron Algorithm

The Perceptron algorithm aims to produce a minimum learning error solution by minimizing the following empirical risk:

$$(\boldsymbol{w}^*, b^*) = \arg\min_{(\boldsymbol{w},b)} \sum_{i=1}^{n} |y_i - d(\boldsymbol{x}_i; \boldsymbol{w}, b)|.$$

- *Why would the obtained solution have the best performance?*
- *Is minimizing the empirical error a good idea?*
- *Is there an alternative ?*



Linear classifier

Definition

We consider a two-class classification problem of n samples in $\mathbb{R}^d$, given a training dataset

$$\mathcal{A}_n = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}.$$

Let $y_i = (-1)$ if $\boldsymbol{x}_i \in \omega_0$, and $y_i = (+1)$ if $\boldsymbol{x}_i \in \omega_1$.

A **linear classifier** is defined by

$$d(\boldsymbol{x}; \boldsymbol{w}, b) = \text{sign}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b).$$

Linear Separability

Definition

We consider a two-class classification problem of n samples in $\mathbb{R}^d$, given a training dataset

$$\mathcal{A}_n = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}.$$

Let $y_i = (-1)$ if $\boldsymbol{x}_i \in \omega_0$, and $y_i = (+1)$ if $\boldsymbol{x}_i \in \omega_1$.

A hyperplane is defined by the following equation, upto a multiplicative constant:

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0 \quad \Longleftrightarrow \quad \langle \gamma\, \boldsymbol{w}, \boldsymbol{x} \rangle + \gamma\, b = 0, \quad \gamma \in \mathbb{R}^*$$

The classes ω_0 and ω_1 are called **linearly separable** if there exist $\boldsymbol{w}$ and b, such that

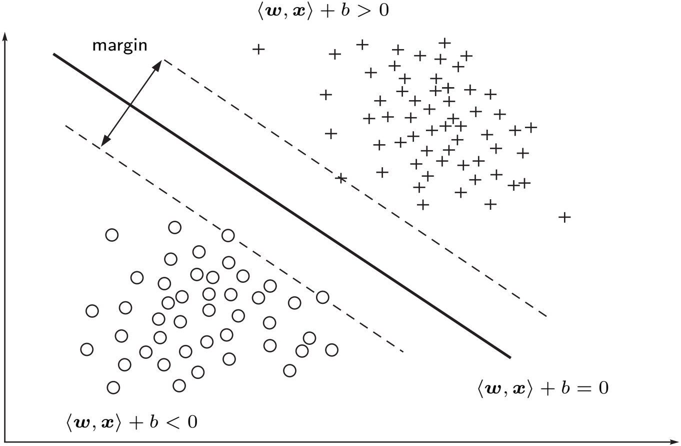
$$\begin{aligned} \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b &\geq +1 \qquad \forall \boldsymbol{x}_i \in \omega_1 \\ \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b &\leq -1 \qquad \forall \boldsymbol{x}_i \in \omega_0 \end{aligned}$$

In the following, we summarize this criterion of separability as

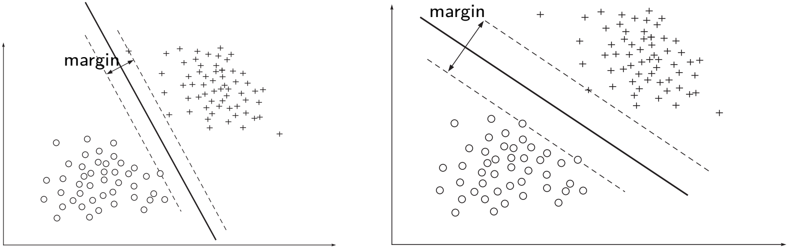
$$y_i\left(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b\right) - 1 \geq 0 \qquad \forall (\boldsymbol{x}_i, y_i) \in \mathcal{A}_n$$

A new induction principle

Among the separators having a minimum empirical error, it is advisable to choose the one of maximum margin (Vapnik 1965, 1992).



A new induction principle
Illustration



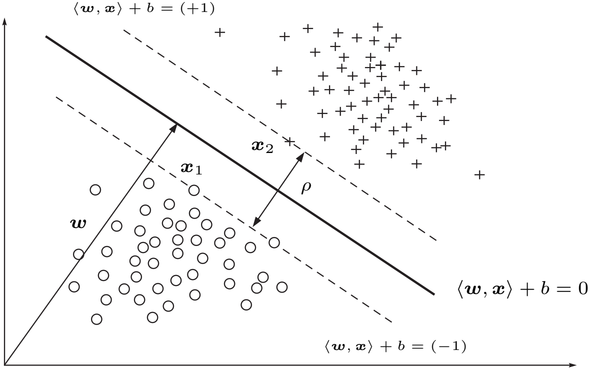
Small margin: *expected low performance in generalization*

Wide marge: *probably good performance in generalization*

This will be justified more rigorously now.

A new induction principle

Margin calculus



We have $\langle \boldsymbol{w}, \boldsymbol{x}_2 \rangle + b = (+1)$ and $\langle \boldsymbol{w}, \boldsymbol{x}_1 \rangle + b = (-1)$. Therefore, we get

$$\rho = \left\langle \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}, \boldsymbol{x}_2 - \boldsymbol{x}_1 \right\rangle = \frac{2}{\|\boldsymbol{w}\|}$$

A new induction principle

Margin maximization

The maximization of the margin ρ, which is the fundamental principle of SVM, is justified by the following result from the statistical theory of learning.

Theorem

Consider the hyperplanes of the form $\langle \boldsymbol{w}, \boldsymbol{x} \rangle = 0$, where $\boldsymbol{w}$ is normalized in a way that the hyperplanes take the canonical form with respect to $\mathcal{A}_n$, namely

$$\min_{\boldsymbol{x} \in \mathcal{A}_n} |\langle \boldsymbol{w}, \boldsymbol{x} \rangle| = 1.$$

The set of decision functions $\psi(\boldsymbol{x}; \boldsymbol{w}) =$ sgn$\langle \boldsymbol{w}, \boldsymbol{x} \rangle$, defined from $\mathcal{A}_n$ and satisfying the constraint $\|\boldsymbol{w}\| \leq \Lambda$, has an upper-bounded VC-dimension h with

$$h \leq R^2 \Lambda^2,$$

where R is the radius of the smallest sphere centered on the origin containing $\mathcal{A}_n$.

As a result, the more $\rho = 2/\|\boldsymbol{w}\|$ is large, the more h is small.

Support vector machines (hard margin)

Formulation of the optimization problem

Maximizing the margin, defined by $\rho = \frac{2}{\|\boldsymbol{w}\|}$, is equivalent to minimizing $\|\boldsymbol{w}\|^2$. The MRS principle is implemented by solving the following optimization problem:

Minimize $\frac{1}{2}\|\boldsymbol{w}\|^2$

subject to $y_i\left(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b\right) \geq 1, \qquad 1 \leq i \leq n.$

Note: This formulation is valid only for linearly separable classes.

Support vector machines (hard margin)

Reminders on Lagrange multipliers

The minimization of a convex function $f(\boldsymbol{x})$, under constraints $g_i(\boldsymbol{x}) \leq 0$, $i = 1, \ldots, n$, is equivalent to the search for the saddle point of the Lagrangian

$$L(\boldsymbol{x}; \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_{i=1}^{n} \alpha_i \, g_i(\boldsymbol{x}).$$

The minimum is taken with respect to $\boldsymbol{x}$. The maximum is relative to Lagrange's n multiplicateurs α_i, which must be positive or null.

The so-called Karush-Kuhn-Tucker conditions are satisfied at the optimum:

$$\alpha_i^* \, g_i(\boldsymbol{x}^*) = 0, \qquad i = 1, \ldots, n.$$

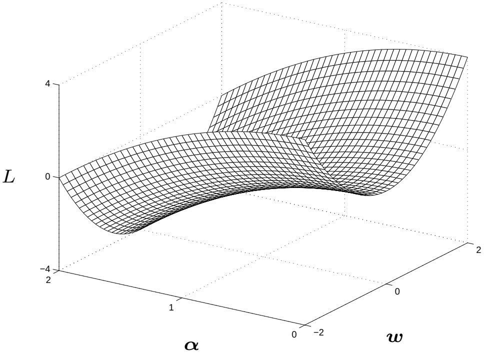
Support vector machines (hard margin)

Resolution by the Lagrangian method

The above problem is solved using the Lagrangian method

$$L(\boldsymbol{w}, b; \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \{y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) - 1\}, \quad \alpha_i \geq 0.$$

The function L needs to be minimized with respect to the primal variables $\boldsymbol{w}$ and b, and maximized with respect to the dual variables α_i.



Support vector machines (hard margin)

Dual problem formulation

The optimality conditions formulated by considering the Lagrangian,

$$L(\boldsymbol{w}, b; \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \{y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) - 1\},$$

result in null derivatives with respect to the primal and dual variables:

$$\frac{\partial}{\partial \boldsymbol{w}} L(\boldsymbol{w}, b; \boldsymbol{\alpha}) = 0 \qquad \frac{\partial}{\partial b} L(\boldsymbol{w}, b; \boldsymbol{\alpha}) = 0.$$

A quick calculation leads to the following relationships which, injected into the Lagrangian expression, provides the dual problem to be solved:

$$\sum_{i=1}^{n} \alpha_i^*\, y_i = 0 \qquad \boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^*\, y_i\, \boldsymbol{x}_i.$$

The dual optimization problem is finally expressed as:

Maximize $W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i\, \alpha_j\, y_i\, y_j\, \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle$

subject to $\sum_{i=1}^{n} \alpha_i\, y_i = 0, \quad \alpha_i \geq 0, \qquad \forall i = 1, \ldots, n.$

Support vector machines (hard margin)

Formulation of the solution and support vectors

The normal vector to the optimum separator plane is expressed as:

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* \, y_i \, \boldsymbol{x}_i$$

From the Karush-Kuhn-Tucker conditions, we have at the optimum:

$$\alpha_i^* \{ y_i(\langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle + b^*) - 1 \} = 0.$$

Case 1: $y_i(\langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle + b^*) > 1$
we have $\alpha_i^* = 0$, which means that $\boldsymbol{x}_i$ is not present in the expression of $\boldsymbol{w}^*$.

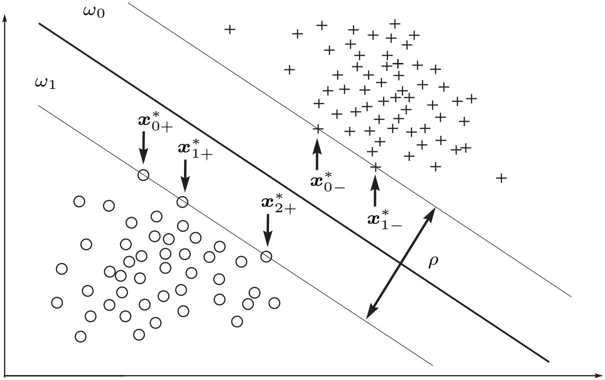
Case 2: $y_i(\langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle + b^*) = 1$
We have $\alpha_i^* \neq 0$ and $\boldsymbol{x}_i$ is on the margin. We deduce b^* from such samples.

The vector $\boldsymbol{w}^*$ is defined only from the $\boldsymbol{x}_i$ located on the margin, the so-called *Support Vectors*.

Support vector machines (hard margin)

Support vectors

The support vectors are indicated below by the arrows.



Generalization performance of SVM

The fact that the optimum hyperplane is expressed only in terms of the support vectors is remarkable because, in general, their number is small.

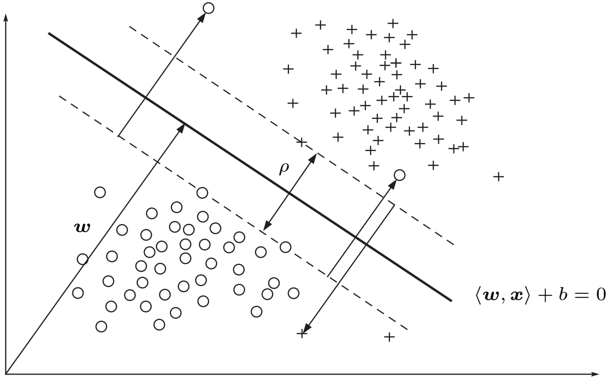
The number n_{sv} of support vectors allows to estimate the generalization performance of the classifier:

$$E\{P_e\} \leq \frac{E\{n_{sv}\}}{n}$$

Support vector machines (soft margin)

Penalty

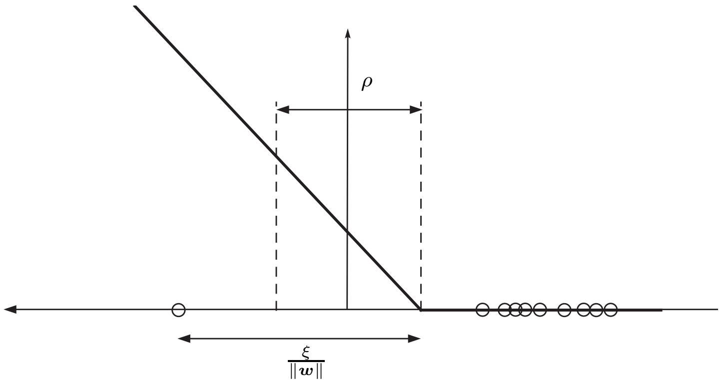
When the competing classes are not linearly separable, the formulation of the problem needs to be modified in order to penalize misclassified data.



Support vector machines (soft margin)

Penalization functions

The most common mode of penalization is related to the distance of the misclassified sample to the margin. Its square value is sometimes considered.



Support vector machines (soft margin)

Formulation of the optimization problem

The previous diagram motivates to reformulate the problem of optimization as follows.

$$\textit{Minimize } \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i, \quad C \geq 0$$

$$\textit{subject to } y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \qquad 1 \leq i \leq n.$$

The term $C\sum_{i=1}^{n}\xi_i$ has the effect of penalizing badly classified samples. Other penalization functions exist as well.

Support vector machines (soft margin)

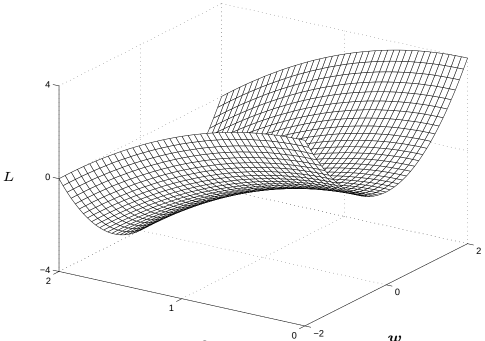
Resolution with the Lagrangian method

The above problem is solved using the Lagrangian method

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

where the α_i and β_i are the Lagrange multipliers, positive or nul.

The function L has to be minimized with respect to the primal variables $\boldsymbol{w}$ and b, and maximized with respect to the dual variables α_i and β_i.



Support vector machines (soft margin)

Formulation of the dual problem

The optimality conditions, defined by the Lagrangian formulation

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

can be interpreted by nullifying the derivatives with respect to the primal and dual variables:

$$\frac{\partial}{\partial \boldsymbol{w}} L(\boldsymbol{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0 \quad \Longrightarrow \quad \boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^*\, y_i\, \boldsymbol{x}_i$$

$$\frac{\partial}{\partial b} L(\boldsymbol{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0 \quad \Longrightarrow \quad \sum_{i=1}^{n} \alpha_i^*\, y_i = 0$$

$$\frac{\partial}{\partial \boldsymbol{\xi}} L(\boldsymbol{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0 \quad \Longrightarrow \quad \beta_i^* = C - \alpha_i^*$$

Injected in the Lagrangian expression, these relations provide the dual problem to be solved.

Support vector machines (soft margin)

Formulation of the dual problem and solution

The dual optimization problem is finally expressed as follows:

$$\textit{Minimize } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \, \alpha_j \, y_i \, y_j \, \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$$

$$\textit{subject to } \sum_{i=1}^{n} \alpha_i \, y_i = 0, \quad 0 \leq \alpha_i \leq C, \qquad \forall i = 1, \ldots, n.$$

The solution of the problem is finally written as

$$\psi(\boldsymbol{x}; \boldsymbol{\alpha}^*, b^*) = \text{sign}\left(\sum_{sv} \alpha_i^* \, y_i \, \langle \boldsymbol{x}, \boldsymbol{x}_i \rangle + b^* \right)$$

To determine b^*, we use the Karush-Kuhn-Tucker conditions:

$$\alpha_i^* \{ y_i(\langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle + b^*) - 1 + \xi_i^* \} = 0, \qquad \beta_i^* \, \xi_i^* = 0.$$

For any support vector $\boldsymbol{x}_i$ such as $\alpha_i < C$, we have $\xi_i = 0$ and $b^* = y_i - \langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle$.

Support vector machines (soft margin)

Choice of the parameter C

Minimize $\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i, \quad C \geq 0$

subject to $y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \qquad 1 \leq i \leq n.$

The parameter C makes a tradeoff between the width of the margin, which has a regularizing role, and the number of misclassified samples.

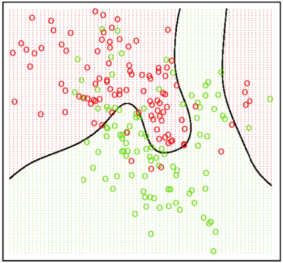
Big $\mathbf{C}$*: small margin, less errors in classification*

Small $\mathbf{C}$*: large margin, more errors in classification*

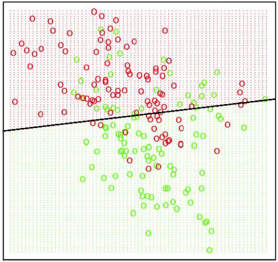
The value of the parameter C can be determined by cross-validation.

Examples of implementation

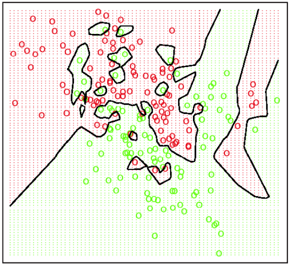
Bayes classifier



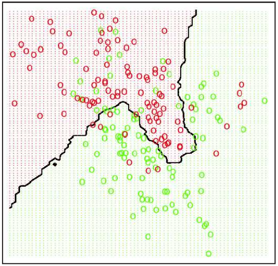
Regression 0/1



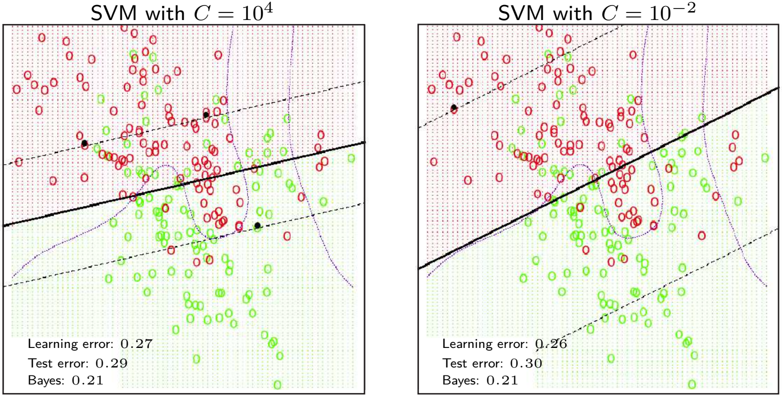
1-ppv



15-ppv



Examples of implementation
Support vector machines



Outline

Nonlinear Support Vector Machines

Nonlinear Support Vector Machines

From linear to nonlinear

Linear classifiers have limited classification capabilities. To remedy this, we can implement them after a nonlinear transformation of the data.

$$\boldsymbol{x} \longrightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots]^t$$

where the $\varphi_i(\boldsymbol{x})$ are the nonlinear functions that are chosen beforehand.

A classifier that is linear in $\boldsymbol{\phi}(\boldsymbol{x})$ are nonlinear with respect to $\boldsymbol{x}$

Nonlinear Support Vector Machines

Example: polynomial classifier

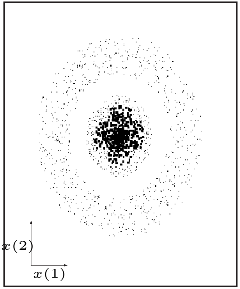
Let $\boldsymbol{x} = [x(1)\ x(2)\ x(3)]^t$. Consider the following transformation:

$$\begin{array}{lll} \phi_1(\boldsymbol{x}) = x(1) & \phi_4(\boldsymbol{x}) = x(1)^2 & \phi_7(\boldsymbol{x}) = x(1)\,x(2) \\ \phi_2(\boldsymbol{x}) = x(2) & \phi_5(\boldsymbol{x}) = x(2)^2 & \phi_8(\boldsymbol{x}) = x(1)\,x(3) \\ \phi_3(\boldsymbol{x}) = x(3) & \phi_6(\boldsymbol{x}) = x(3)^2 & \phi_9(\boldsymbol{x}) = x(2)\,x(3) \end{array}$$

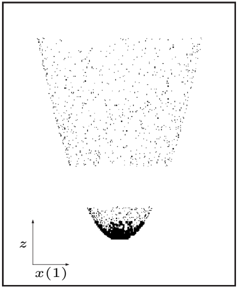
A classifier, linear in the transformed space $\{\boldsymbol{\phi}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathbb{R}^3}$, namely

$$\psi(\boldsymbol{x}; \boldsymbol{w}, b) = \mathrm{sign}(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x})\rangle + b),$$

is a polynomial classifier of degree 2 with respect to $\boldsymbol{x}$.



$z = x(1)^2 + x(2)^2$



The polynomial transformation makes the data linearly separable.

Nonlinear Support Vector Machines

Optimization in a transformed space

The dual optimization problem is thus expressed as:

$$\textit{Minimize } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \, \alpha_j \, y_i \, y_j \, \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle$$

$$\textit{subject to } \sum_{i=1}^{n} \alpha_i \, y_i = 0, \quad 0 \leq \alpha_i \leq C, \qquad \forall i = 1, \ldots, n.$$

The solution can be written as

$$\psi(\boldsymbol{x}; \boldsymbol{\alpha}^*, b^*) = \operatorname{sign}\left(\sum_{sv} \alpha_i^* \, y_i \, \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}_i) \rangle + b^* \right).$$

We can see that

- we never need to explicitly compute $\phi(\boldsymbol{x})$;
- if $\boldsymbol{x}$ has a big dimension, the dimension of $\phi(\boldsymbol{x})$ is even bigger, sometimes infinite.

Nonlinear Support Vector Machines

The kernel trick

If you can define a kernel $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$ such that:

- the associated decision rule is efficient

$$\psi(\boldsymbol{x}; \boldsymbol{\alpha}^*, b^*) = \text{sign}\left(\sum_{sv} \alpha_i^* \, y_i \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) + b^* \right),$$

- it is easy to compute $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, even for large-dimension data,

then that's it !

Nonlinear Support Vector Machines

Polynomial kernels

In the case of a 2-degree polynomial transformation, it is easy to show that:

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle = (1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle)^2 \triangleq \kappa(\boldsymbol{x}, \boldsymbol{x}')$$

▷ **The inner product can be computed in $\mathbb{R}^2$!**

More generally, one is interested in $\kappa(\boldsymbol{x}, \boldsymbol{x}') = (1 + \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle)^q$, with $\boldsymbol{x} \in \mathbb{R}^d$.

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = (1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle)^q = \sum_{j=0}^{q} \binom{q}{j} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle^j .$$

Each component $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle^j = [x(1)\, x'(1) + \ldots + x(d)\, x'(d)]^j$ of this expression can be expanded into a weighted sum of monomials of degree j of the form

$$[x(1)\, x'(1)]^{j_1} [x(2)\, x'(2)]^{j_2} \ \ldots \ [x(d)\, x'(d)]^{j_d}$$

with $\sum_{i=1}^{d} j_i = j$. This directly leads to the expression of $\phi(\boldsymbol{x})$...

The kernel trick

Mercer theorem

We are interested in the functions $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ that can act as an inner product in a space $\mathcal{H}$. We call *kernel* a symmetric function κ from $\mathcal{X} \times \mathcal{X}$ dans $\mathbb{R}$.

Theorem (Mercer)

If κ is a continuous kernel of a positive definite integral operator, which means that

$$\int\int \varphi(\boldsymbol{x})\, \kappa(\boldsymbol{x}, \boldsymbol{x}')\, \varphi^*(\boldsymbol{x}')\, d\boldsymbol{x}\, d\boldsymbol{x}' \geq 0$$

for all $\varphi \in \mathcal{L}^2(\mathcal{X})$, it can be decomposed into the forme

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{\infty} \lambda_i\, \psi_i(\boldsymbol{x})\, \psi_i(\boldsymbol{x}'),$$

where ψ_i and λ_i are the eigenfunctions (orthogonal) and eigenvalues (positive) of the kernel κ, respectively, such that

$$\int \kappa(\boldsymbol{x}, \boldsymbol{x}')\, \psi_i(\boldsymbol{x})\, d\boldsymbol{x} = \lambda_i\, \psi_i(\boldsymbol{x}').$$

The kernel trick

Mercer theorem

It is easy to see that a kernel κ satisfying Mercer's theorem can act as a inner product in a transformed space $\mathcal{H}$. Just write:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} \sqrt{\lambda_1}\,\psi_1(\boldsymbol{x}) \\ \sqrt{\lambda_2}\,\psi_2(\boldsymbol{x}) \\ \cdots \end{pmatrix}$$

In these conditions, we check that we have: $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle = \kappa(\boldsymbol{x}, \boldsymbol{x}')$.

We define the space $\mathcal{H}$ as being generated by the eigenfunctions ψ_i of the kernel κ, that is to say

$$\mathcal{H} = \{f(\cdot) \mid f(x) = \sum_{i=1}^{\infty} \alpha_i\ \psi_i(x),\ \alpha_i \in \mathbb{R}\}.$$

The kernel trick

Examples de Mercer kernels

We can show that the following kernels satisfy the Mercer condition, and therefore correspond to an inner product in a space $\mathcal{H}$.

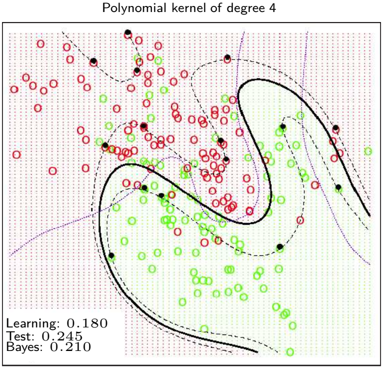
Projective kernels	
monomial of degree q	$\langle \boldsymbol{x}, \boldsymbol{x}' \rangle^q$
polynomial of degree q	$(1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle)^q$
sigmoidal	$\frac{1}{\eta_0} \tanh(\beta_0 \langle \boldsymbol{x}, \boldsymbol{x}' \rangle - \alpha_0)$

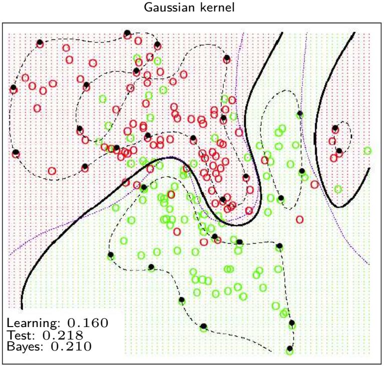
Radial kernels	
Gaussian	$\exp(-\frac{1}{2\sigma_0^2} \Vert \boldsymbol{x} - \boldsymbol{x}' \Vert^2)$
exponential	$\exp(-\frac{1}{2\sigma_0^2} \Vert \boldsymbol{x} - \boldsymbol{x}' \Vert)$
uniforme	$\frac{1}{\eta_0} \mathbb{1}_{\Vert \boldsymbol{x} - \boldsymbol{x}' \Vert \leq \beta_0}$
Epanechnikov	$\frac{1}{\eta_0} (\beta_0^2 - \Vert \boldsymbol{x} - \boldsymbol{x}' \Vert^2) \mathbb{1}_{\Vert \boldsymbol{x} - \boldsymbol{x}' \Vert \leq \beta_0}$
Cauchy	$\frac{1}{\eta_0} \frac{1}{1 + \Vert \boldsymbol{x} - \boldsymbol{x}' \Vert^2 / \beta_0^2}$

... and more $\kappa_1(\boldsymbol{x}, \boldsymbol{x}') + \kappa_2(\boldsymbol{x}, \boldsymbol{x}')$, $\kappa_1(\boldsymbol{x}, \boldsymbol{x}') \cdot \kappa_2(\boldsymbol{x}, \boldsymbol{x}')$, ...

Nonlinear Support Vector Machines

Example of implementation





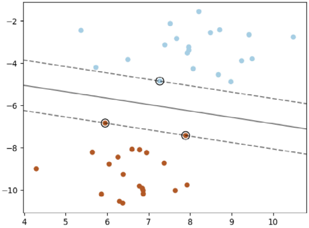
Outline

Python implementation

Illustration of Maximum Margin Separating Hyperplane in SVM

Plot separating hyperplane

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.datasets import make_blobs


# we create 40 separable points
X, y = make_blobs(n_samples=40, centers=2, random_state=6)

# fit the model, don't regularize for illustration purposes
clf = svm.SVC(kernel='linear', C=1000)
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
# plot support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none', edgecolors='k')
plt.show()
```

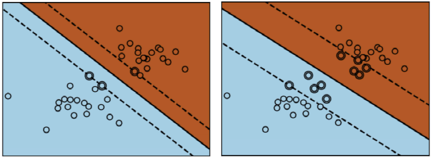


Margins for different C values

Plot SVM margin

```
# Code source: Gael Varoquaux
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# we create 40 separable points
np.random.seed(0)
X = np.r_[np.random.randn(20, 2)
        - [2, 2], np.random.randn(20, 2) + [2, 2]]
Y = [0] * 20 + [1] * 20

# figure number
fignum = 1

# fit the model
for name, penalty in (('unreg', 1), ('reg', 0.05)):

    clf = svm.SVC(kernel='linear', C=penalty)
    clf.fit(X, Y)

    # get the separating hyperplane
    w = clf.coef_[0]
    a = -w[0] / w[1]
    xx = np.linspace(-5, 5)
    yy = a * xx - (clf.intercept_[0]) / w[1]

    # plot the parallels to the separating hyperplane that pass
    # through the support vectors (margin away from hyperplane
    # in direction perpendicular to it). This is sqrt(1+a^2) away
    # vertically in 2-d.
    margin = 1 / np.sqrt(np.sum(clf.coef_ ** 2))
    yy_down = yy - np.sqrt(1 + a ** 2) * margin
    yy_up = yy + np.sqrt(1 + a ** 2) * margin

    # plot the line, points, and nearest vectors
    plt.figure(fignum, figsize=(4, 3))
    plt.clf()
    plt.plot(xx, yy, 'k-')
    plt.plot(xx, yy_down, 'k--')
    plt.plot(xx, yy_up, 'k--')
```

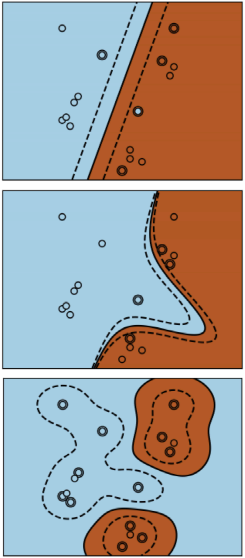
```
    plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
                s=80, facecolors='none', zorder=10, edgecolors='k')
    plt.scatter(X[:, 0], X[:, 1], c=Y, zorder=10,
                cmap=plt.cm.Paired, edgecolors='k')

    plt.axis('tight')
    x_min = -4.8
    x_max = 4.2
    y_min = -6
    y_max = 6

    XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
    Z = clf.predict(np.c_[XX.ravel(), YY.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(XX.shape)
    plt.figure(fignum, figsize=(4, 3))
    plt.pcolormesh(XX, YY, Z, cmap=plt.cm.Paired)

    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)

    plt.xticks(())
    plt.yticks(())
    fignum = fignum + 1

plt.show()
```



Nonlinear SVM

Plot SVM kernels

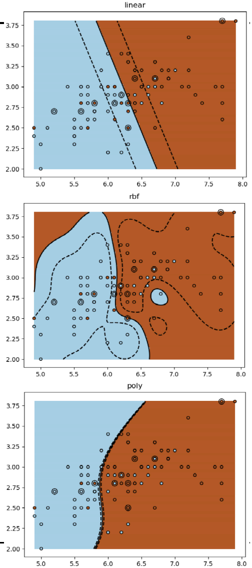
```
# Code source: Gael Varoquaux; License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# Our dataset and targets
X = np.c_[(.4, -.7),
          (-1.5, -1),
          (-1.4, -.9),
          (-1.3, -1.2),
          (-1.1, -.2),
          (-1.2, -.4),
          (-.5, 1.2),
          (-1.5, 2.1),
          (1, 1),
          # --
          (1.3, .8),
          (1.2, .5),
          (.2, -2),
          (.5, -2.4),
          (.2, -2.3),
          (0, -2.7),
          (1.3, 2.1)].T
Y = [0] * 8 + [1] * 8

fignum = 1 # figure number

# fit the model
for kernel in ('linear', 'poly', 'rbf'):
    clf = svm.SVC(kernel=kernel, gamma=2)
    clf.fit(X, Y)

    # plot the line, the points, and the nearest vectors to the plane
    plt.figure(fignum, figsize=(4, 3))
    plt.clf()

    plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=80, facecolors='none', zorder=10, edgecolors='k')
    plt.scatter(X[:, 0], X[:, 1], c=Y, zorder=10, cmap=plt.cm.Paired, edgecolors='k')

    plt.axis('tight')
    x_min = -3
    x_max = 3
    y_min = -3
    y_max = 3

    XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
    Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(XX.shape)
    plt.figure(fignum, figsize=(4, 3))
    plt.pcolormesh(XX, YY, Z > 0, cmap=plt.cm.Paired)
    plt.contour(XX, YY, Z, colors=['k', 'k', 'k'], linestyles=['--', '-', '--'], levels=[-.5, 0, .5])

    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)

    plt.xticks(())
    plt.yticks(())
    fignum = fignum + 1
plt.show()
```



Nonlinear SVM on IRIS Dataset

Plot IRIS exercice

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, svm

iris = datasets.load_iris()
X = iris.data
y = iris.target

X = X[y != 0, :2]
y = y[y != 0]

n_sample = len(X)

np.random.seed(0)
order = np.random.permutation(n_sample)
X = X[order]
y = y[order].astype(np.float)

X_train = X[:int(.9 * n_sample)]
y_train = y[:int(.9 * n_sample)]
X_test = X[int(.9 * n_sample):]
y_test = y[int(.9 * n_sample):]

# fit the model
for fig_num, kernel in enumerate(('linear', 'rbf', 'poly')):
    clf = svm.SVC(kernel=kernel, gamma=10)
    clf.fit(X_train, y_train)

    plt.figure(fig_num)
    plt.clf()
    plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10,
                cmap=plt.cm.Paired, edgecolor='k', s=20)

    # Circle out the test data
    plt.scatter(X_test[:, 0], X_test[:, 1], s=80,
                facecolors='none', zorder=10, edgecolor='k')

    plt.axis('tight')
    x_min = X[:, 0].min()
    x_max = X[:, 0].max()
    y_min = X[:, 1].min()
    y_max = X[:, 1].max()

    XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
    Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(XX.shape)
    plt.pcolormesh(XX, YY, Z > 0, cmap=plt.cm.Paired)
    plt.contour(XX, YY, Z, colors=['k', 'k', 'k'],
                linestyles=['--', '-', '--'], levels=[-.5, 0, .5])

    plt.title(kernel)
plt.show()
```

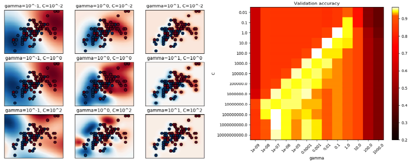


Parameters of RBF SVM

Plot RBF parameters

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Normalize

from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV


# Utility function to move the midpoint of a colormap to be around
# the values of interest.
class MidpointNormalize(Normalize):

    def __init__(self, vmin=None, vmax=None, midpoint=None, clip=False):
        self.midpoint = midpoint
        Normalize.__init__(self, vmin, vmax, clip)

    def __call__(self, value, clip=None):
        x, y = [self.vmin, self.midpoint, self.vmax], [0, 0.5, 1]
        return np.ma.masked_array(np.interp(value, x, y))
# #############################################################################
# Load and prepare data set:
# dataset for grid search

iris = load_iris()
X = iris.data
y = iris.target

# Dataset for decision function visualization: we only keep the first two
# features in X and sub-sample the dataset to keep only 2 classes and
# make it a binary classification problem.

X_2d = X[:, :2]
X_2d = X_2d[y > 0]
y_2d = y[y > 0]
y_2d -= 1

# It is usually a good idea to scale the data for SVM training.
# We are cheating a bit in this example in scaling all of the data,
# instead of fitting the transformation on the training set and
# just applying it on the test set.

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_2d = scaler.fit_transform(X_2d)
# #############################################################################
# Train classifiers:
# For an initial search, a logarithmic grid with basis 10 is often helpful.
# Using a basis of 2, a finer tuning can be achieved but at a much higher cost.

C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
param_grid = dict(gamma=gamma_range, C=C_range)
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
grid.fit(X, y)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))

# Now we need to fit a classifier for all parameters in the 2d version
# (we use a smaller set of parameters here because it takes a while to train)

C_2d_range = [1e-2, 1, 1e2]
gamma_2d_range = [1e-1, 1, 1e1]
classifiers = []
for C in C_2d_range:
    for gamma in gamma_2d_range:
        clf = SVC(C=C, gamma=gamma)
        clf.fit(X_2d, y_2d)
        classifiers.append((C, gamma, clf))
```

```python
# #############################################################################
# Visualization:
# draw visualization of parameter effects

plt.figure(figsize=(8, 6))
xx, yy = np.meshgrid(np.linspace(-3, 3, 200), np.linspace(-3, 3, 200))
for (k, (C, gamma, clf)) in enumerate(classifiers):
    # evaluate decision function in a grid
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # visualize decision function for these parameters
    plt.subplot(len(C_2d_range), len(gamma_2d_range), k + 1)
    plt.title("gamma=10^%d, C=10^%d" % (np.log10(gamma), np.log10(C)),
              size='medium')

    # visualize parameter's effect on decision function
    plt.pcolormesh(xx, yy, -Z, cmap=plt.cm.RdBu)
    plt.scatter(X_2d[:, 0], X_2d[:, 1], c=y_2d, cmap=plt.cm.RdBu_r,
                edgecolors='k')
    plt.xticks(())
    plt.yticks(())
    plt.axis('tight')

scores = grid.cv_results_['mean_test_score'].reshape(len(C_range), len(gamma_range))

# Draw heatmap of the validation accuracy as a function of gamma and C:
# The score are encoded as colors with the hot colormap which varies from dark
# red to bright yellow. As the most interesting scores are all located in the
# 0.92 to 0.97 range we use a custom normalizer to set the mid-point to 0.92 so
# as to make it easier to visualize the small variations of score values in the
# interesting range while not brutally collapsing all the low score values to
# the same color.

plt.figure(figsize=(8, 6))
plt.subplots_adjust(left=.2, right=0.95, bottom=0.15, top=0.95)
plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot,
           norm=MidpointNormalize(vmin=0.2, midpoint=0.92))
plt.xlabel('gamma')
plt.ylabel('C')
plt.colorbar()
plt.xticks(np.arange(len(gamma_range)), gamma_range, rotation=45)
plt.yticks(np.arange(len(C_range)), C_range)
plt.title('Validation accuracy')
plt.show()
```



Output

```
The best parameters are 'C': 1.0, 'gamma': 0.1 with a score of 0.97
```

Outline

Epilogue

Assessment on the SVM

Compared to competing techniques such as neural networks, SVMs have great qualities:

1. Unique solution
   $\longrightarrow$ Quadratic problem

2. Integrated regularization process, sparse solution
   $\longrightarrow$ Cost function and resulting inequality constraints

3. Easy extension to the non-linear case, non black-box solution
   $\longrightarrow$ Kernel trick

Concluding remarks

Outline

Final Remark

Outline: Next