SVM
Support Vector Machine

Supervised Learning - Classification

Ricco Rakotomalala

Université Lumière Lyon 2

Outline

Binary classification

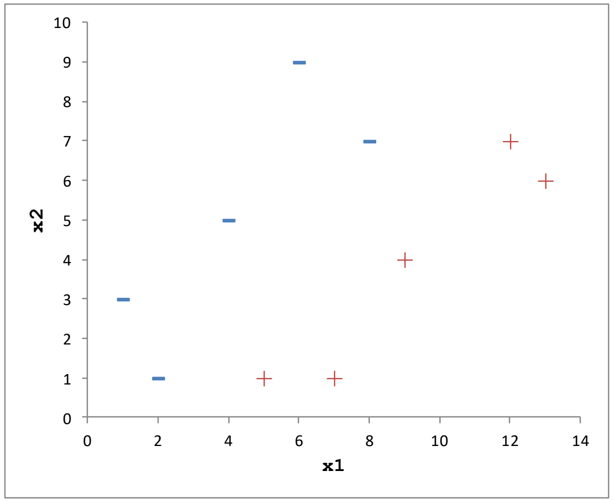
LINEAR SVM

Binary classification

Data linearly separable

Supervised learning : $Y = f(x_1,x_2,\ldots,x_p ; \beta)$ for a binary problem i.e. $Y \in \{+, -\}$ or $Y \in \{+1, -1\}$

x1	x2	y
1	3	-1
2	1	-1
4	5	-1
6	9	-1
8	7	-1
5	1	1
7	1	1
9	4	1
12	7	1
13	6	1



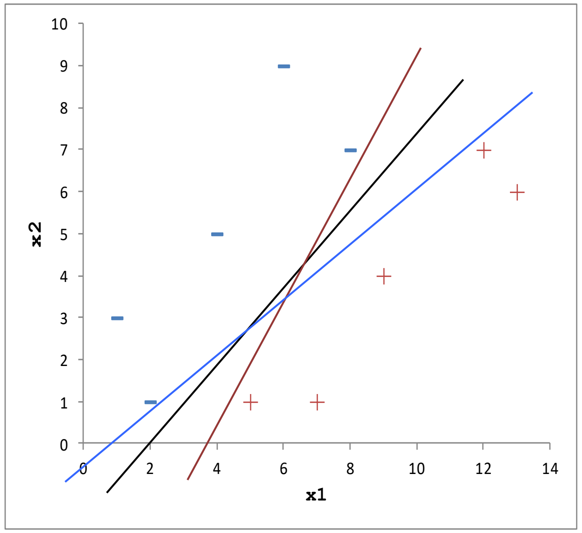
The aim is to find a hyperplane which enables to separate perfectly the "+" and "-". The classifier comes in the form of a linear combination of the variables.

$\beta=(\beta_1, \beta_2,\ldots, \beta_p)$ and β_0 are the (p+1) parameters (coefficients) to estimate.

$$f(x) = x^T\beta + \beta_0$$
$$= x_1\beta_1 + x_2\beta_2 + \cdots + \beta_0$$

Finding the “optimal” solution

Once the "shape" of the decision boundary defined, we have to choose a solution among the infinite number of possible solutions.



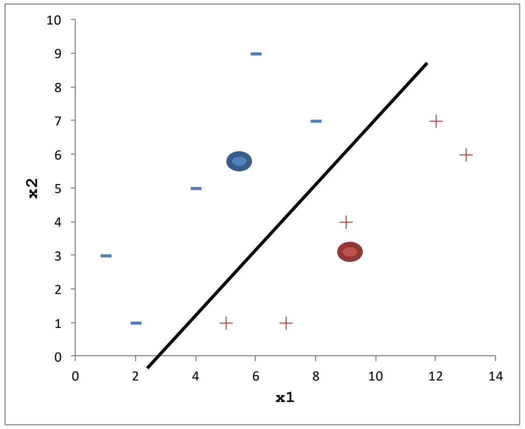
Two keys issues always in the supervised learning framework:

(1) Choosing the “Representation bias” or “hypothesis bias” ➔ we define the shape of the separator

(2) Choosing the search bias i.e. the way to select the best solution among all the possible solutions ➔ it often boils down to set the objective function to optimize

Example: Linear Discriminant Analysis

The separating hyperplane is to halfway between the two conditional centroids within the meaning of the Mahalanobis distance.



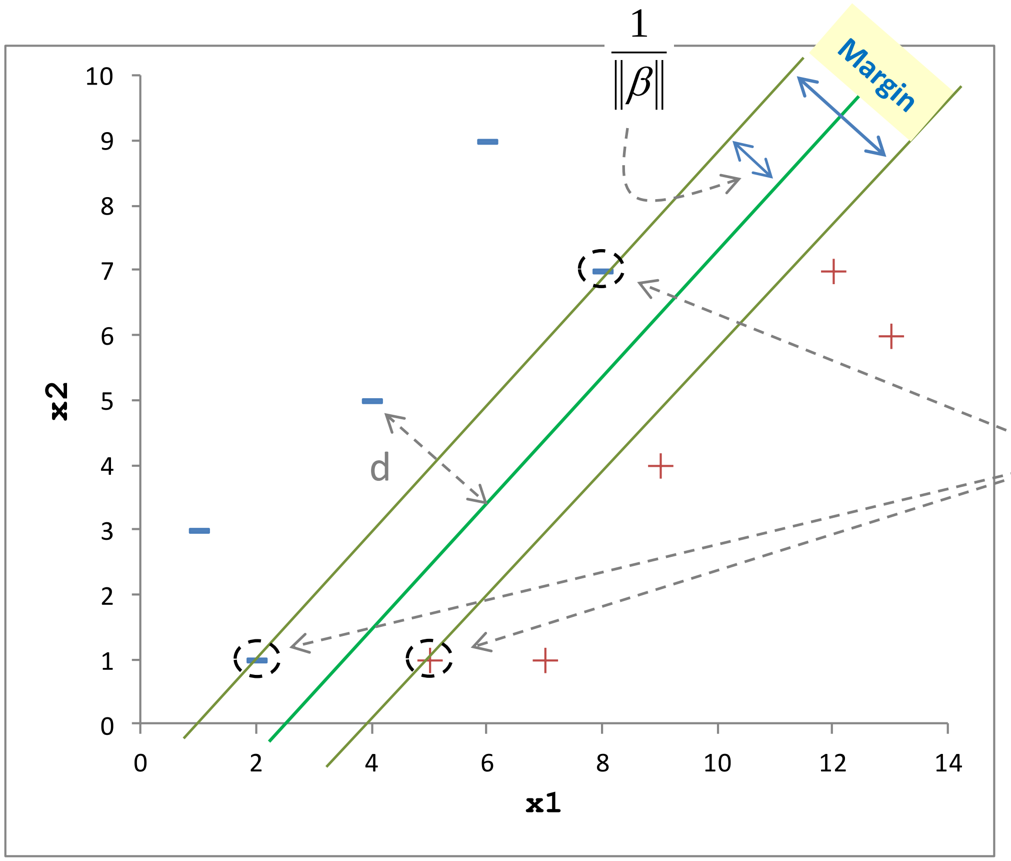
Primal problem

MAXIMIZE THE MARGIN (I)

Hard-margin principle

Intuitive layout

The optimal separating hyperplane separates the two classes and maximizes the distance to the closest point from either class (Vapnik, 1996) [HTF, page 132]



- Distance from any point x with the boundary (see projection)

$$d = \frac{\left|x^T \beta + \beta_0\right|}{\|\beta\|}$$

- The maximum margin is

$$\delta = \frac{2}{\|\beta\|}$$

- The instances from which rely the margins are "support vectors". If we remove them from the sample, the optimal solution is modified.

- Several areas are defined in the representation space

f(x) = 0, we have the maximum margin hyperplane

f(x) > 0, the area of « + » instances

f(x) < 0, the area of « - » instances

f(x) = +1 or -1, these hyperplanes are the margins

Maximizing the margin

Mathematical formulation

Maximize the margin is equivalent to minimize the norm of the vector of parameters β

$$\max \frac{2}{\|\beta\|} \Leftrightarrow \min \|\beta\|$$

$$\min_{\beta,\beta_0} \|\beta\|$$

Subject to

$$y_i \times f(x_i) \geq 1,\ i = 1, \ldots, n$$

- Norm: $$\|\beta\| = \sqrt{\beta_1^2 + \cdots + \beta_p^2}$$

- Constraints point out that all points are on the right side, at least they are on the hyperplane of support vectors.

- We have a problem of convex optimization (quadratic objective function, linear constraints). A global optimum exists.

- But there is no literal solution. It must pass through numerical optimization programs.

Note: There are often also writing

$$\min_{\beta,\beta_0} \frac{1}{2} \|\beta\|^2$$

Maximizing the margin

A toy example under **EXCEL (!)**

We use the SOLVER to solve the optimization problem.

	beta.1	beta.2	beta.0
	0.667	-0.667	-1.667

(p + 1) variable cells

n°	x1	x2	y	f(x)	f(x)*y
1	1	3	-1	-3	3
2	2	1	-1	-1	1
3	4	5	-1	-2.33333333	2.33333333
4	6	9	-1	-3.66666667	3.66666667
5	8	7	-1	-1	1
6	5	1	1	1	1
7	7	1	1	2.33333333	2.33333333
8	9	4	1	1.66666667	1.66666667
9	12	7	1	1.66666667	1.66666667
10	13	6	1	3	3

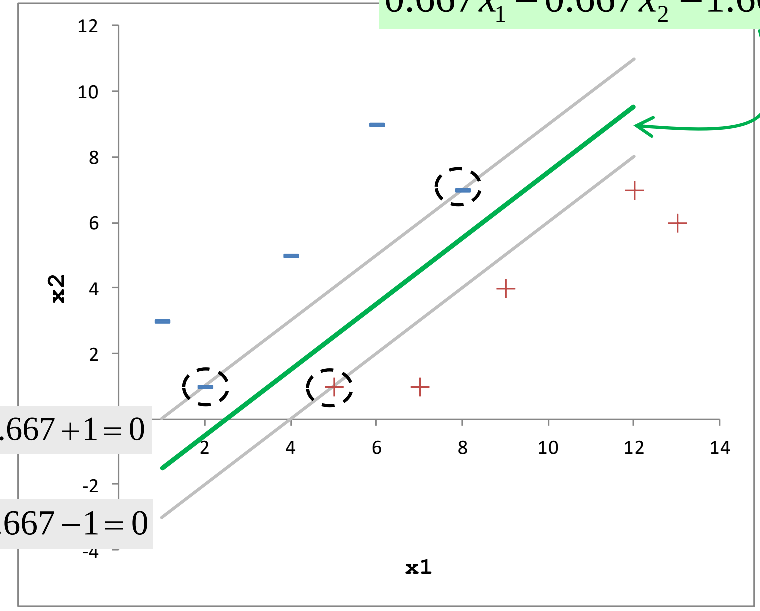
Saturated constraints: 3 support vectors were found (n°2, 5 et 6)

Norme.Beta	0.943

Objective cell: $\|\beta\|$

n = 10 constraints

$$0.667x_1 - 0.667x_2 - 1.667 = 0$$



$$x^T\beta + \beta_0 \pm 1 = 0$$

$$0.667x_1 - 0.667x_2 - 1.667 + 1 = 0$$

$$0.667x_1 - 0.667x_2 - 1.667 - 1 = 0$$

Primal problem

Comments

Rule assignment for the instance i* based on the estimated coefficients $\hat{\beta}_j$

$$f(x_{i*})\begin{cases}\geq 0 \Rightarrow \hat{y}_{i*}=1\\ <0 \Rightarrow \hat{y}_{i*}=-1\end{cases}$$

beta.1	beta.2	beta.0
0.667	-0.667	-1.667

x1	x2	y	f(x)	prediction
1	3	-1	-3	-1
2	1	-1	-1	-1
4	5	-1	-2.3333	-1
6	9	-1	-3.6667	-1
8	7	-1	-1	-1
5	1	1	1	1
7	1	1	2.33333	1
9	4	1	1.66667	1
12	7	1	1.66667	1
13	6	1	3	1

Drawbacks of this primal form

(1) Algorithms for numerical optimization (quadratic prog.) are not operational when “p” is large (> a few hundred). This often happens when we handle real problems (e.g. text mining, image,...) (few examples, many descriptors)

(2) This primal form does not highlight the possibility of using "kernel" functions that enable to go beyond to the linear classifiers

Dual problem

MAXIMIZE THE MARGIN (II)

Dual problem

Lagrangian multiplier method

A convex optimization problem has a dual form by using the Lagrange multipliers.

The primal problem…

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2$$
$$s.c. \quad y_i \times (x_i^T\beta + \beta_0) \geq 1, \forall i = 1,\ldots,n$$

…becomes under the dual form

$$L_P(\beta,\beta_0,\alpha) = \tfrac{1}{2}\|\beta\|^2 - \sum_{i=1}^{n}\alpha_i\left[y_i \times \left(x_i^T\beta + \beta_0\right) - 1\right]$$

Where α_i are the Lagrange multipliers

By setting each partial derivate equal to zero

$$\left\{\begin{aligned}
&\frac{\partial L}{\partial \beta} = \beta - \sum_{i=1}^{n}\alpha_i y_i x_i = 0 \\
&\frac{\partial L}{\partial \beta_0} = \sum_{i=1}^{n}\alpha_i y_i = 0 \\
&\frac{\partial L}{\partial \alpha_i} = -y_i \times \left(x_i^T\beta + \beta_0\right) + 1 \leq 0,\ \forall i
\end{aligned}\right.$$

$\Rightarrow$ We can obtain the parameters (coefficients) of the hyperplane from the Lagrange multipliers

The solution must satisfy the Karush-Kuhn-Tucker (KKT) conditions

$$\alpha_i[y_i \times \left(x_i^T\beta + \beta_0\right) - 1] = 0,\ \forall i$$

Dual problem

Optimization

By using information from the partial derivative of the Lagrangian, the results rely only on multipliers

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle$$

Subject to

$$\begin{cases} \alpha_i \geq 0, \forall i \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \end{cases}$$

• $\langle x_i, x_{i'} \rangle$ is the scalar product between the vectors of values for the instances i and i'

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} \times x_{i'j}$$

• $\alpha_i > 0$ define the important instances i.e. the support vectors

• Inevitably, there will be support vectors with different class labels, otherwise this condition cannot be met.

Example

Using Excel again

We use again the SOLVER to solve the optimization problem.

Variable cells α_i

Only the support vectors have a « weight » $\alpha_i > 0$ (n°2, 5 and 6)

The matrix of the scalar product $<x_i,x_{i'}>$ is called Gram matrix

$$\alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle$$

n°	x1	x2	y	alpha	y*alpha
1	1	3	-1	**0**	0
2	2	1	-1	**0.33333**	-0.3333
3	4	5	-1	**0**	0
4	6	9	-1	**0**	0
5	8	7	-1	**0.11111**	-0.1111
6	5	1	1	**0.44444**	0.44444
7	7	1	1	**0**	0
8	9	4	1	**0**	0
9	12	7	1	**0**	0
10	13	6	1	**0**	0
			Somme	0.88889	7.8E-16
			LD	0.44444	

n°	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0.6	0	0	0.9	-1.6	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0.9	0	0	1.4	-2.3	0	0	0	0
6	0	-1.6	0	0	-2.3	5.1	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

Somme	0.889
Racine	0.943

$$\sum_{i=1}^{n} \alpha_i$$

Objective function $L_D(\alpha)$

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

$$\sum_{i=1}^{n}\sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle = 0.889$$

$$\|\beta\| = \sqrt{\sum_{i=1}^{n}\sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle} = 0.943$$

Recall that the margin is equal to $\delta = \frac{2}{\|\beta\|}$

From the support vectors to the hyperplane coefficients (I)

Computing β from the α (Lagrange multipliers)

Since the primal and dual expressions are two facets of the same problem, one must be able to pass from one to the other.

From the partial derivative of the Lagrangian with respect to β

$$\frac{\partial L}{\partial \beta} = \beta - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \Rightarrow \beta = \sum_{i=1}^{n} \alpha_i y_i x_i$$

Only the support vectors points are involved in the calculation of the coefficients, since they are the only ones for which ($\alpha_i > 0$)

Form KKT conditions, computed from any support vector (α_i must be > 0), we can obtain β_0

$$\alpha_i [y_i \times \left(x_i^T \beta + \beta_0\right) - 1] = 0$$

$$\Rightarrow \beta_0 = \frac{1 - y_i x_i^T \beta}{y_i}$$

Note: Because $y_i \in \{-1,+1\}$, we can write also: $$\beta_0 = y_i - x_i^T \beta$$

From the support vectors to the hyperplane coefficients (II)

Numerical example

For β_1, only the variable X_1 participates in the calculations

$$\begin{aligned}\beta_1 &= 0.333\times(-1)\times 2\\ &\quad + 0.111\times(-1)\times 8\\ &\quad + 0.444\times(1)\times 5\\ &= 0.6667\end{aligned}$$

Vecteurs de support

n°	x1	x2	y	alpha
2	2	1	-1	**0.333**
5	8	7	-1	**0.111**
6	5	1	1	**0.444**

beta.1	0.6667
beta.2	-0.6667

beta.0	-1.6667
	-1.6667
	-1.6667

The result is the same whatever the support vector used.

We use the support vector n°2

$$\begin{aligned}\beta_0 &= \frac{1-y_i x_i^T\beta}{y_i}\\ &= \frac{1-(-1)\times\left[0.667\times 2+(-0.667)\times 1\right]}{(-1)}\\ &= -1.6667\end{aligned}$$

Classification of an unseen instance (I)

Using the support vectors

Utilization of the support vectors for the classification of unseen instances. This formulation will be important when we will use the kernel functions.

The classification function can be written based on the coefficients β or the Lagrange multipliers α

$$
\begin{aligned}
f(x) &= x^T \beta + \beta_0 \\
&= \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle + \beta_0 \\
&= \sum_{i' \in S} \alpha_{i'} y_{i'} \langle x_{i'}, x \rangle + \beta_0 \quad \leftarrow\text{- - - -}
\end{aligned}
$$

S is the set of support vectors. There are the only ones which have a weight ($\alpha_i > 0$)

Only the support vectors participate in the classification process!

➡ We have a kind of nearest neighbors algorithm where only the instances corresponding to the support vectors participate to the classification process. These instances are weighted (α_i)

➡ The intercept β_0 can be obtained from the KKT conditions applied on the support vectors (see previous pages)

Classification of an unseen instance (II)

Numerical example

For the classification of the instance n°1

$$
\begin{aligned}
f(x) &= \sum_{i' \in S} \alpha_{i'} y_{i'} \langle x_{i'}, x \rangle + \beta_0 \\
&= 0.333 \times (-1) \times (2 \times 1 + 1 \times 3) + 0.111 \times (-1) \times (8 \times 1 + 7 \times 3) + 0.444 \times (1) \times (5 \times 1 + 1 \times 3) + (-1.667) \\
&= -3.0
\end{aligned}
$$

n°	x1	x2	y	f(x)	prediction
1	1	3	-1	-3.000	-1
2	2	1	-1	-1.000	-1
3	4	5	-1	-2.333	-1
4	6	9	-1	-3.667	-1
5	8	7	-1	-1.000	-1
6	5	1	1	1.000	1
7	7	1	1	2.333	1
8	9	4	1	1.667	1
9	12	7	1	1.667	1
10	13	6	1	3.000	1

Utilization of the 3 support vectors.

Vecteurs de support

n°	x1	x2	y	**alpha**
2	2	1	-1	**0.333**
5	8	7	-1	**0.111**
6	5	1	1	**0.444**

Beta.0	-1.667

Dual form

Comments

1. This formulation is completely consistent with the primal form
2. It highlights the part of the support vectors with the weights α_i
3. It highlights also the importance of the scalar product $<x_i,x_{i'}>$ during the calculations (Gram matrix)
4. Facing the high dimensionality (''p'' is very high, e.g. text mining), this formulation makes calculations tractable for optimization techniques

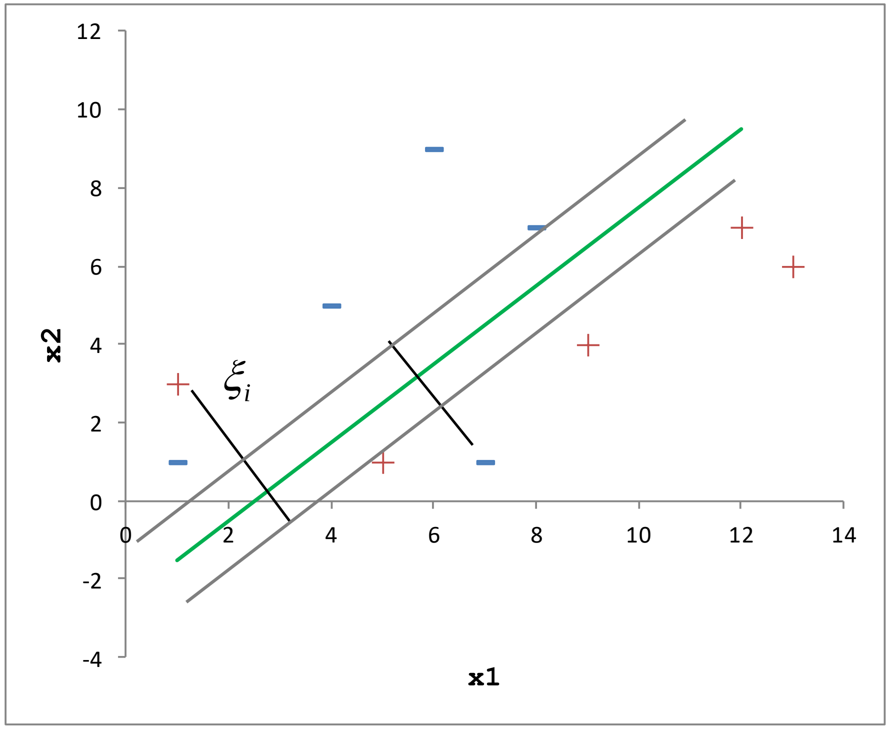
Noisy dataset labels

SOFT MARGIN

"Slack variables"

Using the slack variables ξ_i to handle misclassified instances

In real problems, a perfect classification is not feasible. Some instances are in the wrong side of the margins.



- ξ is a vector of size n
- $\xi_i \geq 0$ marks the misclassified instances
- $\xi_i = 0$, the instance is in the right side of the margin
- $\xi_i < 1$, the instance is in the right side of the maximum margin hyperplane, but it exceeds its margin
- $\xi_i > 1$, the instance is misclassified i.e. it is in the wrong side of the maximum margin hyperplane

Reformulation of the problem

Introduction of the cost parameter “C”

We should penalize the error, more or less strongly depending on whether you want a model that more or less fits to the training data.

Primal form

$$
\begin{aligned}
&\min_{\beta,\beta_0,\xi_i} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}\xi_i \\
&s.c. \\
&y_i \times (x_i^T\beta + \beta_0) \geq 1 - \xi_i, \forall i = 1,\ldots,n \\
&\xi_i \geq 0,\ \forall i
\end{aligned}
$$

The tolerance for errors is more or less accentuated with the **C** parameter (‘’cost’’ parameter)

➔ C is too high: overfitting

➔ C is too low: underfitting

Dual form

$$
\begin{aligned}
&\max_{\alpha} L_D(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n}\alpha_i\alpha_{i'}y_iy_{i'}\langle x_i, x_{i'}\rangle \\
&s.c. \\
&\sum_{i=1}^{n}\alpha_i y_i = 0 \\
&0 \leq \alpha_i \leq C,\ \forall i
\end{aligned}
$$

Soft-margin - An example

beta.1	beta.2	beta.0
0.333	-0.333	-0.667

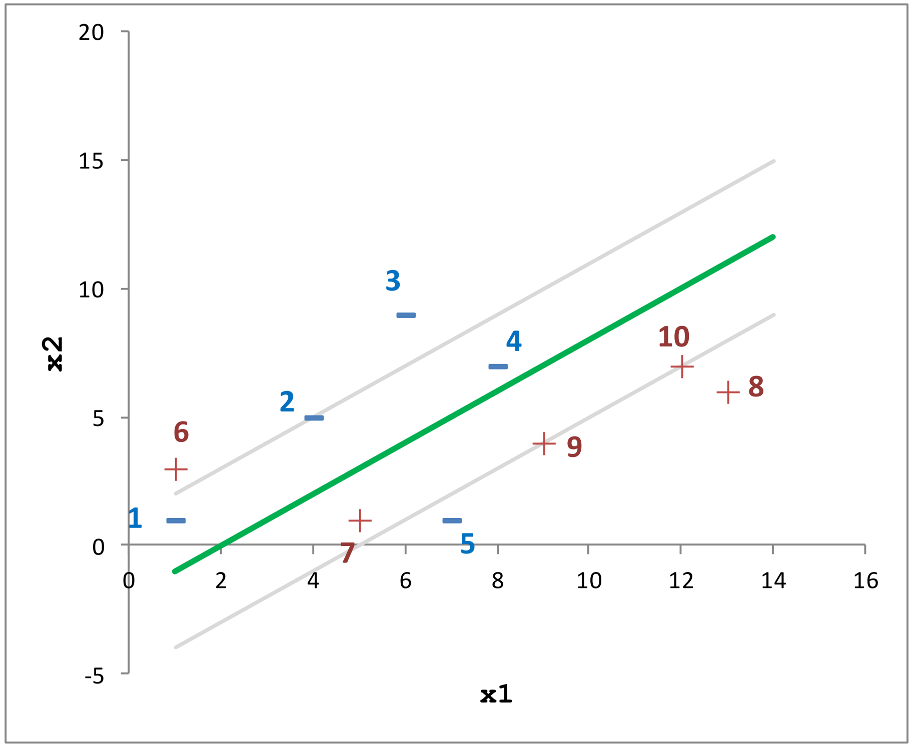
n°	x1	x2	y	ksi	1-ksi	y*f(x)
1	1	1	-1	0.333	0.667	0.667
2	4	5	-1	0	1	1
3	6	9	-1	0	1	1.667
4	8	7	-1	0.667	0.333	0.333
5	7	1	-1	2.333	-1.333	-1.333
6	1	3	1	2.333	-1.333	-1.333
7	5	1	1	0.333	0.667	0.667
8	13	6	1	0	1	1.667
9	9	4	1	0	1	1
10	12	7	1	0	1	1

C	5
Fonc.Obj	30.1111

Primal form

- Minimization of the objective function w.r.t. β and ξ
- C is a parameter, we set C = 5

The value of **C** is an important issue in practice



- $y_i(x_i^T\beta+\beta_0)=1 - \xi_i$: saturated constraint ➔ support vector (yellow background) i.e. if we remove the case, the solution would be different (8 instances here)
- ξ_i = 0: the case is the right side of the margin
- $\xi_i \geq 1$: the case is in the wrong side of the maximum margin hyperplane (2 misclassified instances here)
- $0 < \xi_i < 1$: the instance is in the right side of the maximum margin hyperplane, but it exceeds its margin

Kernel trick

NONLINEAR CLASSIFICATION

Transformed feature space

Feature construction

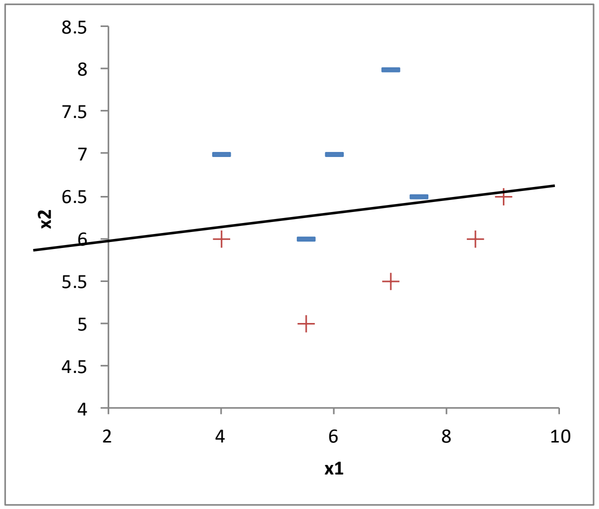
By performing appropriate transformations of variables, we can make linearly separable a problem which was not linearly separable in the original representation space.

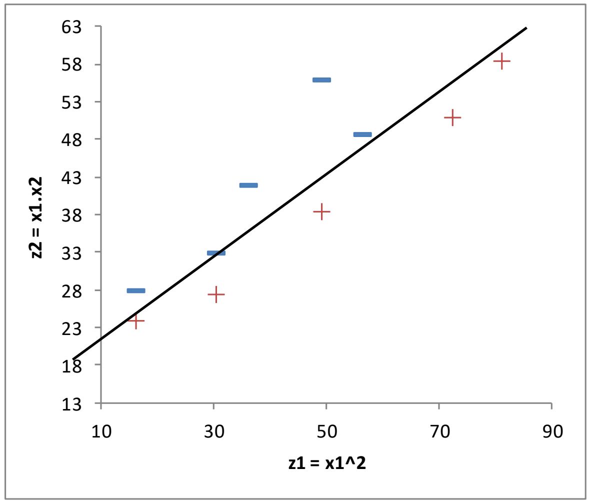
n°	x1	x2	y
1	4	7	-1
2	7	8	-1
3	5.5	6	-1
4	6	7	-1
5	7.5	6.5	-1
6	5.5	5	1
7	4	6	1
8	7	5.5	1
9	8.5	6	1
10	9	6.5	1

n°	z1	z2	y
1	16	28	-1
2	49	56	-1
3	30.25	33	-1
4	36	42	-1
5	56.25	48.75	-1
6	30.25	27.5	1
7	16	24	1
8	49	38.5	1
9	72.25	51	1
10	81	58.5	1

$$z_1 = x_1^2$$

$$z_2 = x_1 x_2$$





But multiplying concretely intermediate variables in the database is expensive, without any assurance that we obtain an efficient transformation.

Kernel functions

Applied to scalar product

The dot product between vectors has an important place in the calculations (dual form). SVM can take advantage of the "kernel" functions.

Let a transformation function $\phi(x)$ of initial variables

$\Rightarrow$ With the dual form, to optimize the Lagrangian, we calculate the scalar product $<\phi(x_i), \phi(x_{i'})>$ for each pair of instances (i, i')

Ex. $x = (x_1, x_2) \rightarrow \phi(x) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$

We should handle 3 variables instead of 2, the calculations are more expensive, not to mention the storage of additional variables.

We can find a function K(.), called Kernel Function, such as

$$K(x_i, x_{i'}) = \langle \phi(x_i), \phi(x_{i'}) \rangle$$

$\Rightarrow$ The main consequence is that we simply calculate the scalar product $<x_i, x_{i'}>$, and we transform the result with the Kernel function.

We handle only the 2 initial variables for calculations. But the algorithm fits the classifier in a 3 dimensional space!

Polynomial kernel

Examples

Dot product between two instances (vectors) *u* and *v* with the following values

$$u = (4,7) \quad v = (2,5) \Rightarrow \langle u, v \rangle = 4 \times 2 + 7 \times 5 = 43$$

Transformation (1)

$$\phi(x) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)$$

$$\left.\begin{aligned} \phi(u) &= (16,39.6,49) \\ \phi(v) &= (4,14.1,25) \end{aligned}\right\} \Rightarrow \langle \phi(u), \phi(v) \rangle = 1849$$

Corresponding function (1)

$$K_1(u,v) = \left(\langle u, v \rangle\right)^2 = 43^2 = 1849$$

The results are equivalent. With K (.), we work in a higher dimensional space without having to explicitly create the variables.

Transformation (2)

$$\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x2)$$

$$\left.\begin{aligned} \phi(u) &= (1,5.7,9.9,16,49,39.6) \\ \phi(v) &= (1,2.8,7.1,4,25,14.1) \end{aligned}\right\} \Rightarrow \langle \phi(u), \phi(v) \rangle = 1936$$

Corresponding function (2)

$$K_2(u,v) = \left(1 + \langle u, v \rangle\right)^2 = (1+43)^2 = 1936$$

We work in a 5-dimensional space in this configuration.

Dual form

Including the kernel function K()

Dual form –

Soft margin

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} K(x_i, x_{i'})$$

$s.c.$

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

$$0 \le \alpha_i \le C, \ \forall i$$

It is no longer possible to obtain an explicit classification function, we must use the support vectors in order to assign a class to unseen instances i.e. we must store them (values and weights) for the deployment (see PMML)

$$f(x) = \sum_{i' \in S} \alpha_{i'} y_{i'} K(x_{i'}, x) + \beta_0$$

β_0 can be obtained from the Karush-Kuhn-Tucker (KKT) conditions, but by using the kernel functions (see page 12 and following)

Some kernel functions

The most popular functions in tools (e.g. Scikit-learn package for Python - SVC)

Setting the right value of the parameters is the key issue, including the ''cost parameter'' **C**.

Polynomial	$K(u,v) = \left(\text{coef0} + \langle u,v \rangle\right)^{\text{degree}}$	coef0 = 0 and degree = 1, we have the "linear" kernel
Gaussian radial basis function (RBF)	$K(u,v) = \exp\left(-\gamma \times \lVert u - v \rVert^2\right)$	if it is not specified, the tools set by default (p: number of variables) $\gamma = \frac{1}{p}$
Hyperbolic tangent	$K(u,v) = \tanh\left(\gamma \times \langle u,v \rangle + \text{coef0}\right)$	

➡ There is a bit of polysemy in the parameters, but they have been popularized by the famous LIBSVM package, included in several data mining tools (Scikit-Learn - Python, e1071 - R, Tanagra, etc.)

Scores and probabilities

CLASS MEMBERSHIP PROBABILITIES

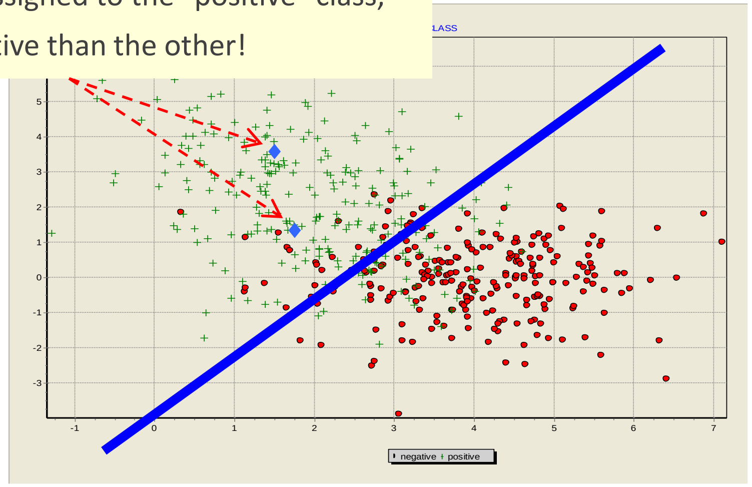
Class membership probability

Output of SVM = scores, but they are not calibrated

The output of the classification function f(x) enables to assign a class to an instance

$$f(x)\begin{cases} \geq 0 \Rightarrow \hat{y} = 1 \\ < 0 \Rightarrow \hat{y} = -1 \end{cases}$$

The two points are assigned to the "positive" class, but one is more positive than the other!



We need an indication about the credibility of the response.

|f(x)| is already a good indication. It allows to rank individuals according to their level of "positivity" (e.g. scoring, targeting, etc.)

But…

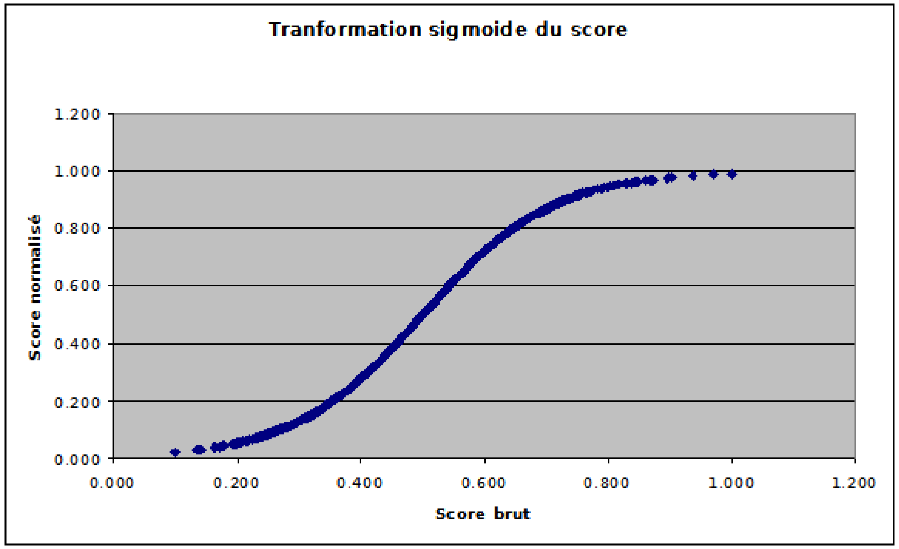
In many areas, we need an estimation of the class membership probability (e.g. interpretation, combination with a cost matrix, comparison with the outputs of other methods, etc.)

Platt scaling

Maximum likelihood estimation

We use a sigmoid function in order to map f (x) in the interval [0, 1]

$$P(Y=1/x)=\frac{1}{1+\exp[-f(x)]}$$



We can develop a more sophisticated solution by using a parameterized expression and estimate the values of the coefficients by maximum likelihood estimation

$$P(Y=1/x)=\frac{1}{1+\exp[-(a\times f(x)+b)]}$$

$\Rightarrow$ A logistic regression program can estimate easily the values of “a” and “b”

Platt scaling

An example

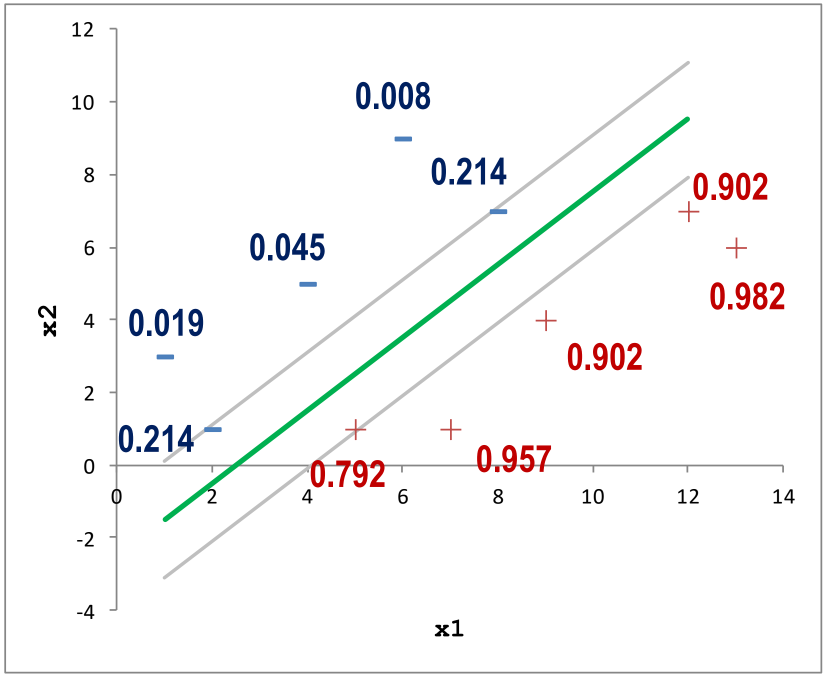
Let us see our first toy example (Page 9)

$$P(Y=1/x)=\frac{1}{1+\exp[-(1.32\times f(x)+0.02)]}$$

	beta.1	beta.2	beta.0		
	0.667	-0.667	-1.667		
n°	x1	x2	y	f(x)	P(y=1/x)
1	1	3	-1	-3.000	0.019
2	2	1	-1	-1.000	0.214
3	4	5	-1	-2.333	0.045
4	6	9	-1	-3.667	0.008
5	8	7	-1	-1.000	0.214
6	5	1	1	1.000	0.792
7	7	1	1	2.333	0.957
8	9	4	1	1.667	0.902
9	12	7	1	1.667	0.902
10	13	6	1	3.000	0.982

a	1.32
b	0.02

Classes membership probabilities are consistent with the position of the point and its distance from the frontier (maximum margin hyperplane).



Detecting the relevant variables

FEATURE SELECTION

Not embedded approaches

Filter and wrapper

Approaches which do not use explicitly the properties of the learning algorithm

Filter methods

The selection is done before and independently of the subsequent learning algorithm. Often based on the concept of "correlation" in the broad sense.

Pros: quickness, generic.

Cons: Not connected to the characteristics of the subsequent learning method, nothing says that the selected variables will be the right ones.

Wrapper methods

Use the classifier as a black box. Searching (e.g. forward, backward) of the best subset of variables which maximizes a performance criterion (e.g. cross-validation error rate).

Pros: selection directly related to a performance criterion.

Cons : very computationally intensive, risk of overfitting, do not use the internal characteristics of the learning algorithm (e.g. maximum margin for SVM).

Embedded approach
Maximum margin criterion

2 main issues: measuring the contribution of the variable in the margin, organizing an algorithm from this criterion [ABE, page 192]

Measuring the contribution of the variable "x_j" in the classifier, without having to explicitly launching the learning process without "x_j"

The variable x_j is disabled by setting its values at zero

$$\Delta^{(j)}\|\beta\|^2 = \sum_{i,i' \in S} \alpha_i \alpha_{i'} y_i y_{i'} \left(K(x_i, x_{i'}) - K(x_i^{(j)}, x_{i'}^{(j)}) \right)$$

➔ For a linear kernel, it is equivalent to test the significance of the coefficient β_j (is it significantly different to zero)

Backward searching process to detect the best subset of relevant variables

1. Compute δ_0, the initial margin with all the features
2. Find j* such as $\Delta^{(j^*)}||\beta||^2$ is minimum, and put it aside
3. Launch the learning without x_{j^*}, calculate the new margin δ
4. If $\frac{\delta_0 - \delta}{\delta} < \varepsilon$ Then remove xj*, set $\delta_0 = \delta$ and go to 2, Else STOP the searching process.

2 important notes:

- The removing of a variable always reduce the margin. The question is: how significant is the reduction? If it is significant, we cannot remove the variable.
- ε is a parameter of the algorithm (ε high, we obtain less variable)

Extension to multiclass problems (K number of classes, K > 2)

MULTICLASS SVM

Multiclass SVM

"One-against-rest" approach

The SVM approach is formally defined for binary problems Y $\in$ {+, -} , how to extend it (simply) to K classes problems Y$\in$\{y_1,…,y_K\} ? The most popular approaches reduce the multiclass problem into multiple binary classification problems.

- K binary classifiers which distinguish one of the classes y_k from the rest (one vs. rest or one vs. all) (Y'$\in$\{y_k=+1,$y_{(k)}$=-1\}).
- We obtain K classification functions $f_k(x)$
- For the prediction, we pick the class which has the highest score i.e. $\hat{y} = \arg\max_k f_k(x)$

This strategy is consistent with the maximum a posteriori (MAP) scheme

- **Properties**: K learning processes to perform on the dataset
- **Pros**: simplicity
- **Cons**: we can artificially insert an imbalance of the classes in the construction of individual models. If the scores are not-well calibrated, comparisons of the output of the classification functions are biased.

Multiclass SVM

One vs. one (pairwise) approach

- Building K(K-1)/2 classifiers which distinguish every pair of classes ($Y' \in \{y_k=+1, y_j=-1\}$). We obtain K(K-1)/2 classification functions $f_{k,j}(x)$
- In prediction, we predict the class using a voting system i.e. the one which has the maximum number of wins

$D_k(x)$ provides "#votes" for the class y_k

Knowing that $f_{j,k}(x) = -f_{k,j}(x)$

$$D_k(x) = \sum_{j \neq k, j=1}^{K} \text{sign}\left[f_{k,j}(x)\right]$$

We assign the class that has #vote max.

$$\hat{y} = \arg\max_k D_k(x)$$

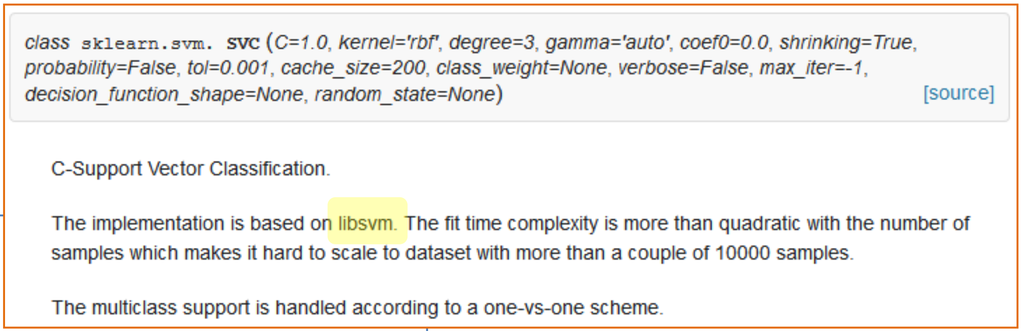
- **Note**: When two classes or more have the same number of votes, we use the sum of the scores $f_{k,j}(x)$ and we select the one corresponds to the max
- **Properties**: the approach is computationally intensive, but each classifier is learned on the subset of the whole dataset
- **Pros**: less problem of imbalance of the classes, the scores are better calibrated
- **Cons**: computing time when K increases (e.g. K = 10 ==> 45 classifiers to build)

Tools, packages, settings (Python, R and Tanagra)

SVM IN PRACTICE

Python

scikit-learn – SVC



class sklearn.svm. **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.

The multiclass support is handled according to a one-vs-one scheme.

```
#importing the training set
import pandas
dtrain = pandas.read_table("ionosphere-train.txt",sep="\t",header=0,decimal=".")
print(dtrain.shape)
y_app = dtrain.as_matrix()[:,32]
X_app = dtrain.as_matrix()[:,0:32]
#importing the module
from sklearn.svm import SVC
svm = SVC() #instanciation de l'objet
#displaying the settings (default kernel "rbf")
#the variables are not standardized (scale)
print(svm)
#learning process
svm.fit(X_app,y_app)
#importing the test set
dtest = pandas.read_table("ionosphere-test.txt",sep="\t",header=0,decimal=".")
print(dtest.shape)
y_test = dtest.as_matrix()[:,32]
X_test = dtest.as_matrix()[:,0:32]
#prediction on the test set
y_pred = svm.predict(X_test)
#measuring the test error rate: 0.07
from sklearn import metrics
err = 1.0 - metrics.accuracy_score(y_test,y_pred)
print(err)
```

Python

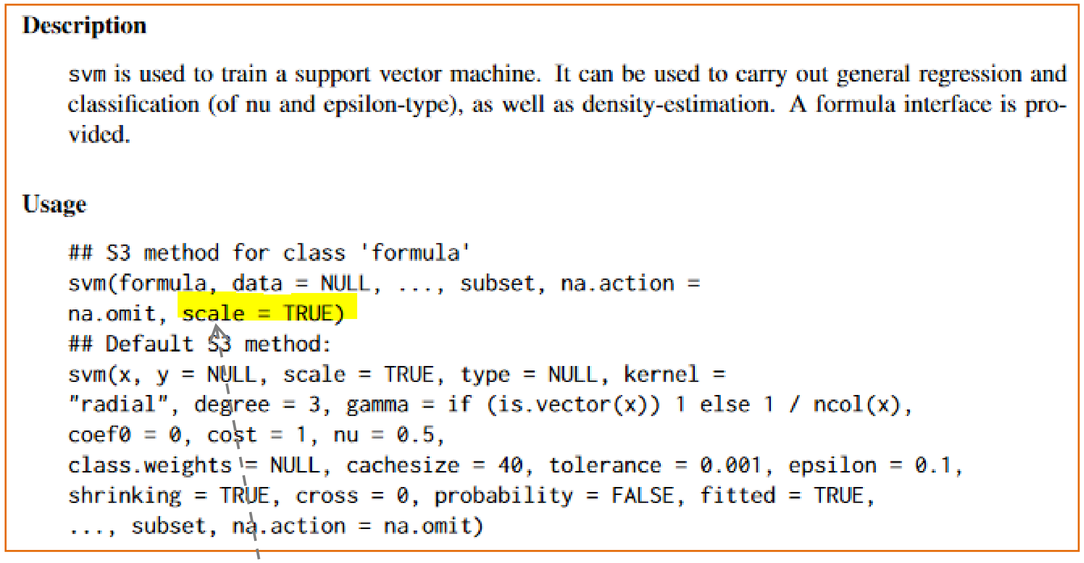
scikit-learn - GridSearchCV

Scikit-learn provides a mechanism for searching the optimal parameters in a cross-validation process. The test sample is not used here, thus we can use it in order to estimate the generalization error rate

```
#grid search tool
from sklearn.grid_search import GridSearchCV

#parameters to evaluate – modifying the kernel and the ‘cost parameter’
parametres = {"kernel":['linear','poly','rbf','sigmoid'],"C":[0.1,0.5,1.0,2.0,10.0]}

#the classifier to use
svmc = SVC()

#creating the object
grille = GridSearchCV(estimator=svmc,param_grid=parametres,scoring="accuracy")

#launching the exploring
resultats = grille.fit(X_app,y_app)

#best settings: {‘kernel’ : ‘rbf’, ‘C’ : 10.0}
print(resultats.best_params_)

#prediction with the best classifier
ypredc = resultats.predict(X_test)

#error rate on the test set = 0.045 (!)
err_best = 1.0 - metrics.accuracy_score(y_test,ypredc)
print(err_best)
```

R

e1071 – svm() from LIBSVM



Description

svm is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

```
#importing the learning set
dtrain <- read.table("ionosphere-train.txt",header=T,sep="\t")
dtest <- read.table("ionosphere-test.txt",header=T,sep="\t")

#package "e1071"
library(e1071)

#learning process
#the variables are automatically scaled
m1 <- svm(class ~ ., data = dtrain)

#displaying
print(m1)

#prediction
y1 <- predict(m1,newdata=dtest)

#confusion matrix and error rate = 0.04
mc1 <- table(dtest$class,y1)
err1 <- 1 - sum(diag(mc1))/sum(mc1)
print(err1)
```

```
Call:
svm(formula = class ~ ., data = dtrain)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.03125

Number of Support Vectors:  77
```

Compared with scikit-learn, the variables are automatically standardized. This is preferable in most cases.

R
e1071 – tune()

e1071 provides also, like scikit-learn, a tool for searching the "optimal" settings

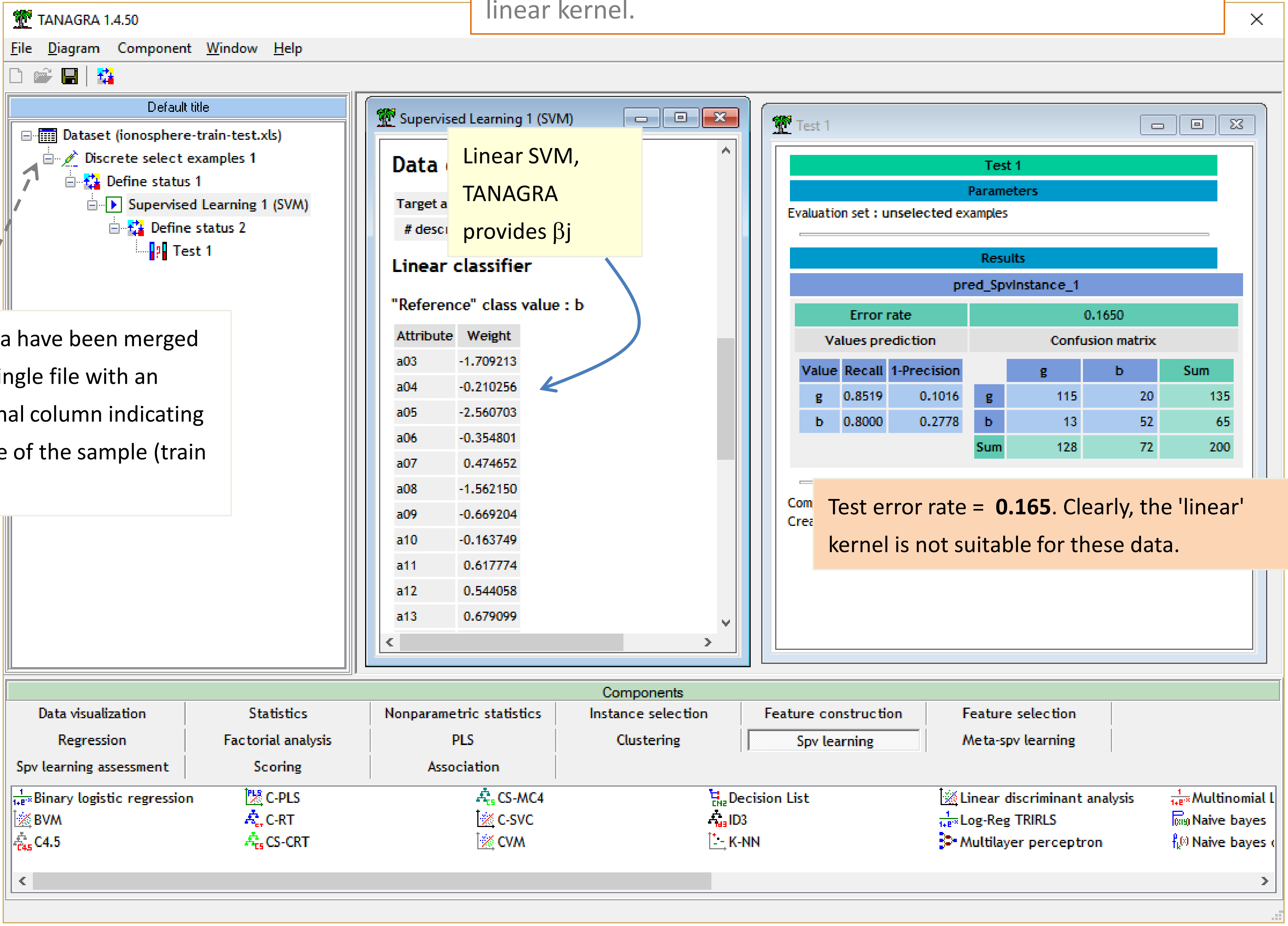
```
#grid search using cross-validation
set.seed(1000) #to obtain the same results for each session
obj <- tune(svm, class ~ ., data = dtrain, ranges =
           list(kernel=c('linear','polynomial','radial', 'sigmoid'), cost =
           c(0.1,0.5,1.0,2.0,10)), tunecontrol = tune.control(sampling="cross"))

#displaying
print(obj)

#build the classifier with the new parameters
m2 <- svm(class ~ ., data = dtrain, kernel='radial', cost = 2)

#displaying
print(m2)

#prediction
y2 <- predict(m2,newdata=dtest)

#confusion matrix - test error rate = 0.035
mc2 <- table(dtest$class,y2)
err2 <- 1 - sum(diag(mc2))/sum(mc2)
print(err2)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 kernel cost
 radial    2

- best performance: 0.06666667
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  2
      gamma:  0.03125

Number of Support Vectors:  62
```

Tanagra
SVM

The SVM component provides an explicit model when using a linear kernel.



TANAGRA 1.4.50

File Diagram Component Window Help

Default title

- Dataset (ionosphere-train-test.xls)
  - Discrete select examples 1
    - Define status 1
      - Supervised Learning 1 (SVM)
        - Define status 2
          - Test 1

The data have been merged into a single file with an additional column indicating the type of the sample (train or test)

Supervised Learning 1 (SVM)

Data

Target a

desc

Linear SVM, TANAGRA provides βj

Linear classifier

"Reference" class value : b

Attribute	Weight
a03	-1.709213
a04	-0.210256
a05	-2.560703
a06	-0.354801
a07	0.474652
a08	-1.562150
a09	-0.669204
a10	-0.163749
a11	0.617774
a12	0.544058
a13	0.679099

Test 1

Test 1

Parameters

Evaluation set : unselected examples

Results

pred_SpvInstance_1

Error rate	0.1650

Values prediction

Value	Recall	1-Precision
g	0.8519	0.1016
b	0.8000	0.2778

Confusion matrix

	g	b	Sum
g	115	20	135
b	13	52	65
Sum	128	72	200

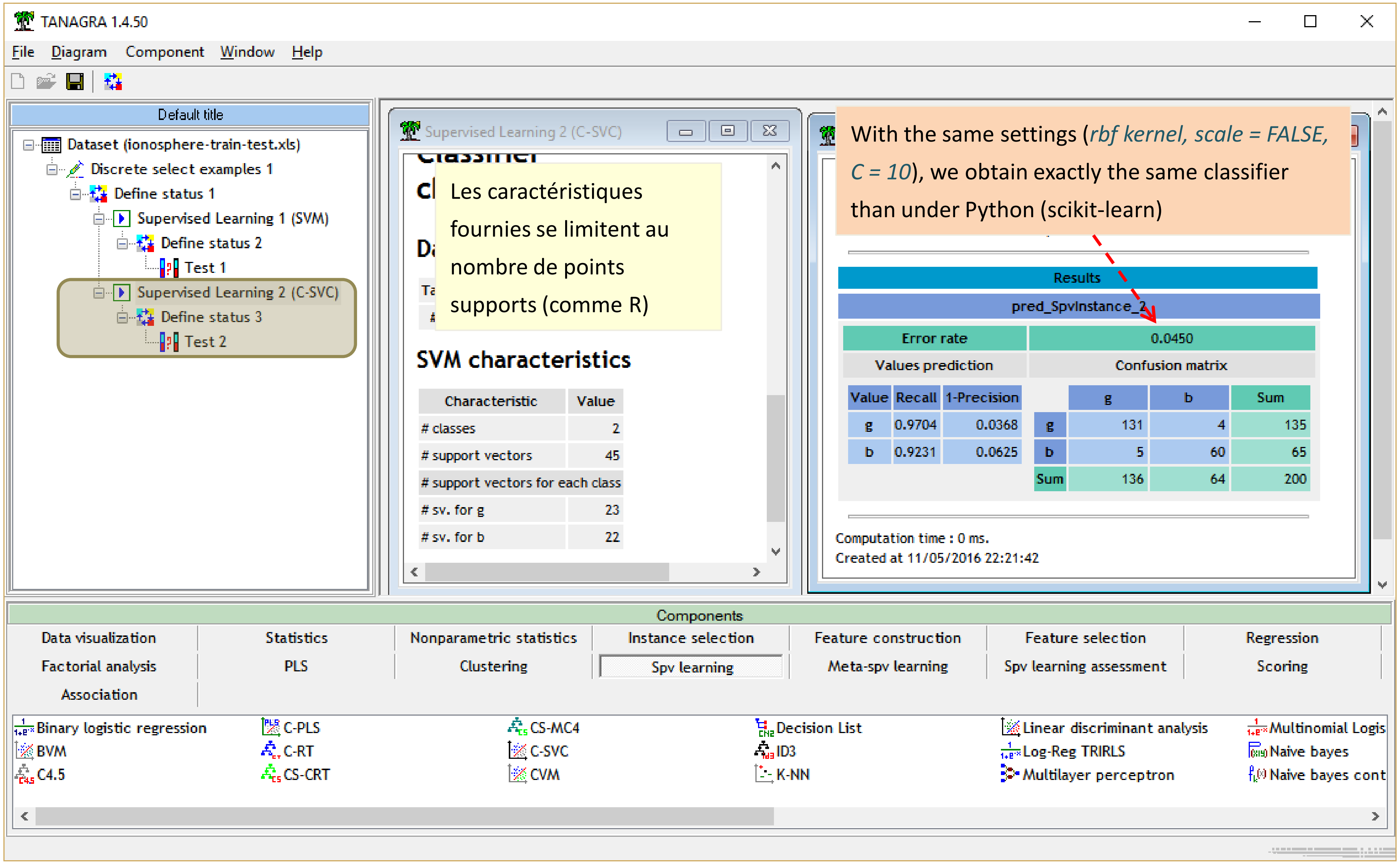
Com

Crea

Test error rate = **0.165**. Clearly, the 'linear' kernel is not suitable for these data.

Tanagra
C-SVC

C-SVC comes from the famous LIBSVM library



TANAGRA 1.4.50

Les caractéristiques fournies se limitent au nombre de points supports (comme R)

With the same settings (*rbf kernel, scale = FALSE, C = 10*), we obtain exactly the same classifier than under Python (scikit-learn)

SVM characteristics

Characteristic	Value
# classes	2
# support vectors	45
# support vectors for each class	
# sv. for g	23
# sv. for b	22

Results

pred_SpvInstance_2

Error rate	0.0450

Values prediction

Value	Recall	1-Precision
g	0.9704	0.0368
b	0.9231	0.0625

Confusion matrix

	g	b	Sum
g	131	4	135
b	5	60	65
Sum	136	64	200

Computation time : 0 ms.
Created at 11/05/2016 22:21:42

Pros and cons of SVM

OVERVIEW

SVM – Pros and cons

Pros

- Ability to handle high dimensional dataset (high #variables)
- Robust even if the ratio ‘’#observations / #variables’’ is inverted
- Efficient processing of the nonlinear problems with the kernel mechanism
- Nonparametric
- Robust against the outliers (controlled with the parameter C)
- #support vector provides a good indication about the complexity of the problem to handle
- Often effective compared with other approaches
- The possibilities of parameters adjustments allow flexibility (e.g. linear vs. nonlinear, regularization, etc.)

Cons

- Identifying the optimal values of the parameters is not obvious (SVM may be highly sensitive to the parameters)
- Difficulty in processing large dataset (#instances)
- Problems when we deal with noisy data labels (proliferation of the number of support vectors)
- No explicit model when we use nonlinear kernel
- Interpretation of the classifier for nonlinear kernel. Difficulty in the identification of the influence of the descriptors.
- The handling of the multiclass problem remains an open issue

SVM - Extensions

Popularized in the classification task, SVMs can be applied to other kind of problems:

- semi-supervised learning (partially labeled data)
- support vector regression (regression problem)
- support vector clustering (clustering problem)

Specific kernel functions are developed (text mining, image mining, speech recognition,...). they must be adapted to the notion of similarity between observations in the domain.

The SVM approach is very close to research, with often new developments and improvements. All these new variants are not always available in the usual tools.

Bibliography, tutorials

REFERENCES

References

[ABE] Abe S., « Support Vector Machines for Pattern Classification », Springer, 2010 ; the whole book, and especially the chapters 2 and 3.

[BLU] Biernat E., Lutz M., « Data Science : fondamentaux et études de cas », Eyrolles, 2015 ; chapter 13.

[BIS] Bishop C.M., « Pattern Recognition and Machine Learning », Springer, 2006 ; chapter 7.

[CST] Cristianini N., Shawe-Taylor J., « Support Vector Machines and other kernel-based learning methods », Cambridge University Press, 2000.

[HTF] Hastie T., Tibshirani R., Friedman J., « The elements of Statistical Learning - Data Mining, Inference and Prediction », Springer, 2009 ; chapters 4 et 12.

… and many course materials found on the web…

Tutorials

Chang C.-C., Lin C.J., « LIBSVM: a library for support vector machines », in ACM Transactions on Intelligent Systems and Technology, 2(27), p. 1-27, 2011. The LIBSVM library is available in various data mining software.

Tanagra Tutorial, « Implementing SVM on large dataset », July 2009; comparison of various tools (Tanagra, Orange, RapidMiner, Weka) on a high dimensional dataset (31809 descriptors).

Tanagra Tutorial, « SVM using the LIBSVM library », November 2008; using the LIBSVM library from Tanagra.

Tanagra Tutorial, « CVM and BVM from the LIBCVM library », July 2012; extension of LIBSVM, this library can handle large dataset (high number of instances).

Tanagra Tutorial, « Support Vector Regression », April 2009 ; SVM in the regression context under Tanagra and R (e1071).