Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES MATHÉMATIQUES

Etude et implémentation des machines à vecteurs de support

Baetmans, Florian

Award date:
2010

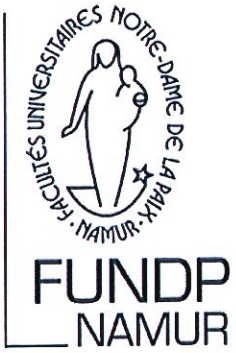
Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. May. 2026



FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX
NAMUR

Faculté des Sciences

ETUDE ET IMPLEMENTATION DES MACHINES A VECTEURS DE SUPPORT

Promoteurs : Annick SARTENAER
Caroline SAINVITU

Mémoire présenté pour l'obtention
du grade académique de master en Sciences Mathématique

Florian BAETMANS

Août 2010

Remerciements

En préambule à ce mémoire, je tiens à remercier les personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce travail.

En premier lieu, je désire remercier Madame le Professeur Annick Sartenaer pour sa disponibilité et ses conseils très judicieux lors de l'encadrement de ce mémoire et spécialement pour la relative liberté qu'elle m'a autorisée concernant le contenu de ce mémoire.

Je tiens à remercier vivement Madame le Docteur Caroline Sainvitu pour son encadrement continu pendant la réalisation de ce mémoire, sa patience et ses encouragements. Elle a toujours porté un regard constructif sur le travail accompli et m'a donné de très bons conseils. Je tiens aussi à la remercier pour m'avoir accueilli au sein de l'entreprise Cenaero et pour la confiance qu'elle m'a témoignée.

Je tiens bien évidemment à remercier les dirigeants de Cenaero de m'avoir ouvert les portes de leur entreprise pendant une année pour la réalisation de mon mémoire.

Je remercie Messieurs les Professeurs Philippe Toint et Jean-Jacques Strodiot d'avoir accepté de prendre part au jury de ce mémoire.

Je remercie également tous les membres du département de Mathématique pour leur aide et leurs conseils.

Enfin, je remercie mes parents, mes amis et Claire Linart pour leur contribution, leur soutien, leur patience ainsi que leurs nombreuses relectures qui ont permis d'améliorer ce mémoire.

Résumé

Ce mémoire résulte d'une demande formulée par l'entreprise Cenaero. Il a pour objectif l'étude théorique des méthodes de classification et, en particulier, celles de la régression par les machines à vecteurs de support (SVM) afin d'élargir l'éventail des modèles disponibles dans le logiciel Minamo. Ces méthodes sont issues de la théorie d'apprentissage statistique proposée par V. Vapnik en 1995.

L'approximation de fonctions inconnues est une des utilisations les plus fréquentes des machines à vecteurs de support. Le contexte du problème est de déterminer une fonction inconnue entre un ensemble de données d'entrée et de sortie. L'idée essentielle des SVM consiste à transformer, à l'aide de noyaux, les données d'apprentissage de l'espace d'entrée dans un espace de plus grande dimension, appelé espace de redescription. Nous étudions la technique de construction d'un hyperplan permettant, dans le cas de la régression, de construire un modèle qui minimise l'erreur entre les sorties du modèle et les données de sortie.

Les modèles obtenus par les machines à vecteurs de support pour la régression peuvent être utilisés afin d'accélérer des procédures d'optimisation. En effet, les simulations étant généralement très coûteuses en temps de calcul, l'utilisation de modèles permet de diminuer le nombre d'appels à celles-ci au cours du processus d'optimisation.

Dans le cadre du partenariat avec l'entreprise Cenaero, nous avons analysé et implémenté l'algorithme SMO permettant la construction de modèles SVM pour la régression.

Des résultats numériques accompagnent la méthode des machines à vecteurs de support pour la régression proposée et indiquent qu'elle est aussi compétitive que les autres méthodes RBF classiques. De plus, le modèle SVR a pu être testé et validé après comparaison des différents modèles pour un cas test industriel sur un ensemble de données d'apprentissage réelles.

Abstract

This master's thesis results from a request of Cenaero. This work is concerned with the theoretical study of classification and regression methods by Support Vector Machines in order to expand the range of models available in Minamo. The SVM's methods are new techniques of the statistical learning theory proposed by Vapnick in 1995 and developed from the Structural Risk Minimization theory.

The approximation of unknown functions is one of the most frequent use of SVM. The context of this problem is to determine an unknown function relating a set of input and output vectors. The inductive principle of SVM consists in mapping, using kernels, the input vectors into a high-dimensional feature space. In this master's thesis, we study the technique of constructing a hyperplane allowing, in the case of regression, to construct a model wich minimizes the error between the model's output vectors and the true output vectors.

The models obtained via the support vector machines for regression can be used to speed up optimization procedures. Indeed, simulations are generally expensive in calculation time and the use of models reduces the number of exacts simulations during the optimization process.

Within the framework of the partnership agreement with Cenaero, we have analyzed and implemented the SMO algorithm wich allows to construct SVM models for regression.

Numerical results follow the proposed SVM method for regression and indicate that they are as competitive as other RBF classical methods. Moreover, the SVM's model has been tested and validated after comparison of different models for an industrial test case.

Principaux symboles utilisés

n	le nombre de données de l'ensemble d'apprentissage
m	le nombre de classes pour le problème de la classification
p	la dimension des données de l'ensemble d'apprentissage
$\mathcal{S}$	l'ensemble d'apprentissage
$\mathcal{X}$	l'espace d'entrée ($\mathbb{R}^p$)
$\mathcal{Y}$	l'espace de sortie ($\{-1,+1\}$ pour la classification et $\mathbb{R}$ pour la régression)
$\mathcal{R}$	l'espace de redescription
x_i	un vecteur de l'espace d'entrée
y_i	un vecteur de l'espace de sortie
$[x]_i$	la $i^{\text{ème}}$ composante du vecteur x
$(x)_+$	la partie positive de x
$f(x)$	la fonction transformant un vecteur de l'espace d'entrée en un vecteur de l'espace de sortie (classificateur ou fonction de régression)
$h(x)$	la fonction de décision d'un classificateur
$\mathcal{F}$	une classe de fonctions
$sgn(expression)$	la fonction renvoyant le signe d'une expression $(+1,-1)$
$d(w,b;x_i)$	la distance euclidienne d'une donnée x_i à l'hyperplan déterminé par w et b
$\rho(w,b)$	la marge d'un hyperplan déterminé par w et b
w	le vecteur normal à l'hyperplan $\langle w,x\rangle + b = 0$
b	le terme constant dans l'équation de l'hyperplan
C	la constante de compromis dans la formulation des SVM et SVR à marge souple
ε	la déviation dans la formulation des SVR à marge souple
α_i	une variable duale du problème SVM
$\Phi(x_i)$	la transformation d'un vecteur x_i de l'espace d'entrée vers l'espace de redescription
$\langle x_i,x_j\rangle$	le produit scalaire entre les vecteurs x_i et x_j
$k(x_i,x_j)$	la fonction noyau appliquée aux vecteurs x_i et x_j
K	la matrice de Gram : $K_{ij} = k(x_i,x_j)$
H	la matrice hessienne : $H_{ij} = y_i y_j k(x_i,x_j)$
L	le Lagrangien
ξ_i	les variables d'écart dans la formulation des SVM à marge souple ($i = 1,\ldots,n$)
$p(x,y)$	la distribution de probabilités générant les données

$E(X)$	l'espérance de la variable aléatoire X
$l(f(x), y)$	la fonction coût
$R[f]$	le risque fonctionnel
$R_{\text{emp}}[f] = E_S[f]$	le risque empirique
$R_{\text{estimate}}[f]$	l'erreur moyenne
R	le coefficient de corrélation
h	la dimension Vapnik-Chervonenkis
σ	l'écart type d'une gausienne ou paramètre du noyau RBF Gaussien

Principaux acronymes utilisés

AG	Algorithme génétique
CAE	Computer Aided Engineering - Conception assistée par ordinateur
CPU	Central Processing Unit - Unité centrale de traitement
CVT	Centroïdal Voronoi Tessellation - Tesselation de Voronoi Centroidale
DoE	Design of Experiments - Plan d'expériences
HPC	High Performance Computing - Calcul haute performance
IPM	Interior Point Method - Méthode de points intérieurs
KKT	conditions de Karush Kuhn Tucker
LCVT	Latinized Centroïdal Voronoi Tessellation - Tesselation de Voronoi Centroidale latinisée
LHS	Latin Hypercube Sampling - Echantillonnage hypercube latin
LOO	Leave-One-Out
LP	Linear Problem - Problème linéaire
LPT	Low Pression Turbines - Turbines basse pression
ML	Machine Learning - Apprentissage automatique
MAE	Maximal Absolute Error - Erreur absolue maximum
MRE	Minimisation du Risque Empirique
MRS	Minimisation du Risque Structurel
MSE	Mean Square Error - Erreur aux moindres carrés
OO	Orienté Objet
QP	Quadratic Problem - Problème quadratique
RBF	Radial Basis Function - Fonction à base radiale
RMSE	Root Mean Squared Error - Erreur de validation
SMO	Sequential Minimal Optimization - Optimisation minimale séquentielle
SVM	Support Vector Machines - Machines à vecteurs de support
SVR	Support Vector Regression - Machines à vecteurs de support pour la régression
VC	dimension de Vapnik-Chervonenkis
VS	Vecteurs de Support

Principales abréviations utilisées

al.	les autres personnes
cf.	confer
e.g.	par exemple
i.e.	c'est-à-dire
i.i.d.	indépendantes et identiquement distribuées
max	maximiser
min	minimiser
s.c.	sous contraintes
s.r.s	sans restriction de signe

Table des matières

1 L'entreprise Cenaero et le logiciel d'optimisation Minamo 16
- 1.1 Présentation de l'entreprise Cenaero . . . 16
- 1.2 Contexte et objectifs du mémoire . . . 18
  - 1.2.1 Description de Minamo . . . 18
  - 1.2.2 Méthodologie d'optimisation . . . 18
  - 1.2.3 Algorithme génétique . . . 21
  - 1.2.4 Plan d'expériences . . . 23
  - 1.2.5 Méta-modèles . . . 26
  - 1.2.6 Objectif du mémoire . . . 26

2 L'apprentissage automatique 28
- 2.1 Problème général . . . 28
  - 2.1.1 Fonction d'erreur et risque . . . 30
- 2.2 Les hyperparamètres . . . 31
- 2.3 Estimation du risque fonctionnel . . . 32
  - 2.3.1 Validation croisée . . . 33
- 2.4 La classification . . . 34
  - 2.4.1 La classification binaire . . . 35
  - 2.4.2 La classification linéaire . . . 36
  - 2.4.3 La classification non-linéaire . . . 36
  - 2.4.4 Sur- et sous-apprentissage . . . 37
  - 2.4.5 La classification multi-classes . . . 39
- 2.5 La régression . . . 39
  - 2.5.1 La régression linéaire . . . 40
  - 2.5.2 La régression non-linéaire . . . 40
    - 2.5.2.1 Les modèles paramétriques . . . 41
    - 2.5.2.2 Les modèles non-paramétriques . . . 41
  - 2.5.3 Sur-apprentissage et sous-apprentissage . . . 44

3 Machines à vecteurs de support pour la classification 45
- 3.1 Classificateur : notions de base . . . 45
  - 3.1.1 Classificateur linéaire . . . 46
  - 3.1.2 Marge de l'hyperplan . . . 47
  - 3.1.3 Hyperplans canoniques . . . 49
  - 3.1.4 Motivation : pourquoi maximiser la marge? . . . 51
- 3.2 Classificateur SVM linéaire . . . 53
- 3.3 Classificateur SVM non-linéaire . . . 57
- 3.4 Fonction noyau . . . 59

3.4.1 Condition de Mercer 62
3.4.2 Exemples de noyaux 64
3.4.2.1 Les noyaux standards 64
3.4.2.2 Le noyau RBF Gaussien 65
3.4.3 Composition de noyaux 66
3.5 Classificateur SVM : formulation générale 67
3.5.1 Cas non-linéairement séparable : marge fixe 67
3.5.2 Cas non-linéairement séparable : marge souple 67
3.6 Etude de l'unicité et de la globalité de la solution 69
3.7 Relation avec l'apprentissage statique 69
3.7.1 Risque empirique et risque fonctionnel 69
3.7.2 Consistance non triviale et convergence uniforme 70
3.7.3 Borne sur l'erreur d'apprentissage 71
3.7.4 Dimension VC 71
3.7.5 Bornes basées sur la marge 72
3.7.6 Minimisation du risque empirique et fonctionnel 73
3.7.7 Algorithmes d'apprentissage 74
3.7.8 En conclusion 75

4 Machines à vecteurs de support pour la régression 76
4.1 La régression par vecteurs de support (SVR) 76
4.1.1 Le cas linéaire 76
4.1.2 Le cas non-linéaire 78
4.1.2.1 Le problème quadratique 78
4.2 Relation avec l'apprentissage statique 84

5 Implémentation d'un algorithme SVR et l'algorithme SMO 86
5.1 Le fonctionnement général d'un algorithme SVR 86
5.2 Les différents algorithmes d'optimisation 88
5.2.1 L'algorithme de points intérieurs 88
5.2.2 L'algorithme du gradient 88
5.2.3 L'algorithme de sélection de sous-ensembles 88
5.2.3.1 Le chunking 89
5.2.3.2 L'algorithme SMO 89
5.2.4 Les différentes librairies 103
5.3 Procédure de sélection des hyperparamètres 103
5.4 Les différentes astuces et points techniques 105

6 Tests, résultats et interprétations 107
6.1 Méthodologie 107
6.1.1 Fonctions tests utilisées 107
6.1.2 Validation des modèles 112
6.1.2.1 Types d'erreurs mesurées 113
6.1.2.2 Les Tableaux de validation 114
6.2 Présentation des modèles RBF 114
6.3 Illustrations des concepts relatifs aux SVR 115
6.3.1 Sélection du modèle en fonction du tube d'insensibilité ε . 115
6.3.2 Relation entre la qualité de l'approximation et les vecteurs de support 116
6.3.3 Illustration de l'action des multiplicateurs de Lagrange . . 116

6.4 Résultats 117
6.4.1 Performance en cas de données bruitées 117
6.4.2 Etude des modèles obtenus pour chaque fonction test et de la capacité de généralisation 124
6.4.2.1 En conclusion 139
6.4.3 Etude des différents hyperparamètres de la méthode SVR 140
6.4.4 Etude du temps d'exécution de l'algorithme SMO 141
6.4.5 Test du modèle SVR sur la base de données LPT 142
6.4.6 Couplage avec l'optimisation 145
6.4.7 Avantages du modèle SVR par rapport au modèle de réseaux de neurones 149

A Codes sources **152**
A.1 SVMmain.cpp 158
A.2 SVMmain.h 172
A.3 SVM.cpp 175
A.4 SVM.h 196
A.5 kFoldCrossValidation.cpp 199
A.6 kFoldCrossValidation.h 207
A.7 TestSVM.cpp 210
A.8 onedim.py 221
A.9 twodim.py 223

Liste des figures

1.1 Méthodologie générale de l'optimiseur Minamo. 19
1.2 Illustration de l'optimisation assistée par méta-modèle à la base de Minamo. 20
1.3 Etapes d'un algorithme génétique. 21
1.4 Propriété LHS. 24
1.5 Exemple des trois techniques d'échantillonnage : LHS, CVT et CVT Latinisée. 25

2.1 Machine Learning supervisée. La fonction $f(x)$ est une estimation de la fonction inconnue $g(x)$ transformant x en y. 29
2.2 Classificateur linéaire séparant les points bleus et les points rouges par un hyperplan. 35
2.3 Transformation $\Phi : (x_1, x_2) \rightarrow (z_1, z_2, z_3)$ de l'espace d'entrée (gauche) dans l'espace de redescription (droite) permettant une séparation linéaire des données. 37
2.4 Classificateurs linéaire et non-linéaire construits sur un ensemble d'apprentissage clairsemé (cas 1). En fonction de la vraie distribution des données, le classificateur linéaire peut être en situation de sur-apprentissage (cas 2) ou le classificateur non-linéaire peut être en situation de sur-apprentissage (cas 3) 38
2.5 Régression linéaire à partir de données bruitées. 41
2.6 Régression non-linéaire réalisée à l'aide de modèles polynomiaux de différents degrés. 42
2.7 Représentation des fonctions à base radiale Gaussienne et multi-quadratique. 43
2.8 A gauche : approximation de la fonction $sinc(x)$ par un modèle polynomial de différents degrés à partir de données bruitées. Au milieu : le modèle polynomial correspondant au risque empirique minimum. A droite : le risque fonctionnel et empirique par rapport au degré du modèle polynomial. 44

3.1 Représentation graphique dans $\mathbb{R}^2$ des paramètres w et b de l'hyperplan correspondant à la fonction de décision d'un classificateur linéaire. 47
3.2 Représentation de la distance euclidienne d'une donnée x. 48
3.3 Représentation des hyperplans canoniques. 50
3.4 Différents classificateurs pour deux classes de données. 51
3.5 Classification d'une nouvelle donnée. 51

3.6 Classement de nouvelles données en fonction de l'hyperplan. . . . 52
3.7 Variables de relaxation. 58
3.8 Transformation non-linéaire Φ. 65
3.9 Transformation Φ des données x_i et x_j. 65
3.10 Similarité de toutes les données de l'ensemble d'apprentissage vue en utilisant un noyau RBF Gaussien avec un σ tendant vers 0. . 66
3.11 Consistance. 70
3.12 Exemple de configuration de 3 données séparables de toutes les manières possibles par les droites de $\mathbb{R}^2$. Les points bleus et rouges représentent respectivement les données assignées positivement et négativement. La flèche représente le côté de la droite où les données seront classées positivement. 71
3.13 Illustration de l'inégalité (3.17). La courbe croissante, appelée *confiance*, correspond à la borne supérieure du terme de capacité. 73
3.14 Minimisation du risque structurel. Les sous-ensembles sont ordonnés selon leur dimension VC. 74
3.15 Principe de minimisation du risque structurel. 75

4.1 La fonction de coût pour $e = y - f(x)$ et le tube d'insensibilité ε pour un modèle linéaire et non-linéaire f. 78
4.2 Les fonctions de coût quadratique et de Huber. 84

5.1 Organigramme d'un algorithme SVR. 87
5.2 Procédure de sélection des meilleurs hyperparamètres. 104

6.1 Représentation de la fonction test à une dimension. 108
6.2 Courbes de niveau et graphique de la fonction Ackley. 109
6.3 Courbes de niveau et graphique de la fonction Branin. 109
6.4 Courbes de niveau et graphique de la fonction Hosaki. 110
6.5 Courbes de niveau et graphique de la fonction Mystery. 111
6.6 Courbes de niveau et graphique de la fonction Rastrigin. 111
6.7 Courbes de niveau et graphique de la fonction Rosenbrock. . . . 112
6.8 Cas 1 : fonction originale $\sin(x)$, Cas 2 : approximation avec $\varepsilon = 0.1$, Cas 3 : approximation avec $\varepsilon = 0.2$ et Cas 4 : approximation avec $\varepsilon = 0.4$. Les courbes rouges indiquent les frontières supérieures et inférieures du tube d'insensibilité ε - source [11]. . 115
6.9 Vecteurs de support et approximations (courbe bleue) pour différentes valeurs de ε - source [11]. 116
6.10 Les multiplicateurs de Lagrange agissant comme des "forces" sur le modèle afin que celui-ci se situe à l'intérieur du tube d'insensibilité ε - source [11]. 117
6.11 Bruit gaussien pour différentes valeurs de la variance. 118
6.12 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.1$ et $C = 1$, pour 40 données bruitées et pour la fonction (6.1). 119
6.13 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 100$, pour 40 données bruitées et pour la fonction (6.1). . 119
6.14 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.1$, $\varepsilon = 0.001$ et $C = 1$, pour 40 données bruitées et pour la fonction (6.1). . . . 120
6.15 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.1$ et $C = 1$, pour 40 données bruitées et pour la fonction (6.4). 121

6.16 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 20$, pour 40 données bruitées et pour la fonction (6.4). . . 121
6.17 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 28.1$, pour 40 données bruitées et pour la fonction (6.4). . 122
6.18 Modèle obtenu avec 40 données non bruitées pour le jeu d'hyperparamètres $\sigma = 0.1$, $\varepsilon = 0.001$ et $C = 9.04$, pour 40 données et pour la fonction (6.1). 124
6.19 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 28.1$, pour 40 données et pour la fonction (6.2). 125
6.20 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.1$, $\varepsilon = 0.001$ et $C = 52.83$, pour 80 données et pour la fonction (6.2). 126
6.21 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 28.1$, pour 20 données et pour la fonction (6.4). 128
6.22 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 28.1$, pour 40 données et pour la fonction (6.4). 128
6.23 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 1$, pour 40 données et pour la fonction (6.3). 130
6.24 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 9.04$, pour 80 données et pour la fonction (6.3). 130
6.25 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.3$, $\varepsilon = 0.001$ et $C = 9.04$, pour 40 données et pour la fonction (6.5). 132
6.26 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.1$, $\varepsilon = 0.001$ et $C = 1$, pour 80 données et pour la fonction (6.5). 132
6.27 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 52.83$, pour 40 données et pour la fonction (6.6). 133
6.28 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 9.04$, pour 60 données et pour la fonction (6.6). 134
6.29 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.5$, $\varepsilon = 0.001$ et $C = 52.83$, pour 40 données et pour la fonction (6.7). 136
6.30 Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.3$, $\varepsilon = 0.001$ et $C = 9.04$, pour 80 données et pour la fonction (6.7). 137
6.31 Temps d'exécution de l'algorithme SMO pour la fonction Rosenbrock - Echelle logarithmique. 141
6.32 Résultat de la procédure LOO pour le modèle RBF par défaut pour le premier ensemble de données. 143
6.33 Résultat de la procédure LOO pour le modèle RBF "tuné" pour le premier ensemble de données. 143
6.34 Résultat de la procédure LOO pour le modèle SVR pour le premier ensemble de données. 143
6.35 Résultat de la procédure LOO pour le modèle RBF par défaut pour le deuxième ensemble de données. 144
6.36 Résultat de la procédure LOO pour le modèle RBF "tuné" pour le deuxième ensemble de données. 144
6.37 Résultat de la procédure LOO pour le modèle SVR pour le deuxième ensemble de données. 145
6.38 Historique de convergence pour la fonction Ackley pour l'algorithme génétique avec et sans modèle. 146
6.39 Historique de convergence pour la fonction Rastrigin. 147
6.40 Historique de convergence pour la fonction Ackley. 147
6.41 Historique de convergence pour la fonction Branin. 148

Introduction

Ce mémoire est issu d'une demande formulée par l'entreprise Cenaero. Le centre de recherche appliquée Cenaero est un centre d'excellence en technologies de simulations numériques multidisciplinaires. Parmi leurs nombreux domaines d'activités, il en est un qui se concentre sur le développement de méthodes d'optimisation basées sur des algorithmes génétiques fortement accélérés par des modèles approchés. Les modèles utilisés jusqu'à présent chez Cenaero sont de types réseaux de neurones artificiels et le Kriging.

L'objectif de ce mémoire est de fournir un nouveau modèle basé sur les machines à vecteurs de support pour la régression. Nous disposons pour cela d'un ensemble d'apprentissage de vecteurs d'entrée et de sortie. Une fonction inconnue relie les vecteurs d'entrée aux vecteurs de sortie. Le problème est alors de déterminer cette fonction inconnue, c'est-à-dire de construire un modèle approché qui minimise l'erreur entre les sorties du modèle et les vecteurs de sorties de l'ensemble d'apprentissage. De plus, le modèle doit présenter une bonne qualité de généralisation, en d'autres mots, il doit représenter correctement tout nouveau couple "entrée - sortie".

Cenaero dispose d'un logiciel d'optimisation permettant d'effectuer l'interpolation de fonctions, actuellement par modèle RBF ou Kriging. Les modèles construits par ce logiciel se basent sur quelques données d'apprentissage issues de simulations numériques exactes, comme par exemple des simulations aérodynamiques. Cependant, la simulation exacte est très coûteuse en temps de calcul. L'optimisation sur un modèle approché permet donc une réduction considérable du temps de calcul. Les modèles utilisés par Cenaero donnent de très bons résultats mais présentent néanmoins plusieurs inconvénients, notamment lorsque les données sont bruitées. Le but de ce mémoire est de proposer un nouveau modèle permettant la construction de modèles robustes.

En premier lieu, nous présentons les activités de Cenaero et le logiciel d'optimisation Minamo. Les contraintes fixées par Cenaero sont ensuite exposées. Ces dernières permettent de définir plus précisément le contexte et le but de ce mémoire. Nous introduisons ensuite, au Chapitre 2, les concepts fondamentaux de l'apprentissage automatique à partir de données ainsi que les problèmes bien connus de la classification et de la régression. Les Chapitres 3 et 4 se concentrent respectivement sur les méthodes des machines à vecteurs de support (SVM) pour la classification et la régression. Nous y introduisons de façon simple et complète, les différentes notions nécessaires à la compréhension du principe général de fonctionnement des SVM. Les méthodes sont tout d'abord étudiées dans le cas linéaire, puis elles sont étendues au cas non-linéaire grâce à des fonctions noyaux. Ces noyaux permettent de résoudre le problème non-linéaire par une transformation implicite des données dans un espace de dimension supérieure.

Au Chapitre 5, nous présentons la partie implémentation du mémoire. Nous étudions plus particulièrement l'algorithme Sequential Minimal Optimization (SMO) et nous traitons des différentes questions pratiques comme, par exemple, celle de la sélection des hyperparamètres. Finalement, pour terminer ce mémoire, nous présentons au Chapitre 6 les différents résultats obtenus sur différentes fonctions tests analytiques. De plus, un cas test industriel sur les turbines basse pression permet d'évaluer les performances du modèle SVM par rapport aux modèles déjà présents dans Minamo.

Chapitre 1

L'entreprise Cenaero et le logiciel d'optimisation Minamo

RÉSUMÉ. *Dans ce chapitre, nous présentons les activités de Cenaero et plus précisément celles du groupe de Méthodes Numériques et Optimisation avec lequel j'ai travaillé en partenariat pour réaliser ce mémoire. Nous décrivons le logiciel d'optimisation Minamo et faisons ensuite le lien entre le sujet du mémoire, les machines à vecteurs de support et le logiciel Minamo. Nous illustrons et expliquons pour cela la méthodologie générale de la plate-forme d'optimisation Minamo, laquelle se base principalement sur des plans d'expériences, des méta-modèles et des algorithmes génétiques. Nous justifions également l'intérêt d'effectuer l'optimisation sur des méta-modèles plutôt que directement sur les données exactes et donnons des méthodes permettant de vérifier la fiabilité de ces méta-modèles. Finalement, les contraintes posées par Cenaero sont formulées. Ces dernières permettent de définir plus précisément le contexte et le but de ce mémoire. Ce chapitre se base principalement sur les références [58, 68].*

1.1 Présentation de l'entreprise Cenaero

Ce mémoire, qui concerne les machines à vecteurs de support, a été réalisé de février 2009 à août 2010 en partenariat avec l'entreprise **Cenaero**. Il a aussi été encadré par le département de Mathématique des FUNDP de Namur.

Le centre de recherche appliquée Cenaero est un centre d'excellence en technologies de simulations numériques fondé en 2002 par un consortium d'universités et d'entreprises actives principalement dans le domaine de l'aéronautique. Cenaero est implanté, avec de nombreuses autres entreprises, dans l'Aéropôle de Gosselies (Belgique). Il se concentre sur le développement de méthodes et de logiciels modélisant des problèmes industriels complexes. L'activité centrale de Cenaero est la simulation numérique avancée, un procédé permettant d'accélérer les cycles de conception en limitant le recours à l'expérimentation et de mieux

comprendre les mécanismes physiques complexes, dans le but de concevoir des produits optimisés. La mission principale de Cenaero est de venir en aide aux entreprises dans leurs efforts d'innovation dans un ou plusieurs des domaines suivants :

- la fabrication virtuelle (simulation des procédés d'assemblage tels que le soudage, l'usinage, etc.),
- la modélisation des matériaux et des structures (conception en matériaux composites, mécanique de la rupture, collages, etc.),
- la dynamique des fluides multi-physique (aéroacoustique, aéroélasticité, aérothermique, etc.),
- l'optimisation multi-disciplinaire (optimisation d'aubages, de profils, de structures, etc.).

Cette aide se concrétise en fournissant, au sein de toutes ces disciplines, des services de consultance, des expertises CAE (Computer Aided Engineering) de pointe et des logiciels d'engineering high-tech. Le personnel de Cenaero se compose actuellement d'une cinquantaine de personnes, essentiellement des ingénieurs et des docteurs spécialisés en méthodes numériques, sciences des matériaux, fabrication virtuelle, mécanique des fluides (CFD), physique, mathématiques appliquées. Parmi les principales références de Cenaero, on compte Snecma, Techspace Aero, Airbus, Sabca, Messier-Dowty, Arcelor-Mittal, Renault, Caterpillar, CNES, DCNS et bien d'autres encore. Afin de soutenir toutes ces activités, Cenaero abrite le centre de calcul HPC (High Performance Computing) le plus puissant du pays, appelé *Ernest*. Continuellement upgradé depuis 2004, Ernest possède une capacité de plusieurs dizaines de Teraflops et plus de 2000 processeurs. La puissance de ce super calculateur, nécessaire au développement des nombreuses activités de simulations numériques du centre, peut également être mise à la disposition des partenaires de Cenaero.

Le centre de recherche est divisé en quatre groupes : CFD Multi-Physique, Fabrication Virtuelle, Modélisation Multi-Echelles des Matériaux et Structures, et Méthodes Numériques et Optimisation. Ce dernier étant le groupe qui a encadré ce mémoire, nous détaillons ci-après ses activités. Le centre de recherche Cenaero collabore avec de nombreux industriels, des éditeurs de logiciels et des universités au travers de projets wallons et européens, de projets de consultance et/ou de développement, de recherches avancées, mais également à l'occasion de partenariat avec des étudiants dans la réalisation de mémoires de fin d'études et de thèses de doctorat afin de rester constamment à l'avant-garde de leurs thèmes de recherche privilégiés et de permettre à leur personnel de se spécialiser notamment par le biais de doctorats.

Les activités du groupe Méthodes Numériques et Optimisation [1] se concentrent principalement sur le développement et l'utilisation de la plateforme d'optimisation multi-disciplinaire, **Minamo** [60, 34, 35]. Celle-ci, basée sur des algorithmes génétiques fortement accélérés par des méta-modèles, permet de ré-

1. Les explications concernant les activités des 3 autres groupes de Cenaero peuvent être consultées dans le Rapport annuel 2008, [68] ou sur le site internet www.cenaero.be.

soudre des problèmes mono- et multi-objectif(s) complexes. Grâce à un accès direct aux logiciels CAE durant le processus d'optimisation, Minamo permet la réalisation et l'optimisation de designs multi-disciplinaires notamment pour les industries du secteur aérospatial et automobile. C'est dans le cadre des activités spécifiques de ce groupe que s'inscrit ce mémoire.

1.2 Contexte et objectifs du mémoire

Ce mémoire est issu d'une demande posée par Cenaero dans le cadre des activités du groupe Méthodes Numériques et Optimisation et, plus particulièrement, du développement de méthodes d'optimisation basées sur des algorithmes génétiques fortement accélérés par des modèles [2] approchés.

1.2.1 Description de Minamo

Minamo est le logiciel d'optimisation multi-disciplinaire développé à Cenaero. Il est capable de résoudre des problèmes mono- et multi-objectif(s) dans le cadre d'applications industrielles complexes. Minamo est basé principalement sur des algorithmes génétiques. Ces derniers sont souvent beaucoup plus robustes dans leur capacité à identifier l'optimum global que des algorithmes classiques d'optimisation notamment si le problème n'est pas convexe et contient plusieurs optima locaux. Ils connaissent un très grand succès pour les problèmes d'optimisation présentant des optima multiples, plusieurs objectifs contradictoires ainsi que des variables de nature mixte (i.e., continues et/ou entières). Les algorithmes génétiques offrent également l'avantage d'être particulièrement bien adaptés pour traiter des fonctions fortement non-linéaires, bruitées et/ou discontinues. Leur aptitude à gérer des fonctions qui ne peuvent pas être calculées (par exemple, des simulations CFD non-convergées) est aussi un de leurs nombreux avantages.

Cependant le principal point faible des algorithmes génétiques est le nombre important d'évaluations de la fonction objectif et éventuellement des contraintes. Cela les rend, dans leur forme la plus pure, inapplicables en pratique pour des problèmes où l'évaluation des fonctions en jeu est coûteuse en temps de calcul (ce qui est souvent le cas des études réalisées à Cenaero). Le but est alors de diminuer le nombre d'appels à ces fonctions en utilisant un méta-modèle de celles-ci.

1.2.2 Méthodologie d'optimisation

Le principe de base de l'optimisation assistée par méta-modèle est de construire un modèle approché du problème d'origine (i.e., une approximation de la fonction objectif et éventuellement des contraintes s'il y en a). Ensuite, ce modèle mathématique approché peut être exploité dans une boucle d'optimisation en lieu et place du modèle numérique original. Le modèle approché peut être évalué rapidement et, par conséquent, le nombre d'évaluations du modèle approché par l'algorithme d'optimisation ne constitue plus un problème critique. A la fin de cette phase d'optimisation, basée sur le modèle approché, nous avons recours à la simulation numérique exacte afin d'estimer la qualité de la solution fournie

2. Par la suite, nous utiliserons indifféremment les termes "modèle" et "méta-modèle".

par l'optimisation basée sur le modèle et de recalibrer le modèle approché. Le schéma d'optimisation utilisé par Minamo est représenté à la Figure 1.1 et à la Figure 1.2.

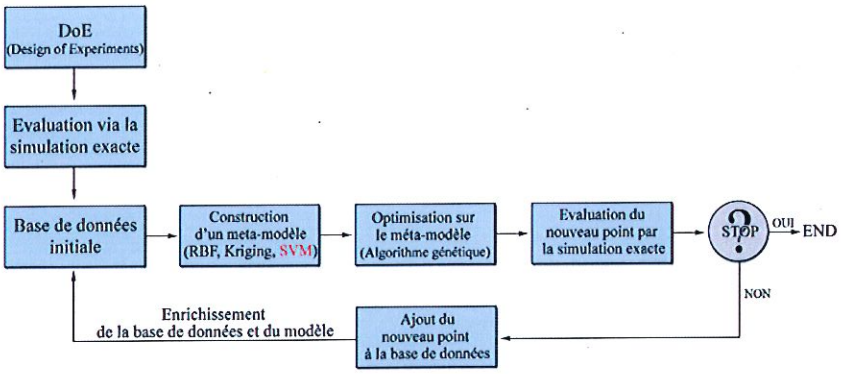


FIGURE 1.1 – Méthodologie générale de l'optimiseur Minamo.

Cette approche, souvent appelée optimisation assistée par méta-modèle, est de plus en plus privilégiée dans le monde industriel car elle permet un gain de temps de calcul crucial. Pour de plus amples informations le lecteur peut consulter les références [24, 25]. Comme l'illustrent les Figures 1.1 et 1.2, la méthodologie implantée dans Minamo peut être décomposée en cinq grandes étapes :

1. Une base de données est générée à l'aide d'une technique de plan d'expériences (Design of Experiments - DoE), celle-ci est représentée par les points bleus au premier schéma de la Figure 1.2. Plusieurs techniques existent, celles implémentées dans Minamo consistent en des procédures de remplissage de l'espace et seront présentées par la suite (voir Section 1.2.4).

2. Un méta-modèle, modélisé par les courbes oranges à la Figure 1.2, est ensuite construit sur base des points du plan d'expériences dans le but de construire une relation analytique entre les paramètres d'entrée et les réponses. Dans notre cas, le modèle sera basé sur les machines à vecteurs de support (SVM).

3. Un algorithme d'optimisation (dans notre cas un algorithme génétique) est appliqué afin de trouver un optimum, représenté par les étoiles rouges à la Figure 1.2, en utilisant le(s) méta-modèle(s) pour estimer la fonction objectif (et les contraintes).

4. La simulation exacte est ensuite utilisée pour évaluer et valider les valeurs de la fonction objectif et des contraintes à l'optimum trouvé à l'étape précédente. Le résultat de ces nouvelles simulations exactes est ajouté à la base de données. Cette dernière est donc continuellement améliorée par de nouveaux points, ce qui permet également une amélioration continue des méta-modèles.

5. Le processus est répété jusqu'à ce qu'un certain nombre maximal d'itérations fixé par l'utilisateur soit atteint.

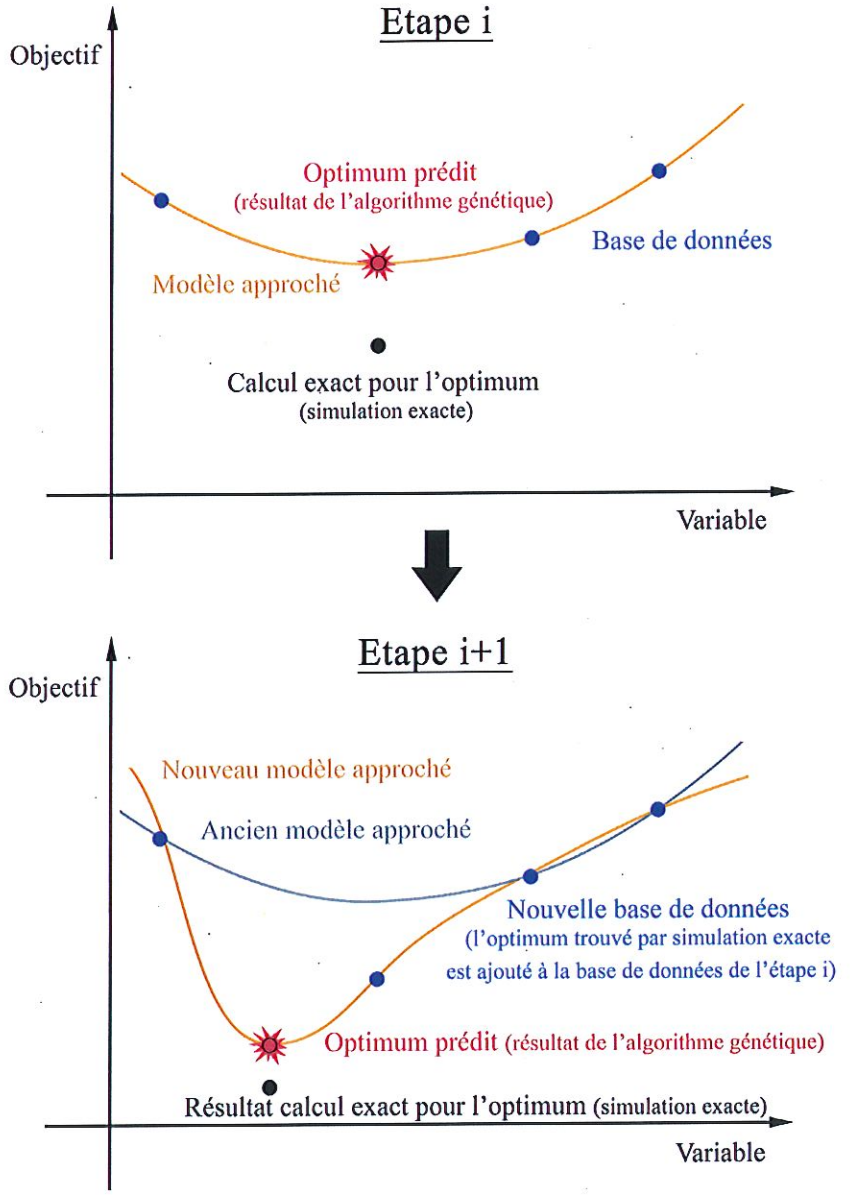


FIGURE 1.2 – Illustration de l'optimisation assistée par méta-modèle à la base de Minamo.

1.2.3 Algorithme génétique

Outre l'exploitation de méta-modèles, un composant majeur de l'algorithme d'optimisation de Minamo est un algorithme génétique. Ce type d'algorithme est classifié comme une méthode d'ordre zéro car il n'utilise pas d'information sur les dérivées. Les algorithmes génétiques font partie des méthodes stochastiques (en opposition aux méthodes déterministes). Leur but est d'obtenir un optimum global approché et non un optimum local (contrairement aux méthodes de type gradient par exemple) à un problème d'optimisation quelconque en un temps fini acceptable. Ils sont utilisés en général lorsque l'on ne dispose pas de méthode exacte (i.e., de méthode qui garantit l'obtention de la solution optimale pour un problème d'optimisation donné) pour résoudre le problème d'optimisation.

Le principe général des algorithmes génétiques repose sur la théorie de Darwin de la sélection naturelle, à savoir : "les individus les plus adaptés tendent à survivre plus longtemps et à se reproduire plus aisément". Les algorithmes génétiques sont basés sur des évolutions stochastiques successives d'un ensemble de solutions candidates appelées individus. Cet ensemble est lui-même appelé une population. L'algorithme génétique implémenté dans Minamo utilise un codage réel dans lequel les paramètres d'entrée sont toujours représentés par des valeurs réelles, en opposition au codage binaire qui est aussi souvent utilisé. Les explications du processus général d'un algorithme génétique sont données ci-après et ce dernier est illustré par la Figure 1.3.

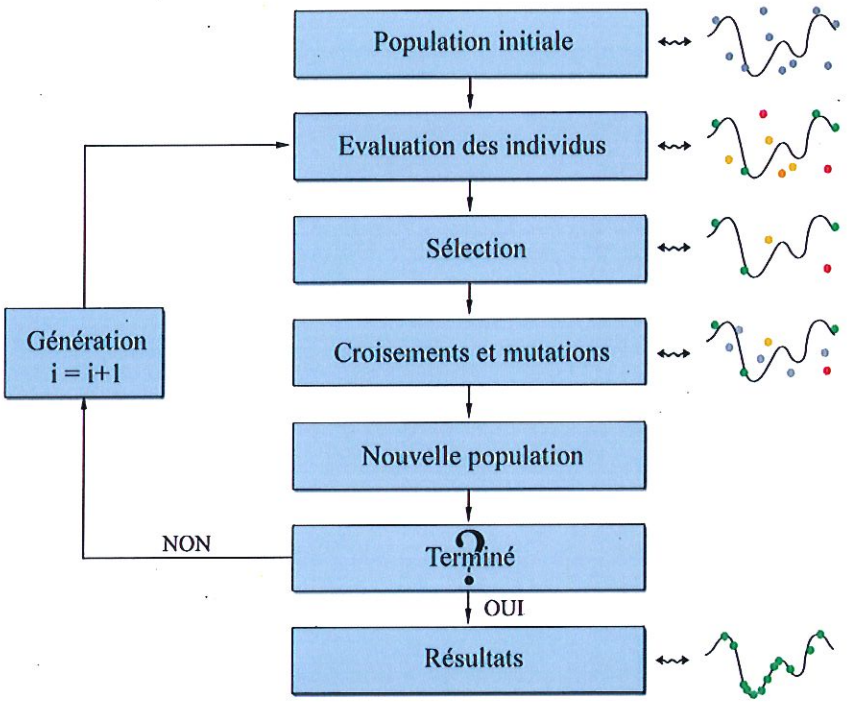


FIGURE 1.3 – Etapes d'un algorithme génétique.

Le processus général d'un algorithme génétique se déroule de la façon suivante :

1. Une population initiale est générée aléatoirement. Celle-ci contient des individus qui possèdent les bonnes caractéristiques (nombre et type de gènes corrects, etc.).

2. Une valeur réelle, appelée *fitness*, peut être attribuée à chaque individu de la population initiale. Cette valeur peut être vue comme une cote, une manière de quantifier la qualité de l'individu. Cette fitness est attribuée de différentes manières selon le problème traité.

3. Une fois les valeurs de fitness connues pour chaque individu, l'algorithme sélectionne dans la population les individus susceptibles de donner naissance à des descendants grâce à un opérateur de sélection. Quelques exemples de sélection sont présentés ci-dessous :

   - *Sélection par tournoi* : si la population comporte n individus, n duels sont effectués. À chaque duel, deux individus sont tirés au hasard. Nous sélectionnons celui des deux qui a la meilleure fitness. Cet individu peut alors donner naissance à un descendant.

   - *Sélection par rang* : ce type de sélection choisit toujours les m individus ayant la meilleure fitness. Le hasard n'entre donc jamais en jeu.

   - *Sélection uniforme* : chaque individu a une probabilité identique d'être sélectionné. Les valeurs de fitness n'entrent donc jamais en ligne de compte.

   - *Sélection par roulette* : chaque individu a une probabilité proportionnelle à sa fitness d'être sélectionné.

4. Les gènes de deux individus "parents" sélectionnés dans la population sont recombinés afin de donner naissance à un descendant. Cette étape se fait via un opérateur de croisement. Le croisement consiste à mélanger les gènes des parents. Il n'a pas lieu dans tous les cas : la plupart des algorithmes génétiques utilisent une probabilité de croisement.

5. Une mutation est appliquée au descendant, avec une faible probabilité (généralement comprise entre 0.001 et 0.01), en introduisant du bruit dans les gènes de ce nouvel individu. L'opérateur de mutation assure la diversité dans la population, évitant une convergence prématurée de l'algorithme et réduisant la possibilité de rester coincé à un optimum local.

6. Le processus (i.e., les étapes 2 à 5) est répété jusqu'à ce qu'un nombre maximum de générations soit atteint, dû à un temps de calcul limité. Il peut arriver que tous les individus de la population convergent vers un même individu ayant une fitness élevée, mais non optimale. En effet, lorsqu'une population évolue, il se peut que certains individus qui, à un instant donné, occupent une place importante au sein de cette population, deviennent majoritaires. À ce moment, il se peut que la population converge vers cet individu et s'écarte ainsi d'individus plus intéressants mais trop éloignés de l'individu vers lequel on converge. Outre l'opérateur de mutation, il existe des techniques qui permettent d'éviter pareille situation comme l'ajout d'individus générés aléatoirement à chaque génération ou encore des procédures dites de *sharing* qui diminuent la valeur de la fitness des individus en fonction de leur densité dans l'espace de conception (espace dans lequel les individus sont choisis), afin d'éviter une convergence prématurée de l'algorithme. Nous ne donnerons pas plus de détails sur ces méthodes qui sortent du cadre de ce mémoire [32].

L'algorithme génétique implémenté dans Minamo utilise la fonction objectif du problème à optimiser comme fonction fitness des individus, lorsqu'il n'y a pas de contrainte à satisfaire. Les individus sont déterminés par une sélection par tournoi. Ceci évite des problèmes de mise à échelle si la fitness d'un individu est négative. Les croisements sont effectués par un algorithme qui allie plusieurs techniques dont le *line-crossover* et le *box-crossover*. Il est à noter que les étapes de sélection, croisement et mutation [3] sont réalisées un nombre de fois égal au nombre d'individus de la population, ceci afin que la nouvelle population comporte un nombre d'individus égal à celui de l'ancienne population. Finalement, Minamo implémente une méthode d'élitisme. Celle-ci permet, lors de la création d'une nouvelle population, d'éviter de perdre les meilleurs individus. Cette méthode consiste simplement à copier un ou plusieurs des meilleurs individus dans la nouvelle population. Pour plus de précisions concernant les algorithmes génétiques, le lecteur peut se référer à [32].

1.2.4 Plan d'expériences

Comme nous l'avons vu dans la structure générale de la méthodologie implémentée dans Minamo, un ensemble de données, représentées par des points bleus sur les schémas 1, 2 et 3 de la Figure 1.5, évaluées par la simulation exacte, appelé plan d'expériences, doit être disponible afin de construire les méta-modèles. Dans Minamo, plusieurs techniques d'échantillonnage, dites de remplissage de l'espace, ont été mises en oeuvre.

La première est l'échantillonnage hypercube Latin (LHS), cf le schéma 1 de la Figure 1.5, qui présente l'avantage d'être facile à implémenter et rapide. De plus, cette technique ne dépend pas de la dimension de l'espace de conception. LHS est une technique basée sur la propriété du carré Latin. Un tableau carré contenant des points est un carré Latin si il y a seulement un point dans chaque ligne et dans chaque colonne. Cette propriété est illustrée à la Figure 1.4. L'hypercube Latin est la généralisation de cette notion à un nombre arbitraire de

3. L'étape de la mutation dépend en fait d'une probabilité de mutation et est donc réalisée au plus un nombre de fois égal au nombre d'individus de la population. Nous ne rentrons pas plus en détail concernant cette probabilité car celle-ci sort du cadre de ce mémoire.

dimensions. La distribution des points LHS n'est pas unique et il se peut que la répartition des points dans l'espace de conception soit pauvre dans certaines régions. Celles-ci sont représentées par les zones rouge clair du schéma 1 de la Figure 1.5.

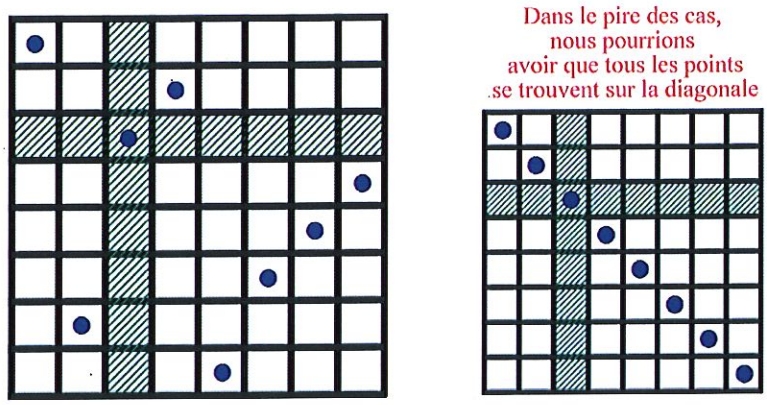


FIGURE 1.4 – Propriété LHS.

La seconde technique intégrée dans Minamo est la méthode CVT (Centroïdal Voronoi Tessellation), cf. le schéma 2 de la Figure 1.5. Une tessellation de Voronoi est une manière de diviser l'espace de conception en compartiments, basée sur la proximité de chaque point à certains points appelés les générateurs. De plus, cette technique ne dépend pas non plus de la dimension de l'espace de conception. Les points issus d'un échantillonnage CVT sont meilleurs que les points issus de LHS en terme de mesures volumétriques de la qualité de l'échantillonnage. En effet, nous remarquons que les points issus d'un échantillonnage CVT sont bien répartis dans l'espace de conception, comme illustré par le schéma 2 de la Figure 1.5, c'est-à-dire que la distance séparant les différents points est plus ou moins uniforme contrairement à la distance entre les points issus d'un échantillonnage LHS. Cependant, le plan d'expériences obtenu par CVT peut présenter de faibles capacités de distribution par rapport aux points LHS. Cela signifie que si nous projetons les poins CVT sur une des "faces" de l'hypercube, ces points ne seront pas bien distribués contrairement aux points LHS. De plus, avec la méthode CVT, très peu de points sont placés aux bornes et dans les régions proches de celles-ci. Pour de plus amples informations le lecteur peut se référer à [61].

La troisième technique d'échantillonnage implémentée dans Minamo est une méthode hybride qui combine les avantages des deux techniques évoquées ci-dessus, d'où son nom de CVT Latinisée (LCVT), cf. le schéma 3 de la Figure 1.5. L'idée est de commencer par créer un échantillonnage avec la méthode CVT et d'ensuite appliquer une latinisation à cet ensemble de points, c'est-à-dire que nous transformons l'ensemble des points CVT en un autre ensemble de points proches qui satisfont la propriété de l'hypercube Latin. La méthode LCVT tente donc d'obtenir des propriétés de bonne dispersion dans deux sens

opposés : LCVT produit un échantillonnage présentant une meilleure capacité de distribution que CVT tout en préservant une uniformité volumétrique meilleure qu'avec LHS. De plus, la technique LCVT place des points aux bornes et près de celles-ci contrairement à la technique CVT. Ceux-ci sont entourés par des cercles oranges au schéma 3. Tout comme les techniques LHS et CVT, cette technique ne dépend pas non plus de la dimension de l'espace de conception. La Figure 1.5 représente les trois techniques d'échantillonnage implémentées dans Minamo générant 30 points dans l'espace de conception $[-2, 2] \times [-2, 2]$.

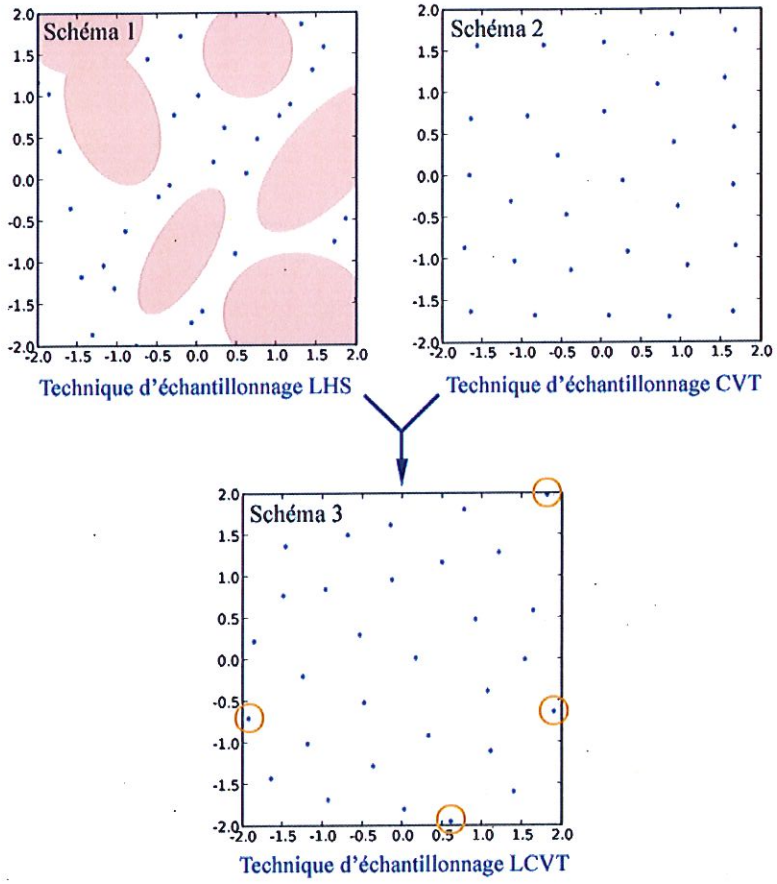


FIGURE 1.5 – Exemple des trois techniques d'échantillonnage : LHS, CVT et CVT Latinisée.

Dans le but d'encore améliorer le méta-modèle qui sera associé au plan d'expériences et donc de localiser l'optimum global avec plus de précision et à moindre coût, une procédure d'échantillonnage auto-adaptative de l'espace de conception a également été implémentée dans Minamo. Démarrant d'un plan d'expériences initial, réalisé par une des méthodes pré-citées (LHS, CVT, LCVT), l'échantillonnage auto-adaptatif permet d'augmenter localement la densité de points dans les zones plus complexes de l'espace de conception en se

basant sur les points du plan d'expériences actuel ainsi que sur la qualité du méta-modèle construit avec ces points. Pour de plus amples informations le lecteur peut se référer à [60].

1.2.5 Méta-modèles

Résoudre un problème d'optimisation en ayant recours à un méta-modèle du modèle de la simulation exacte présente un intérêt majeur. En effet, l'évaluation exacte via la simulation haute-fidélité nécessite un temps conséquent pouvant aller jusqu'à plusieurs heures, voire plusieurs jours de calcul. Dès lors, réaliser l'optimisation directement sur la simulation exacte deviendrait extrêmement long du fait des nombreux appels à l'évaluation de la fonction objectif. Si chacun des appels nécessitait plusieurs heures de calcul, la procédure complète d'optimisation ne serait pas réalisable dans des délais acceptables. En utilisant un méta-modèle, la convergence du processus d'optimisation est considérablement accélérée.

Les différentes méthodes d'approximation disponibles actuellement dans Minamo sont des méthodes génériques d'approximation comme les réseaux de fonctions à base radiale (RBF) ainsi que le kriging [25] ordinaire et universel qui est une méthode d'interpolation issue de la géostatique [24]. Dans ce mémoire, nous utiliserons les machines à vecteurs de support pour la construction de méta-modèles fiables et s'adaptant facilement à des fonctions non-linéaires, comme cela est expliqué à la Section 1.2.7.

La qualité des modèles augmente avec le nombre d'itérations, puisque de nouveaux points sont ajoutés à la base de données. Par conséquent, la procédure dite du *move-limits* adapte les bornes des variables au cours du processus d'optimisation. Le but de la stratégie de move-limits est de confiner la recherche de l'optimum dans une région de l'espace de conception et donc de stabiliser et d'augmenter la vitesse de convergence de l'algorithme.

L'optimisation se faisant sur des méta-modèles, de la (ou des) fonction(s) objectif(s) et des contraintes, il est donc primordial d'essayer d'évaluer la qualité de ces surfaces de réponses construites à partir de la base de données. La procédure de *cross-validation* basée sur le test du *Leave-One-Out* (LOO), permettant d'estimer la qualité d'un modèle, est une technique qui ne nécessite pas la création d'un ensemble supplémentaire de points pour la validation.

L'idée principale des différents tests permettant d'évaluer la fiabilité des méta-modèles sera présentée en détail à la Section (2.3.1) ainsi qu'au Chapitre 5.

1.2.6 Objectif du mémoire

L'objectif de ce mémoire est l'étude et l'implémentation d'une méthode d'approximation de fonctions, à savoir les machines à vecteurs de support ou séparateurs à vaste marge (SVM). Les SVM sont généralement utilisés pour la classification de données. Dans ce cas, le but des SVM est de trouver un hyperplan qui sépare et maximise la marge de séparation entre deux classes comme nous le verrons par la suite. Le problème de la recherche de l'hyperplan séparateur possède une formulation duale, ce qui permet au problème d'être résolu à l'aide de méthodes d'optimisation quadratique. Cependant, nous pouvons aussi utiliser les SVM comme technique de prédiction de fonctions. Finalement, nous comparerons les résultats obtenus pour les SVM avec d'autres méthodes d'approxi-

mations disponibles dans le logiciel Minamo sur des fonctions mathématiques présentant des difficultés variées.

Cette nouvelle méthode d'approximation se basant sur les SVM sera intégrée dans Minamo et devra donc pouvoir fonctionner avec les autres programmes utilisés par Minamo. C'est pourquoi l'implémentation de cette méthode a, en partie, été réalisée sur le site même de Cenaero. Puisque nous devons intégrer cette implémentation au logiciel Minamo existant, il est indispensable dans un premier temps de comprendre le fonctionnement de ce programme afin de pouvoir en modifier certaines parties.

Pour les optimisations réalisées par Cenaero, la contrainte de temps est essentielle. La méthode proposée doit donc pouvoir traiter les problèmes en un temps acceptable. En effet, Cenaero réalise de la consultance et doit donc être en mesure de fournir une réponse à ses clients dans des délais raisonnables.

Finalement, il est souhaitable que la méthode d'approximation basée sur les SVM, qui est le sujet d'étude de ce mémoire, retourne une solution acceptable sinon meilleure que celle proposée par les méthodes actuellement utilisées dans Minamo et qu'elle puisse aussi améliorer la réaction du modèle face au bruit.

Chapitre 2

L'apprentissage automatique

RÉSUMÉ. *Dans ce chapitre, nous introduisons de manière formelle les concepts fondamentaux de l'apprentissage automatique à partir de données. Nous introduisons également plusieurs définitions, termes et notations importantes qui seront nécessaires par la suite dans ce mémoire. Ensuite, nous rappelons et détaillons les problèmes bien connus de la classification et de la régression. Ces deux techniques sont à la base de la construction de systèmes, problèmes plus complexes, dont celui traité dans ce mémoire à savoir, les SVM. Finalement, nous introduisons les modèles non-linéaires et la question du sur- et sous-apprentissage. Ce chapitre se base principalement sur [40, 6, 12, 67, 42].*

2.1 Problème général

La résolution de problèmes mathématiques par la construction de machines capables d'apprendre à partir d'expériences caractérise l'approche fondamentale de l'apprentissage automatique souvent aussi appelé Machine Learning (ML). Parmi les techniques de ML, nous trouvons principalement les méthodes supervisées et les méthodes non supervisées. Dans ce mémoire, nous nous intéressons uniquement aux méthodes d'apprentissage supervisées. Celles-ci ont pour but de construire une fonction f qui permet d'approcher au mieux une fonction inconnue à partir d'un ensemble fini de données étiquetées (i.e., du type entrée - sortie) appelé *ensemble d'apprentissage.*

Définition 2.1.1 *(Ensemble d'apprentissage, vecteur d'entrée et de sortie) Soit n couples "entrée - sortie" représentant les données,*

$$\mathcal{S} = \{(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_n, y_n)\}$$

où $x_i \in \mathcal{X}$ est le ième vecteur d'entrée et $y_i \in \mathcal{Y}$ est le vecteur de sortie correspondant, l'ensemble de ces n données est appelé ensemble d'apprentisage.

Afin de définir le problème général étudié dans ce mémoire, nous décrivons ci-après le fonctionnement d'un système d'apprentissage.

Un système d'apprentissage automatique à partir de données peut être décrit à l'aide de trois modules. Ceux-ci ont été introduits par Vapnik [66] et sont illustrés à la Figure 2.1.

1. **Un générateur** de données aléatoires $x \in \mathcal{X}$, (i.e., de vecteurs d'entrées). Ces vecteurs d'entrées sont supposés indépendants et identiquements distribués (i.i.d.) suivant une distribution de probabilité inconnue $p(x)$. Dans notre cas, les générateurs possibles sont les techniques d'échantillonnage LHS, CVT et LCVT (voir Section 1.2.4).

2. **Un superviseur** qui à chaque vecteur d'entrée x associe un vecteur de sortie $y \in \mathcal{Y}$ suivant une distribution de probabilité également inconnue, définie sur les paires $(x, y) \in \mathcal{X} \times \mathcal{Y}$, $p(x, y)$. Pour de plus amples informations le lecteur peut se référer à [16]. La sortie y est donc obtenue par une fonction inconnue g, qui dépend de $p(x, y)$, appliquée au vecteur d'entrée x, $y = g(x)$.

3. **Une machine d'apprentissage** qui a pour but de trouver une fonction f appartenant à une classe $\mathcal{F}$ de fonctions admissibles (i.e., fonctions qui vérifient une structure particulière ou qui satisfont à certaines hypothèses) qui approximent la fonction g,

   $$\mathcal{F} = \{f_\alpha \text{ tel que } \alpha \in \Lambda \text{ où } \Lambda \text{ est un ensemble de paramètres}\}.$$

   Cette fonction f produit donc pour le vecteur d'entrée x une sortie $\hat{y}$ qui est la plus proche possible de la sortie y du superviseur.

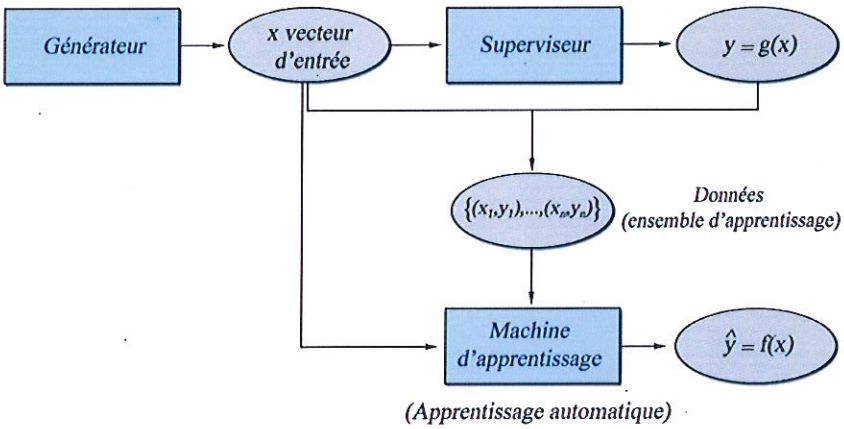


FIGURE 2.1 – Machine Learning supervisée. La fonction $f(x)$ est une estimation de la fonction inconnue $g(x)$ transformant x en y.

Le problème général étudié dans ce mémoire peut maintenant être formulé.

Définition 2.1.2 *(Problème général)*
Soit l'ensemble d'apprentissage

$$\mathcal{S} = \{(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_n, y_n)\}.$$

Le problème général de l'apprentissage automatique consiste à trouver un modèle (i.e. une fonction) $f \in \mathcal{F}$ qui approxime au mieux la fonction inconnue transformant les vecteurs d'entrées $x_i \in \mathcal{X}$ en $y_i \in \mathcal{Y}$.

Afin de déterminer quelle est la meilleure approximation f de la fonction inconnue g, nous devons introduire un critère de sélection. Celui-ci est basé sur la minimisation d'un certain risque et est introduit à la Section (2.1.1).

Remarques

1. Parmi les techniques de ML supervisées, nous pouvons par exemple citer les méthodes de classification, qui seront présentées plus en détail à la Section 2.4. Dans ce cas, une donnée est un objet ou sa représentation accompagnée de la catégorie à laquelle il appartient. La fonction $f(x)$ estimée détermine alors la relation entre les différents objets et leur catégorie.

2. Dans le cas des techniques non-supervisées, les objets sont présentés sans leur catégorie. Un exemple type des méthodes non-supervisées est le clustering. Nous n'entrerons pas plus dans les détails concernant les techniques non-supervisées car ce n'est pas le but de ce mémoire.

2.1.1 Fonction d'erreur et risque

Soit l'ensemble d'apprentissage

$$\mathcal{S} = \{(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_n, y_n)\}$$

dont les données sont tirées, comme expliqué précédemment, suivant une distribution de probabilité inconnue $p(x, y)$. Notre objectif est de trouver $f \in \mathcal{F}$ telle que l'estimation $f(x) = \hat{y}$ de la réponse du superviseur soit la meilleure possible. Le choix de cette fonction f va dépendre des n données de l'ensemble d'apprentissage i.i.d. suivant la distribution de probabilité $p(x, y)$ mais aussi de la minimisation d'un risque $R[f]$. Le risque $R[f]$ du modèle f, appelé *risque fonctionnel*, est défini comme le décalage entre la réponse y du superviseur pour le vecteur d'entrée x et la réponse $f(x)$ fournie par la machine d'apprentissage. Ce décalage est mesuré, donnée par donnée, à l'aide d'une *fonction coût* $l(f(x), y)$ où $f \in \mathcal{F}$. Le risque fonctionnel est donné par

$$R[f] = E[l(f(x), y)] = \int_{\mathcal{X} \times \mathcal{Y}} l(f(x), y)\, p(x, y)\, dxdy. \tag{2.1}$$

Finalement, la meilleure approximation correspondra à la fonction f qui minimise ce risque

$$f = \underset{f_\alpha \in \mathcal{F}}{arg\ min}(R[f_\alpha]).$$

Remarques

1. Le risque fonctionnel dépend de la famille de fonctions $\mathcal{F}$ et de la distribution de probabilité $p(x, y)$.

2. Le risque fonctionnel représente l'erreur moyenne commise sur toute la distribution $p(x, y)$ par la fonction $f(x)$.

3. $E[l(f(x), y)]$ représente l'espérance de la fonction coût $l(f(x), y)$.

Le risque fonctionnel ne peut cependant pas être calculé explicitement car nous ne connaissons pas la distribution de probabilité $p(x, y)$. Donc les algorithmes d'apprentissage doivent estimer le risque $R[f]$ en utilisant uniquement l'ensemble d'apprentissage $\mathcal{S}$. Pour cela, lors de l'apprentissage automatique, une bonne estimation du risque fonctionnel est donnée par le *risque empirique*. Celui-ci est défini comme suit

$$R_{emp}[f] = E_{\mathcal{S}}[f] = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i) \tag{2.2}$$

où $n = |\mathcal{S}|$ est le cardinal de l'ensemble $\mathcal{S}$ (i.e., le nombre de données de l'ensemble d'apprentissage). Afin de sélectionner la meilleure fonction $f(x) \in \mathcal{F}$, il faut donc déterminer l'ensemble de paramètres $\alpha \in \Lambda$ dont la fonction f dépend tel que $R_{emp}[f]$ soit minimal. En d'autres mots, le critère de sélection de la meilleure fonction f est

$$f = \underset{f_\alpha \in \mathcal{F}}{arg\ min}(R_{emp}[f_\alpha]).$$

Ce critère de sélection est appelé *Minimisation du Risque Empirique* (MRE). De plus, l'estimation du risque fonctionnel peut être utilisée pour évaluer la performance des algorithmes d'apprentissage et des modèles en résultant. Plusieurs estimations du risque fonctionnel sont données à la Section 2.3 mais nous devons pour cela définir les hyperparamètres. Dans les sections suivantes nous traitons les problèmes de la classification et de la régression et nous introduisons les choix de la fonction de coût l correspondant à ces deux problèmes.

2.2 Les hyperparamètres

Les paramètres rencontrés dans le processus d'apprentissage automatique d'un modèle peuvent être divisés en deux catégories.

- **Les paramètres** α sont les différents paramètres du modèle, c'est-à-dire de la fonction $f \in \mathcal{F}$. Les valeurs de ces paramètres sont déterminées par l'algorithme d'apprentissage (i.e., la machine d'apprentissage).

  Exemple de paramètres α

  Le vecteur w et la constante b qui déterminent l'hyperplan dans le cas des SVM (voir Chapitres 3 et 4) font partie de ces paramètres.

- **Les hyperparamètres** η sont des paramètres complexes qui influencent la procédure d'apprentissage et donc la forme générale du modèle. Ce sont des constantes qui définissent une instance particulière d'un algorithme d'apprentissage. Généralement, ces paramètres sont fixés au moment de l'implémentation ou de l'exécution de l'algorithme par le programmeur. Cependant, il existe des méthodes qui permettent de déterminer automatiquement ces hyperparamètres.

  Exemple d'hyperparamètres η

  Les variables ε et C du problème d'optimisation à marge souple pour les SVM (voir Chapitres 3 et 4) ou encore les paramètres des noyaux (voir Section 3.4.2) sont des hyperparamètres.

En d'autres mots, α sont les paramètres du modèle et η sont les paramètres de l'algorithme d'apprentissage. Déterminer les hyperparamètres η optimaux minimisant le risque empirique (2.1) est une tâche complexe et requiert souvent de devoir estimer les performances du modèle (voir Section 2.3).

Une méthode souvent employée est de réaliser une recherche à l'aide d'une grille, c'est-à-dire de tester différentes valeurs pour η et d'estimer le risque fonctionnel (2.1) impliquant donc la construction d'un modèle pour chaque jeu de valeurs des hyperparamètres η. Le meilleur jeu d'hyperparamètres correspond au meilleur modèle $f \in \mathcal{F}$ (i.e., ayant le risque fonctionnel minimum). Cette procédure peut être très coûteuse en terme de temps de calcul mais peut être accélérée en commençant la recherche sur une grille générale et en restreignant ensuite celle-ci autour des valeurs donnant les meilleurs résultats (i.e., modèles). Cependant, cette méthode est difficilement utilisable pour plus de deux hyperparamètres car elle serait beaucoup trop coûteuse en temps de calcul et donc d'exécution [1]. Pour un exemple de la stratégie de recherche des hyperparamètres à l'aide d'une grille on se référera au Chapitre 6. De plus, d'autres méthodes ont été proposées citons par exemple les approches évolutionnaires [26] ou encore les approches de type gradient [13, 4]. Ces méthodes ne sont pas développées dans ce mémoire.

2.3 Estimation du risque fonctionnel

Une des procédures la plus simple pour estimer le risque (2.2) d'une machine d'apprentissage est de calculer l'erreur moyenne

$$E_{\hat{\mathcal{S}}}[f] = \frac{1}{|\hat{\mathcal{S}}|} \sum_{i=1}^{|\hat{\mathcal{S}}|} l(f(x_i), y_i) \tag{2.3}$$

sur un ensemble indépendant de données $\hat{\mathcal{S}}$. Lorsque les données de $\hat{\mathcal{S}}$ ne sont ni utilisées pour l'apprentissage ni pour le calcul des hyperparamètres, alors $\hat{\mathcal{S}}$

1. Comme expliqué au Chapitre 1, pour Cenaero le temps de calcul est crucial. Un temps d'exécution de plusieurs heures pour tester tous les hyperparamètres possibles dans le but de construire le meilleur modèle n'est absolument pas envisageable.

est appelé *l'ensemble test* et l'estimation $E_{\hat{\mathcal{S}}}[f]$ du risque fonctionnel est appelée *l'erreur de test.* D'autre part, lorsque les données de $\hat{\mathcal{S}}$ sont utilisées pour l'apprentissage, alors $\hat{\mathcal{S}}$ est appelée *l'ensemble de validation* et $E_{\hat{\mathcal{S}}}[f]$ est appelé *l'erreur de validation.*

2.3.1 Validation croisée

Lorsque nous disposons d'un ensemble d'apprentissage dont le nombre de données est limité, nous ne pouvons pas en extraire une partie pour la validation d'un modèle ou pour différents tests. Dans ce cas, une validation croisée doit être utilisée afin de pouvoir estimer $R[f]$ et ainsi la qualité du modèle. Cette procédure se déroule comme suit :

Algorithme 1 Validation croisée

1: Diviser l'ensemble d'apprentissage $\mathcal{S}$ en $k \in \mathbb{N}$ sous-ensembles

$$\mathcal{S}_j, \quad \forall j : 1 \leq j \leq k$$

de $\lfloor \frac{n}{k} \rfloor$ données, où $\lfloor \cdot \rfloor$ représente la partie entière.
2: **for** $j = 1$ jusque k **do**
3: Construire un modèle f à partir de $\mathcal{S} \backslash \mathcal{S}_j$.
4: Evaluer l'erreur $E_{\mathcal{S}_j}[f]$ de f sur le sous-ensemble $\mathcal{S}_j$ restant par (2.3).
5: **end for**
6: Calculer l'estimation $R_{estimate}[f]$ comme étant l'erreur moyenne

$$R_{estimate}[f] = \frac{1}{k} \sum_{j=1}^{k} E_{\mathcal{S}_j}[f].$$

Plus k sera grand, plus l'estimation sera précise. Il est à noter que le temps de calcul augmente avec k et peut donc être très long. Une estimation par validation croisée quasi non biaisée, appelée test du *Leave-One-Out* (LOO), est obtenue pour le cas limite $k = n$. C'est donc un cas particulier de validation croisée qui permet d'estimer la qualité d'un modèle et ne nécessite pas la création d'un ensemble supplémentaire de données pour la validation ou pour différents tests. Le principe général du LOO est donc de créer un modèle en utilisant $n - 1$ données. La donnée exclue est estimée avec le modèle construit et cette prédiction est alors comparée avec la valeur exacte en ce point. Ce processus est répété pour chacun des n points de l'ensemble d'apprentissage, nous obtenons ainsi un ensemble de réponses estimées (avec le LOO) qui peut être comparé avec les n valeurs exactes. Ces deux ensembles de valeurs sont alors analysés statistiquement. La qualité de la corrélation entre ces valeurs peut être mesurée par le coefficient de corrélation, noté R. La définition du coefficient de corrélation est donnée à la page suivante.

Définition 2.3.1 *(Coefficient de corrélation)*
Supposons que nous ayons les tableaux suivants : $(z_1, ..., z_n)$ *et* $(y_1, ..., y_n)$ *pour les réponses estimées et les valeurs exactes. Alors, le coefficient de corrélation* R *est donné par*

$$R = \frac{\sum_{i=1}^{n}(z_i - \bar{z})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(z_i - \bar{z})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.4}$$

et est toujours compris entre -1 *et* 1. *Plus proche de* 1 *sera le coefficient de corrélation, meilleur sera le modèle.*

Pour la validation des hyperparamètres η sur un ensemble d'apprentissage assez important, la procédure LOO peut devenir très coûteuse en temps de calcul car elle nécessite la construction de n modèles pour chaque valeur de η. La validation croisée avec un $k << n$ offre une alternative pratique. C'est pourquoi nous avons choisi d'utiliser cette technique.

Une procédure de validation croisée a été implémentée dans le logiciel Minamo afin de pouvoir évaluer les performances de l'algorithme SMO (voir Chapitre 5). De plus, la procédure de recherche des hyperparamètres se base sur l'estimation du modèle obtenue par la validation croisée afin de sélectionner les meilleurs hyperparamètres. Pour de plus amples informations le lecteur se reportera au Chapitre 6.

2.4 La classification

La classification a pour but de trouver un modèle appelé *classificateur* qui associe à un vecteur d'entrée (généralement décrit par un ensemble de caractéristiques) une classe. Ce modèle permet donc la reconnaissance (i.e., classification) de nouveaux éléments représentés par des vecteurs d'entrées. Il existe de nombreuses applications de la classification, citons par exemple, la reconnaissance d'images, de caractères ou encore la reconnaissance vocale. Par la suite, nous décrivons l'approche suivie dans ce mémoire dans laquelle un classificateur correspondra à un ensemble de surfaces séparatrices dans l'espace des entrées. Le problème de classification supervisée à n classes peut être énoncé comme suit.

Définition 2.4.1 *(Le problème de la classification et label (classe))*
Soit l'ensemble d'apprentissage

$$\mathcal{S} = \{(x_1, y_1), ..., \ldots, (x_n, y_n)\}$$

où $x_i \in \mathbb{R}^p$ *et* $y_i \in \{1, ..., m\}$, *avec* $m \leq n$ *représente la classe d'appartenance du vecteur d'entrée* x_i. *Dans le cas de la classification, le vecteur de sortie* y_i *est appelé label ou classe.*

Le problème de la classification a pour but de trouver un classificateur

$$f : \mathcal{X} \to \mathcal{Y}, \quad \text{où} \quad f \in \mathcal{F}$$

qui détermine correctement l'appartenance à une classe d'entrées inconnues x, *sur base de l'ensemble d'apprentissage* $\mathcal{S}$.

Dans ce contexte, l'espace de sortie $\mathcal{Y}$ contient seulement des valeurs discrètes et la fonction de coût peut être définie comme

$$l(f(x), y) = \begin{cases} 1 \text{ si } f(x) \neq y, \\ 0 \text{ sinon.} \end{cases} \tag{2.5}$$

Le risque fonctionnel $R[f]$ d'un classificateur peut être vu comme la probabilité que f ne classifie pas correctement une donnée tirée de manière aléatoire à partir d'une distribution $\mathcal{D}$. Cependant, le risque fonctionnel devient une fonction non lisse et non différentiable, elle ne peut donc plus être minimisée. C'est pourquoi, les algorithmes d'apprentissage implémentent généralement des approximations lisses de (2.5). Dans les sections suivantes, nous décrivons l'approche suivie dans ce mémoire, dans laquelle un classificateur correspond à un ensemble de surfaces séparatrices dans l'espace d'entrée.

2.4.1 La classification binaire

Dans ce mémoire, toutes les méthodes sont présentées dans le contexte de la classification binaire (i.e., problème à deux classes : m = 2), dans laquelle les labels $y_i \in \mathcal{Y} = \{-1, +1\}$ peuvent uniquement prendre deux valeurs distinctes. La plupart du temps, nous interprétons $+1$ et -1 respectivement comme l'appartenance et la non appartenance à une catégorie déterminée. Dans ce cas, le problème de la classification consiste à trouver une *surface séparatrice* $\mathcal{H}$ séparant l'espace des entrées $\mathcal{X}$ en deux demi-espaces, chacun d'eux étant assigné à une classe. La surface séparatrice peut être décrite par une fonction à valeur réelle h comme

$$\mathcal{H} = \{x : h(x) = 0\}.$$

Le classificateur f correspondant classifie alors une entrée x par rapport au côté de $\mathcal{H}$ où se trouve x. Sa sortie est donc donnée par

$$f(x) = sign(h(x)), \tag{2.6}$$

comme illustré à la Figure 2.2 pour le cas linéaire (voir Section 2.4.2).

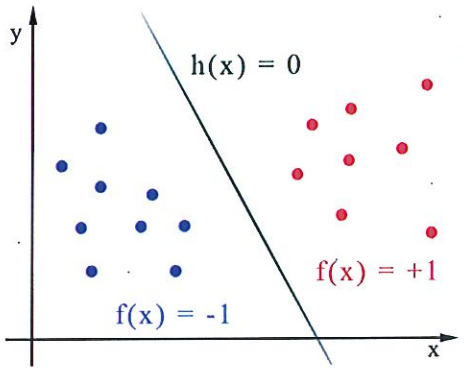


FIGURE 2.2 – Classificateur linéaire séparant les points bleus et les points rouges par un hyperplan.

2.4.2 La classification linéaire

Pour la classification linéaire, la surface de séparation $\mathcal{H}$ est appelée un *hyperplan* et est défini par

$$\mathcal{H} = \{x : h(x) = \langle w, x \rangle + b = 0\} \tag{2.7}$$

où le vecteur w et b sont des paramètres et $\langle\ ,\ \rangle$ représente le produit scalaire dans l'espace d'entrée $\mathcal{X}$. Trouver un hyperplan compatible avec l'ensemble d'apprentissage

$$\mathcal{S} = \{(x_1, y_1), ..., \ldots, (x_n, y_n)\}$$

peut être écrit comme le problème suivant

$$\text{trouver } f(x) \text{ tel que } y_i = f(x_i) = sgn(\langle w, x_i \rangle + b) = \begin{cases} 1 \\ -1 \end{cases}, \forall i : 1 \leq i \leq n, \tag{2.8}$$

ou de manière équivalente [2]

$$y_i(\langle w, x_i \rangle + b) > 0, \quad i = 1, ..., n. \tag{2.9}$$

S'il existe un tel hyperplan, alors l'ensemble d'apprentissage est dit *linéairement séparable*. Une définition plus précise est donnée à la Section 3.1.2

2.4.3 La classification non-linéaire

La plupart des tâches de reconnaissance réelles sont fort complexes. C'est pourquoi nous avons besoin de classificateurs capables d'implémenter des surfaces de séparation plus complexes (i.e., non-linéaire).

Hyperplan séparateur dans l'espace de redescription

Dans ce mémoire nous nous concentrons principalement sur les méthodes à noyau pour la classification non-linéaire. Ces méthodes permettent d'utiliser les classificateurs linéaires et leurs algorithmes pour traiter des problèmes non-linéaires. L'approche générale est la suivante : la classification non-linéaire peut être résolue en transformant les données dans un espace de grande dimension $\mathcal{R}$, appelé *espace de redescription*, par

$$\begin{array}{rccc} \Phi : & \mathcal{X} & \longrightarrow & \mathcal{R} \\ & x & \mapsto & \Phi(x) \end{array}$$

et effectuer la classification linéaire (i.e. construire l'hyperplan) dans cet espace de redescription. En effet, il a été démontré qu'un ensemble de données d'apprentissage de n points représenté dans l'espace de redescription de dimension $d_{\mathcal{R}} > p$ peut toujours être linéairement séparé par un hyperplan [66]. Le classificateur résultant est donné par

$$f(x) = sign(\langle w, \Phi(x) \rangle + b), \tag{2.10}$$

2. Pour de plus amples informations le lecteur se reportera au Chapitre 3.

où $w \in F \subset \mathbb{R}^{d_F}$ et $\langle \cdot, \cdot \rangle$ représente le produit scalaire dans l'espace de redescription $\mathcal{R}$.

Exemple

Considérons les données représentées dans l'espace d'entrée, au schéma gauche de la Figure 2.3. Dans cet exemple, une surface de séparation non-linéaire (une ellipse) est requise pour classifier correctement toutes les données. Cependant, si nous utilisons une transformation non-linéaire

$$\begin{array}{rccc} \Phi : & \mathbb{R}^2 & \longrightarrow & \mathbb{R}^3 \\ & (x_1, x_2) & \mapsto & (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \end{array}$$

alors les données deviennent linéairement séparables dans l'espace de redescription.

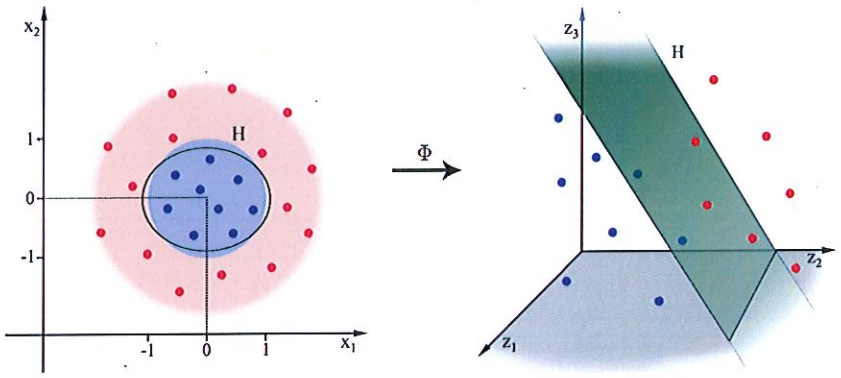


FIGURE 2.3 – Transformation $\Phi : (x_1, x_2) \rightarrow (z_1, z_2, z_3)$ de l'espace d'entrée (gauche) dans l'espace de redescription (droite) permettant une séparation linéaire des données.

Cependant, la construction et les performances de généralisation peuvent se dégrader lorsque la dimension de l'espace de redescription $\mathcal{R}$ croît. Ce phénomène est appelé malédiction de la dimension (*Curse of dimensionality*) [5, 8]. Par la suite, ce problème sera surmonté à l'aide de l'astuce du noyau qui est présentée à la Section 3.4.

2.4.4 Sur- et sous-apprentissage

Pour de petits ensembles d'apprentissage, minimiser le risque empirique peut conduire à un problème de sur-apprentissage. Par exemple, un classificateur apprenant par coeur toutes les données de l'ensemble d'apprentissage peut avoir un risque empirique proche de zéro mais ne pas être capable de généraliser [3], de

3. Un classificateur a un grand pouvoir de généralisation s'il fournit de bons résultats sur les données différentes de celles qui ont été utilisées pour l'apprentissage.

classifier de nouvelles données, c'est-à-dire ayant un risque fonctionnel élevé. Ce problème est dû à une complexité non-contrainte du modèle (voir Section 3.7 pour la définition de complexité d'un modèle). La complexité d'un modèle est lié à l'espace de fonction $\mathcal{F}$ que peut implémenter un algorithme d'apprentissage. Intuitivement une complexité faible (respectivement élevée) correspond à un ensemble $\mathcal{F}$ de petite (respectivement grande) dimension, par exemple uniquement des fonctions linéaires comme c'est le cas pour les classificateurs SVM linéaires (voir Section 3.2). Une complexité non-contrainte correspond donc à toutes les fonctions possibles et imaginables. C'est pourquoi si nous ne contraignons pas le modèle, il est très facile de construire un classificateur donnant un risque empirique nul, (n'importe quel modèle f telle que $f(x_i) = y_i$) mais qui ne minimise pas forcement le risque fonctionnel. Pour un exemple d'un tel classificateur, le lecteur peut se référer à la Section 3.4.

Pour illustrer ce problème, envisageons de classer les données d'un ensemble d'apprentissage $\mathcal{S}$ représentées, au cas 1 de la Figure 2.4, par les points bleus et rouges à l'aide d'un classificateur non-linéaire et non contraint et d'un classificateur linéaire. Les deux classificateurs minimisent leurs erreurs d'apprentissage. En fonction de la vraie distribution inconnue des données, le classificateur linéaire et non-linéaire peuvent être soit en situation de sous-apprentissage, soit en situation de sur-apprentissage. En effet, si les données sont réellement distribuées comme illustré au cas 2 alors le classificateur linéaire est en situation de sur-apprentissage car il classifierait les points de la zone rouge (respectivement bleu) comme faisant partie de la classe des points bleus (respectivement rouges). Si contrairement, les données sont réellement distribuées comme illustré au cas 3, alors le classificateur non-linéaire est clairement [4] en situation de sur-apprentissage. Il faut donc trouver un compromis entre adéquation aux données et complexité du modèle. Les SVM sont une des méthodes permettant de réaliser correctement ce compromis. De plus, une justification théorique de ce fait est donnée à la Section 3.7.

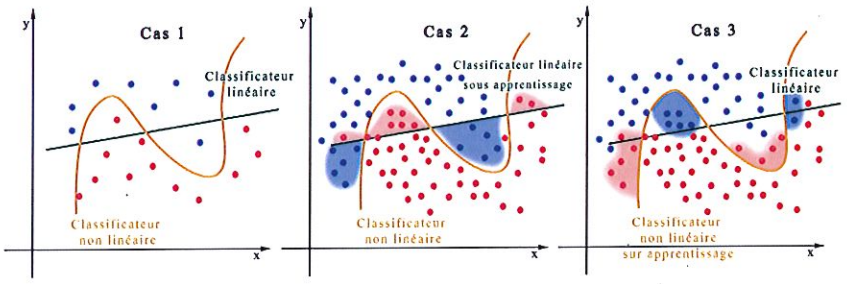


FIGURE 2.4 – Classificateurs linéaire et non-linéaire construits sur un ensemble d'apprentissage clairsemé (cas 1). En fonction de la vraie distribution des données, le classificateur linéaire peut être en situation de sur-apprentissage (cas 2) ou le classificateur non-linéaire peut être en situation de sur-apprentissage (cas 3) .

4. L'explication est similaire au cas 2.

Remarque

Si nous choisissons une classe de fonction $\mathcal{F}$ trop petite, il peut être impossible de trouver une bonne solution.

2.4.5 La classification multi-classes

Une approche générale pour les problèmes de classification multi-classes (i.e. le nombre de classes $m > 2$) consiste à construire un ensemble de classificateurs binaires, chacun d'eux étant assigné à une tâche particulière. Deux méthodes sont principalement utilisées :

- la méthode ***One-against-all***. Chaque classificateur binaire est construit pour séparer une classe de toutes les autres (i.e., attribuer le label 1 aux données de l'une des classes et le label -1 à toutes les autres données). Cette méthode requiert la construction de n classificateurs.

- la méthode ***One-against-one***. Chaque classificateur binaire est construit pour faire la distinction entre deux classes. Cette méthode requiert la construction de $\sum_{i=1}^{m-1} i = m(m-1)/2$ classificateurs binaires.

Il existe aussi plusieurs machines d'apprentissage capables de résoudre directement les problèmes multi-classes tels que les réseaux de neurones ou les machines à vecteurs de support multi-classes [21]. Le problème de la classification multi-classes n'est pas le but de ce mémoire, c'est pourquoi nous n'entrons pas dans les détails concernant celui-ci. Le lecteur peut se référer à [12].

2.5 La régression

La régression a pour but de trouver un modèle qui soit capable d'expliquer une variable dépendante à valeurs réelles à partir de variables explicatives. Il existe de nombreuses applications de la régression en mathématique, principalement en statistique. Le problème de la régression énoncé comme un problème d'approximation d'une fonction est donné comme suit

Définition 2.5.1 *(Le problème de la régression et vecteur de régression) Soit l'ensemble d'apprentissage*

$$\mathcal{S} = \{(x_1, y_1), ..., \ldots, (x_n, y_n)\}$$

où $x_i \in \mathbb{R}^p$ et $y_i \in \mathbb{R}$. Le problème de la régression a pour but de trouver une fonction $f : \mathcal{X} \to \mathcal{Y}$ et $f \in \mathcal{F}$ qui peut prédire la valeur de la sortie $y \in \mathbb{R}$ pour un vecteur d'entrée $x \in R^p$ par $y = f(x)$, sur base de l'ensemble d'apprentissage $\mathcal{S}$. Dans le cas de la régression, le vecteur d'entrée x est appelé vecteur de régression.

Dans ce contexte, la fonction de coût donnée par

$$l(f(x), y) = (y - f(x))^2, \tag{2.11}$$

est souvent utilisée dans le risque (2.1), conduisant à l'erreur quadratique moyenne (MSE pour Mean Square Error) bien connue. Dans ce mémoire, la notation MSE fera référence au test d'erreur réalisé sur un ensemble test $\hat{S}$, comme pour (2.3) et donné par

$$\text{MSE} = \frac{1}{|\hat{S}|}\sum_{i=1}^{|\hat{S}|}(y_i - f(x_i))^2. \tag{2.12}$$

Remarque

Afin de vérifier la qualité de nos modèles, nous utiliserons l'erreur RMSE qui est définie comme la racine carrée de l'erreur MSE. Pour de plus amples informations, le lecteur se référera au Chapitre 6.

2.5.1 La régression linéaire

Le but de la régression linéaire est de construire un modèle $f \in \mathcal{F}$ qui soit linéaire par rapport aux vecteurs d'entrées de l'ensemble d'apprentissage. Dans le cas non bruité, ce problème peut être énoncé comme le problème admissible suivant

$$\text{trouver } w \text{ tel que } y_i = f(x_i) = \langle w, x_i \rangle, \quad \forall i : 1 \leq i \leq n, \tag{2.13}$$

où le vecteur w contient les paramètres du modèle. Cependant, dans la plupart des cas, les données sont bruitées et le problème devient alors un problème de minimisation du risque empirique (2.2) ou de manière équivalente, un problème de minimisation de la norme des résidus $||\mathbf{y} - Xw||_2$ où X est une matrice dont les colonnes sont les vecteurs d'entrées x_i $(i = 1, ..., n)$ et $\mathbf{y}$ est un vecteur dont les composantes sont les sorties y_i $(i = 1, ..., n)$, comme représenté à la Figure 2.5. Si la norme des résidus est la norme Euclidienne, correspondant à une fonction coût quadratique pour (2.1), alors l'estimateur bien connu qui minimise la somme des carrés des résidus est donné par

$$w = (X^T X)^{-1} X^T y. \tag{2.14}$$

Remarques

1. Pour de plus amples informations sur la régression linéaire le lecteur pourra se référer à [18].

2. Différents estimateurs peuvent être définis en choisissant différentes fonctions de coût pour (2.1) ou encore en choisissant d'autres normes pour les résidus. Par exemple, un autre estimateur bien connu est obtenu pour la fonction de coût $l(f(x), y) = |y - f(x)|$, correspondant à l'utilisation de la norme l_1 pour les résidus.

2.5.2 La régression non-linéaire

L'approche généralement suivie pour l'approximation d'une fonction non-linéaire est de construire un modèle linéaire sur base des données transformées, $\Phi(x)$. En fonction des différentes transformations $\Phi : x \rightarrow \Phi(x)$, différentes classes de modèles peuvent être définies.

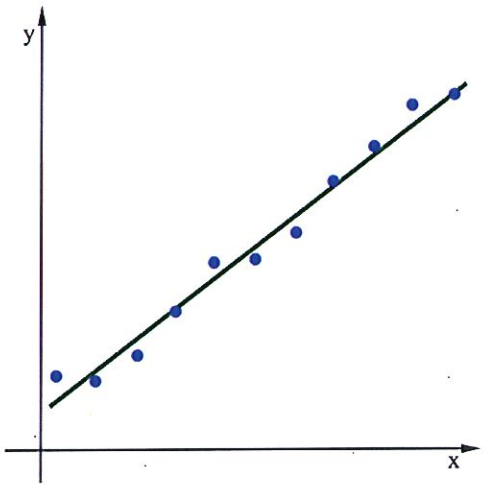


FIGURE 2.5 – Régression linéaire à partir de données bruitées.

2.5.2.1 Les modèles paramétriques

Les modèles paramétriques ont une structure fixée (i.e., le modèle) et un nombre de paramètres qui sont déterminés sur base des connaissances a priori. Citons par exemple en statistique, une distribution normale comme modèle ayant comme paramètres la moyenne et la variance ou encore la régression linéaire multiple. Nous n'entrons pas plus dans les détails de cette classe de modèles. Cependant, le lecteur peut se référer à [46, 54] pour de plus amples informations.

2.5.2.2 Les modèles non-paramétriques

Les modèles non-paramétriques font référence aux modèles dont la structure n'est pas définie a priori mais est au contraire estimée à partir des données. Dans ce cas, le nombre de paramètres n'est bien sûr pas fixé à l'avance.

Modèles polynomiaux

Les modèles polynomiaux construisent une fonction f définie par un polynôme de dimension p et de degré d

$$f(x) = w_0 + \sum_{k=1}^{p} w_k [x]_k + \sum_{k_1=1}^{p} \sum_{k_2=1}^{p} w_{k_1 k_2} [x]_1 [x]_2 + \ldots + \sum_{k_1=1}^{p} \ldots \sum_{k_d=1}^{p} w_{k_1 \ldots k_d} [x]_1 \ldots [x]_d, \tag{2.15}$$

où $[x]_k$ représente la $k^{\text{ième}}$ composante du vecteur de régression x. Ces modèles, dépendant du degré d, peuvent être utilisés pour approximer une très grande classe de fonctions non-linéaires comme illustré à la Figure 2.6. Malheureusement le nombre M de paramètres w augmente rapidement avec la taille de x et avec

le degré d du polynôme

$$M = \frac{(d+p)!}{d!p!}.$$

Ce nombre implique clairement que les modèles polynomiaux souffrent donc de la malédiction des grandes dimensions. Finalement, le modèle (2.15) peut être reformulé comme un modèle linéaire

$$f(x) = \tilde{f}(\tilde{x}) = \theta_0 + \sum_{i=1}^{M} \theta_k \tilde{x}^k$$

où $\theta_k \tilde{x}^k$ correspond au $k^{\text{ième}}$ terme de (2.15) avec un changement de variable de x vers $\tilde{x}$. Cette forme paramétrisée peut être estimée par les méthodes de régression linéaire standards.

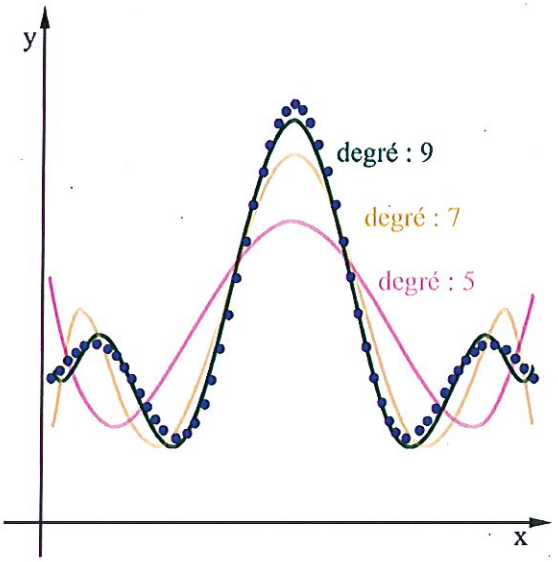


FIGURE 2.6 – Régression non-linéaire réalisée à l'aide de modèles polynomiaux de différents degrés.

Réseaux de fonctions à base radiale

Les réseaux de fonctions à base radiale (RBF) sont un des modèles connexionnistes[5] les plus connus et simples à mettre en oeuvre. Ils sont basés sur une expansion de

$$f(x) = \sum_{k=1}^{n} \alpha_k f_k(x), \tag{2.16}$$

5. "Discipline concernant les techniques de simulation des processus intelligents par des réseaux de neurones et des ordinateurs neuronaux, l'intelligence étant alors censée se trouver dans l'agencement des connexions et non pas directement dans une suite de calculs (i.e., séries d'opérations algébriques)" [http ://jargonf.org/wiki/connexionnisme].

où n est le nombre de données de l'ensemble d'apprentissage. Les *fonctions de base* $f_k(x)$ peuvent être différentes pour chaque k. Généralement, ces fonctions de base sont construites en faisant varier un ensemble de paramètres (σ, γ). La fonction de base probablement la plus connue est la fonction de base radiale (RBF) qui est une fonction ϕ symétrique autour d'un centre γ_k et est donnée par $f_k(x) = \phi(||x-\gamma_k||, \sigma_k)$, où $||\cdot||$ est une norme et σ_k représente la largeur de ces fonctions à base radiale ϕ. Un modèle ou réseau RBF calcule une combinaison linéaire de fonctions radiales

$$f(x) = \sum_{k=1}^{n} w_k \ \phi(||x - \gamma_k||, \sigma_k).$$

Le but de l'approximation est alors de trouver les poids w_k qui minimisent l'erreur entre les réponses réelles y et celles prédites par le modèle $f(x)$. Cenaero utilise principalement deux types de fonction de base : la fonction Gaussienne et la fonction multiquadratique. Les expressions de ces deux fonctions de base sont données dans le tableau ci-dessous et sont illustrées à la Figure 2.7.

	Expression de $\phi(\|\|x - \gamma_k\|\|, \sigma_k)$
Gaussienne	$e^{\left(-\frac{\|\|x-\gamma_k\|\|^2}{2\sigma_k^2}\right)}$
Multiquadratique	$\sqrt{\frac{\|\|x-\gamma_k\|\|^2}{\sigma_k^2} + 1}$

Tableau 2.1 – Expressions des fonctions de base d'un modèle RBF.

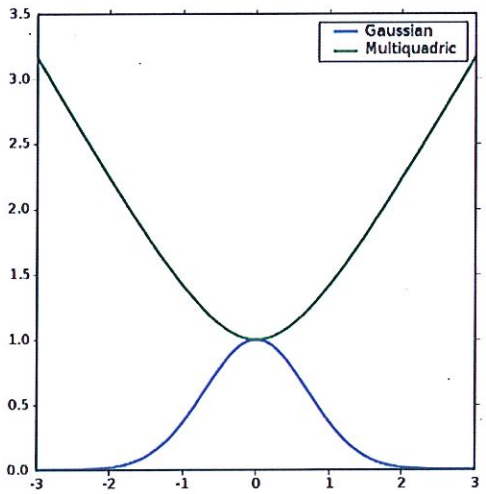


FIGURE 2.7 – Représentation des fonctions à base radiale Gaussienne et multiquadratique.

Finalement, le fonctionnement général des réseaux de fonctions à base radiale peut être divisé en trois étapes décrites par l'algorithme suivant

Algorithme 2 RBF

1: Choisir une fonction de base.
2: Déterminer le nombre et les paramètres des fonctions de bases.
3: Calculer la valeur optimale des poids w_k de (2.16)

Pour de plus amples informations concernant les réseaux de fonctions à base radiale le lecteur peut se référer à [10].

2.5.3 Sur-apprentissage et sous-apprentissage

Pour la régression non-linéaire, nous pouvons aussi être confronté au problème du sur-apprentissage. La raison est que nous pouvons facilement trouver un modèle minimisant le risque empirique mais pour lequel le risque fonctionnel est très grand. Un exemple simple pour la régression non-linéaire est un modèle polynomial pour des données bruitées : plus le degré du polynôme sera grand, plus le risque empirique sera faible, mais plus le risque fonctionnel sera élevé comme cela est illustré à la Figure 2.8. Dans cet exemple, nous voyons bien que le risque empirique décroît et que le risque fonctionnel croît lorsque le degré du polynôme augmente. De même, nous pouvons aussi être confronté au problème de sous-apprentissage. Dans l'exemple de la Figure 2.8, nous remarquons clairement que l'approximation de $sinc(x)$ par un polynome de faible degré (courbe verte) est en situation de sous-apprentissage et a un risque fonctionnel et empirique élevé.

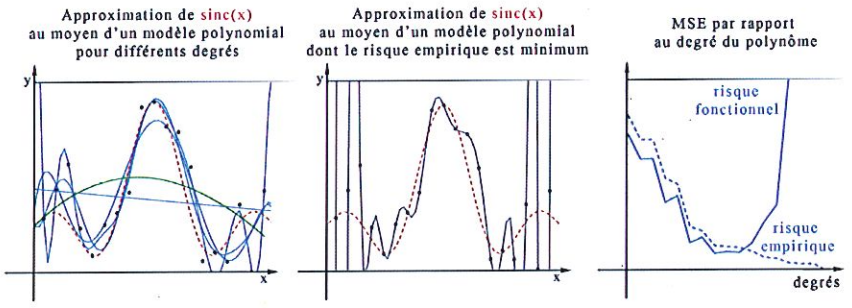


FIGURE 2.8 – A gauche : approximation de la fonction $sinc(x)$ par un modèle polynomial de différents degrés à partir de données bruitées. Au milieu : le modèle polynomial correspondant au risque empirique minimum. A droite : le risque fonctionnel et empirique par rapport au degré du modèle polynomial.

Chapitre 3

Machines à vecteurs de support pour la classification

RÉSUMÉ. *Dans ce chapitre, nous décrivons le principe général d'une méthode de classification particulière qui fut introduite par Vapnik [66] en 1995 : les machines à vecteurs de support (SVM pour Support Vector Machines). Grâce au lien direct entre la théorie de l'apprentissage statistique et l'algorithme d'apprentissage des SVM, qui constituent de solides bases théoriques qui sont développées à la fin de ce chapitre, la méthode des SVM obtient un succès non seulement mérité mais aussi justifié. La méthode liée aux SVM étant assez complexe d'un point de vue mathématique, nous avons divisé ce chapitre en plusieurs parties. Nous introduisons dans un premier temps, de façon simple et complète, les différentes notions nécessaires à la compréhension du principe général de fonctionnement des SVM. Nous présentons ensuite, en détail, l'aspect purement mathématique des SVM pour la classification. En particulier, nous donnons les différentes formulations des problèmes quadratiques linéaires convexes correspondant aux différentes situations des SVM. Finalement, nous introduisons les méthodes à noyau. Ce chapitre se base principalement sur [12].*

3.1 Classificateur : notions de base

Dans cette section, nous nous intéressons aux SVM pour la classification qui permettent de traiter le problème de la classification défini à la Section 2.4. Soit l'ensemble d'apprentissage,

$$\mathcal{S} = \{(x_i, y_i) \text{ tels que } 1 \leq i \leq n,\ x_i \in \mathbb{R}^p \text{ et } y_i \in \{-1, +1\}\},$$

les SVM pour la classification ont pour but de déterminer un classificateur SVM f permettant d'assigner à chaque donnée x_i (respectivement à une nouvelle donnée $x \notin \mathcal{S}$) sa classe y_i (respectivement y). Sans perte de généralité, nous ne considérons dans ce mémoire que les problèmes binaires (i.e., les problèmes à deux classes). Afin de décrire le principe de construction d'un classificateur linéaire et non-linéaire, nous introduisons pour commencer différents concepts importants relatifs aux SVM.

3.1.1 Classificateur linéaire

Pour commencer, nous rappelons la définition d'une fonction de décision

$$h : \mathbb{R}^p \to \mathbb{R}.$$

Définition 3.1.1 *(Fonction de décision et règle de décision)*
Une fonction de décision $f : \mathbb{R}^p \to \mathbb{R}$ pour un problème de classification binaire définit une règle de décision du type

$$\text{si } h(x) \begin{cases} > 0 \\ < 0 \end{cases} \quad \text{alors} \quad y = \begin{cases} +1 & \text{(i.e., } x \in \text{classe 1)}, \\ -1 & \text{(i.e., } x \in \text{classe 2)}. \end{cases}$$

Un classificateur est dit linéaire lorsqu'il est possible d'exprimer sa fonction de décision par une fonction linéaire en x. Dans le cas des SVM, cette fonction est obtenue par combinaison linéaire des composantes du vecteur d'entrée $x \in \mathcal{X}$. Nous supposons à présent que l'espace d'entrée $\mathcal{X} = \mathbb{R}^p$ où p est le nombre de composantes des vecteurs d'entrées.

$$h(x) \stackrel{déf}{=} \langle w, x \rangle + b = \sum_{i=1}^{p} w_i [x]_i + b,$$

où :

- $w \in \mathbb{R}^p$, $b \in \mathbb{R}$ sont les paramètres à calculer pour déterminer $f(x)$ et $x \in \mathbb{R}^p$;
- $\langle w, x \rangle$ représente le produit scalaire entre w et x.
- $[x]_i$ représente la $i^{\text{ème}}$ composante du vecteur x.

Pour décider à quelle classe une donnée x appartient, il suffit de prendre le signe de la fonction de décision. Le classificateur f est alors donné par

$$y = f(x) = sign(h(x)) = \begin{cases} 1 & \text{si } f(x) > 0, \\ -1 & \text{si } f(x) < 0. \end{cases}$$

Géométriquement, cela revient à considérer un hyperplan [1] qui est le lieu des points x satisfaisant à

$$h(x) = \langle w, x \rangle + b = 0.$$

En fixant un côté pour lequel les données sont classées positivement, la règle de décision correspond à observer de quel côté de l'hyperplan se trouve la donnée x. Nous décidons alors que x est de classe 1 si $h(x) > 0$ et de classe -1 si $h(x) < 0$.

Remarque

Le vecteur w définit la pente de l'hyperplan (i.e., la normale) et est perpendiculaire à l'hyperplan. La constante b permet quant à elle de translater l'hyperplan parallèlement à lui-même. Une représentation graphique de ces deux paramètres est donnée à la Figure 3.1.

1. Un hyperplan est un sous-espace vectoriel particulier. Par la suite, nous utilisons indifféremment hyperplan et frontière de décision.

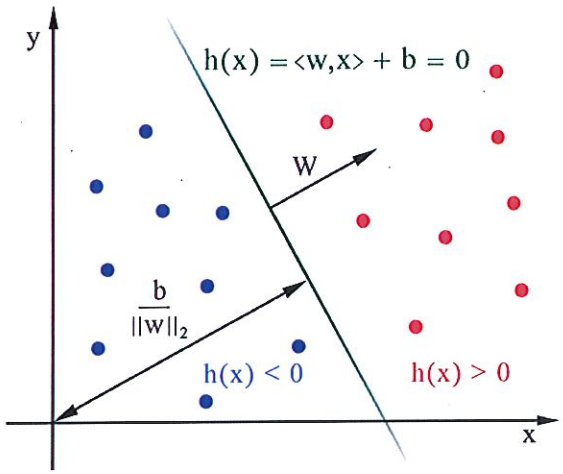


FIGURE 3.1 – Représentation graphique dans $\mathbb{R}^2$ des paramètres w et b de l'hyperplan correspondant à la fonction de décision d'un classificateur linéaire.

3.1.2 Marge de l'hyperplan

Avant d'introduire les différentes notions de marge, il est nécessaire de définir la notion d'ensemble d'apprentissage linéairement séparable.

Définition 3.1.2 *(Ensemble d'apprentissage linéairement séparable)*
Un ensemble d'apprentissage

$$\mathcal{S} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

est dit linéairement séparable par l'hyperplan

$$h(x) = \langle w, x \rangle + b = 0$$

si et seulement si

$$\exists\ w \in \mathbb{R}^p,\ b \in \mathbb{R} \text{ tels que } y_i(\langle w, x_i \rangle + b) \geq 0 \quad \forall i : 1 \leq i \leq n. \tag{3.1}$$

L'hyperplan défini par w et b est encore alors appelé l'hyperplan séparateur.

En d'autres mots, un ensemble d'apprentissage est dit linéairement séparable s'il existe un hyperplan qui sépare les données positives en les laissant d'un côté de l'hyperplan et les données négatives de l'autre côté de l'hyperplan. Dans le cas d'ensemble d'apprentissage linéairement séparable il existe de nombreuses méthodes qui permettent de trouver un tel hyperplan. La plus connue d'entre elles est l'algorithme du Perceptron proposé par Frank Rosenblatt en 1956 [16].

Nous pouvons maintenant introduire la notion de marge. Celle-ci peut être relative à une donnée particulière ou à l'ensemble d'apprentissage. De plus, Vapnik a introduit deux types de marge, à savoir la marge fonctionnelle et la marge géométrique [66].

Définition 3.1.3 *(Marge fonctionnelle)*
La marge fonctionnelle d'une donnée x_i, par rapport à l'hyperplan caractérisé par w et b, est donnée par

$$y_i(\langle w, x_i \rangle + b).$$

La marge géométrique représente quant à elle la distance euclidienne prise perpendiculairement entre l'hyperplan et une donnée x_i. Définissons pour commencer la distance euclidienne d'une donnée à l'hyperplan. La Figure 3.2 illustre ce concept.

Définition 3.1.4 *(Distance euclidienne d'une donnée)*
La distance euclidienne d'une donnée x_i par rapport à l'hyperplan caractérisé par w et b est donnée par sa projection orthogonale sur l'hyperplan

$$d(w, b; x_i) = \frac{|h(x_i)|}{||w||_2} = \frac{|\langle w, x \rangle + b|}{||w||_2}$$

où $||\cdot||_2$ est la norme euclidienne.

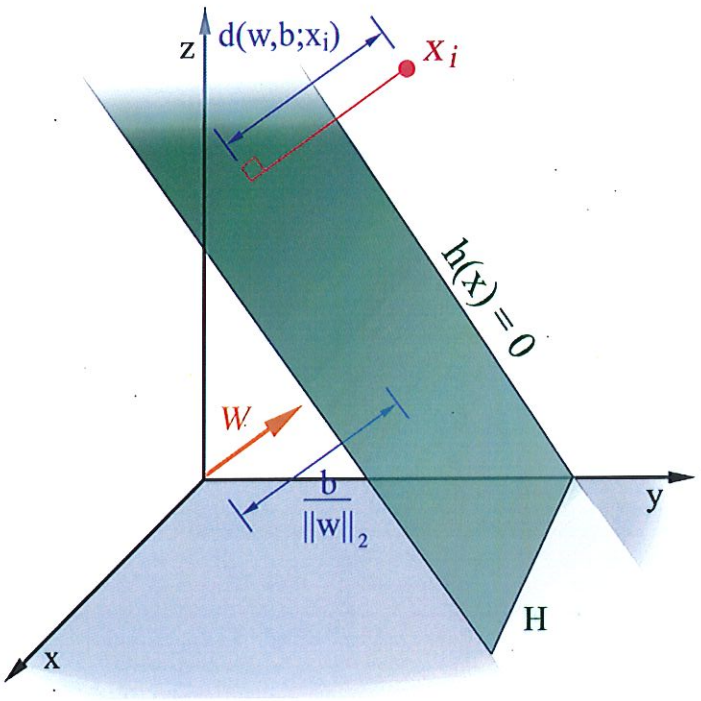


FIGURE 3.2 – Représentation de la distance euclidienne d'une donnée x.

Nous pouvons maintenant définir la marge géométrique.

Définition 3.1.5 *(Marge géométrique de l'ensemble d'apprentissage)*
La marge géométrique de l'ensemble d'apprentissage par rapport à l'hyperplan caractérisé par w et b est définie comme étant

$$\rho(w,b) = \min_{\{x_i : y_i = 1\}} d(w,b;x_i) + \min_{\{x_j : y_j = -1\}} d(w,b;x_j).$$

Par la suite, nous utilisons uniquement la marge géométrique de l'ensemble d'apprentissage, c'est pourquoi nous employons le terme *marge* pour désigner cette dernière. Afin de pouvoir correctement illustrer la marge nous avons besoin d'une notion supplémentaire, à savoir les hyperplans canoniques décrits à la Section 3.1.3. Une représentation graphique de la marge est donnée dans cette section à la Figure 3.3.

Les classificateurs (i.e., les algorithmes d'apprentissage) ayant pour critère de maximiser la marge sont appelés *classificateurs à marge maximale.* En particulier, **SVM est un classificateur à marge maximale.**

3.1.3 Hyperplans canoniques

Nous supposons à partir de maintenant que nous travaillons avec un ensemble d'apprentissage linéairement séparable. Dans le cas de classificateurs à marge maximale, l'hyperplan séparateur défini par les paramètres w et b correspond à la médiatrice du plus petit segment de droite reliant les enveloppes convexes des deux classes. La Figure 3.3 illustre l'hyperplan séparateur en vert ainsi que les deux enveloppes convexes englobant respectivement tous les points bleus et rouges qui représentent les deux catégories (i.e., classes) de données à séparer appartenant à l'ensemble d'apprentissage.

Afin de faciliter l'optimisation (i.e., la recherche des paramètres w et b), nous allons définir deux hyperplans se trouvant de part et d'autre de l'hyperplan séparateur et parallèles à celui-ci, sur lesquels reposent les données les plus proches. Comme le couple de paramètres (w,b), qui caractérise l'hyperplan séparateur, est défini à un coefficient multiplicatif près, l'hyperplan est inchangé s'il est multiplié par une constante arbitraire, il est donc possible que différentes équations correspondent au même plan géométrique

$$\{x : \langle w,x\rangle + b = 0\} \Leftrightarrow \{x : c(\langle w,x\rangle + b) = 0\}, \quad \forall c \in \mathbb{R}_0.$$

Nous allons donc normaliser w et b. Nous prenons pour cela le couple de paramètres $\left(\frac{w}{\|w\|_2}, \frac{b}{\|w\|_2}\right)$, de telle manière que les deux hyperplans parallèles aient respectivement pour équation

$$\langle w,x\rangle + b = 1 \quad \text{et} \quad \langle w,x\rangle + b = -1. \tag{3.2}$$

Ces deux hyperplans sont appelés *hyperplans canoniques* et cette normalisation est appelée *forme canonique de l'hyperplan.* Nous pouvons maintenant caractériser ces données particulières qui se trouvent sur les hyperplans canoniques.

Définition 3.1.6 *(Vecteur de support)*
Une donnée x_i qui se trouve sur un hyperplan canonique, c'est-à-dire qui vérifie

$$\langle w,x_i\rangle + b = 1 \quad \textit{ou} \quad \langle w,x_i\rangle + b = -1$$

est appelée vecteur de support.

Remarques

1. Après normalisation, les données satisfont à :

$$y_i h(x_i) = y_i(\langle w, x_i \rangle + b) \geq 1 \quad \forall i : 1 \leq i \leq n. \tag{3.3}$$

2. La marge des hyperplans canoniques est de $\frac{1}{||w||_2}$. En effet, sans perte de généralité, soient $(x_i, 1)$ et $(x_j, -1)$, deux vecteurs de support parallèles. Nous avons alors

$$\langle w, x_i \rangle + b = 1 \quad \text{et} \quad \langle w, x_j \rangle + b = -1$$

et

$$(\langle w, x_i \rangle + b) - (\langle w, x_j \rangle + b) = \langle w, x_i - x_j \rangle = 2.$$

D'où, par les propriétés du produit scalaire, nous avons que

$$\begin{aligned} \langle w, x_i - x_j \rangle &= ||w||_2 ||x_i - x_j||_2 \cos 0° \\ &= ||w||_2 \underbrace{||x_i - x_j||_2}_{\rho(w,b)} \\ &= 2. \end{aligned}$$

Nous en déduisons que la marge $\rho(w, b) = \frac{2}{||w||_2}$ et, par définition, que la marge des hyperplans canoniques est bien de $\frac{1}{||w||_2}$.

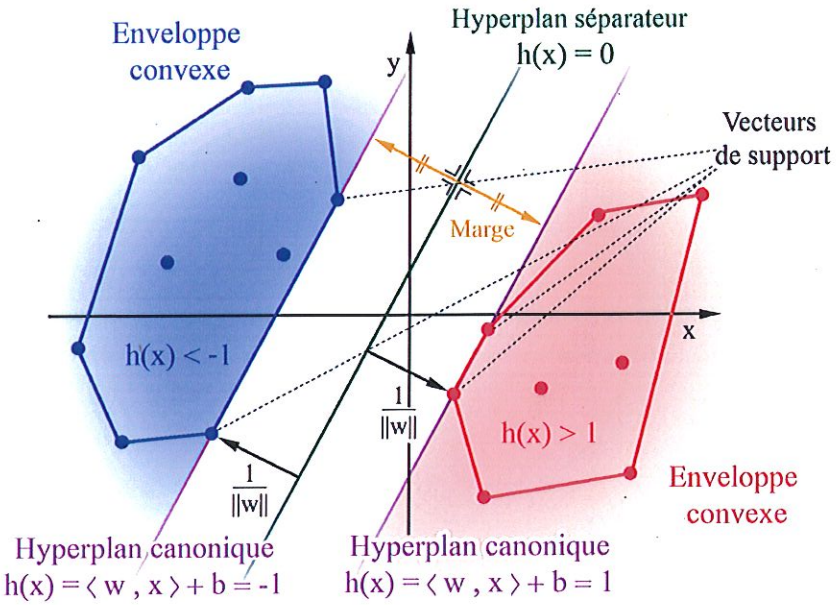


FIGURE 3.3 – Représentation des hyperplans canoniques.

3.1.4 Motivation : pourquoi maximiser la marge ?

Lorsque l'ensemble d'apprentissage est linéairement séparable, il existe de nombreuses possibilités de classificateurs linéaires qui peuvent séparer les données (Figure 3.4). La question est alors de savoir comment sélectionner le meilleur hyperplan (i.e., commettant le moins d'erreurs lors de la séparation des données en deux classes et ayant la meilleure capacité de généralisation). Nous devons pour cela introduire un critère de sélection [2].

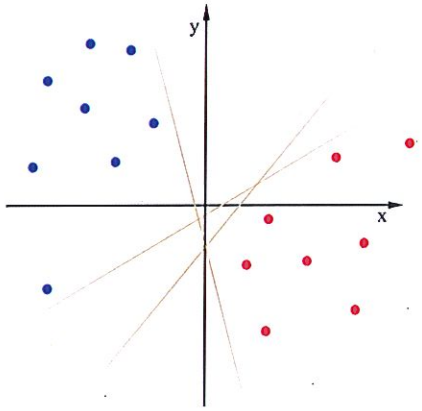


FIGURE 3.4 – Différents classificateurs pour deux classes de données.

Comme définie précédemment (voir Définition 3.1.1), la classification d'une nouvelle donnée inconnue est établie par sa position par rapport à l'hyperplan séparateur. Dans l'exemple suivant (Figure 3.5), la nouvelle donnée symbolisée par un point orange sera classée dans la classe des données représentées par des points bleus.

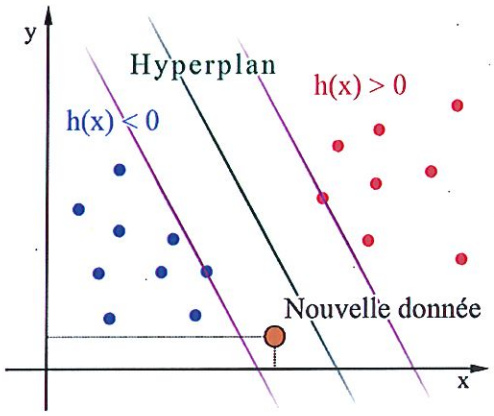


FIGURE 3.5 – Classification d'une nouvelle donnée.

2. Dans notre cas ce critère est le fait de maximiser la marge.

Intuitivement donc, le fait de maximiser la marge permet une classification plus sûre d'une nouvelle donnée. En d'autres mots, le fait de trouver l'hyperplan séparateur qui a la marge la plus grande par rapport aux données de l'ensemble d'apprentissage permet de classer de nouvelles données plus fiablement. Cet hyperplan est appelé *hyperplan séparateur optimal* et est donc défini par (w, b) tel que

$$(w, b) = \arg\max_{w', b'} \rho(w', b').$$

Les graphes suivants (Figure 3.6) illustrent le fait qu'avec un hyperplan séparateur optimal, une nouvelle donnée x_i rouge (i.e., $h(x_i) > 0$) est bien classée. A contrario, (voir le graphe de droite de la Figure 3.6) avec un hyperplan quelconque ne commettant pas d'erreurs sur l'ensemble d'apprentissage et ayant une marge plus petite, cette même donnée, qui devrait normalement être classée dans l'ensemble des points rouges, se voit mal classée dans l'ensemble des points bleus. De plus, avec l'hyperplan séparateur optimal défini ci-dessus, si une donnée n'a pas été décrite parfaitement, une petite perturbation ne modifiera pas sa classification car sa distance à l'hyperplan est maximale.

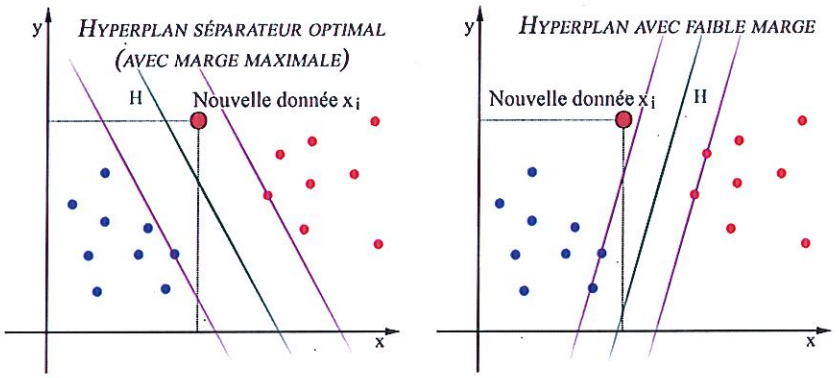


FIGURE 3.6 – Classement de nouvelles données en fonction de l'hyperplan.

Maximiser la marge permet donc à un algorithme d'apprentissage de contrôler la complexité (ou capacité) du modèle (i.e., de l'hyperplan). Cette intuition (i.e., la capacité de généralisation dépend de la distance entre l'hyperplan et les données de l'ensemble d'apprentissage) est plus précisément exprimée par Vapnik [66] et est étudiée en détail dans la Section 3.7.

3.2 Classificateur SVM linéaire

Les notions de marge, d'hyperplans canoniques et de vecteurs de support étant maintenant définies, nous pouvons formuler le problème d'optimisation portant sur les paramètres w et b telle que sa solution nous fournisse l'hyperplan optimal (i.e., l'hyperplan maximisant la marge). Soit

$$\mathcal{S} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

où $x_i \in \mathbb{R}^p$ et $y_i \in \{-1, +1\}$, l'hyperplan optimal est la solution du problème d'optimisation

$$\begin{cases} \min & \frac{1}{2}||w||_2^2 \\ \text{s.c.} & y_i(\langle w, x_i \rangle + b) \geq 1 \quad \forall i : 1 \leq i \leq n, \end{cases}$$

qui est encore équivalent à

$$\begin{cases} \min & \frac{1}{2}||w||_2^2 \\ \text{s.c.} & y_i(\langle w, x_i \rangle + b) - 1 \geq 0 \quad \forall i : 1 \leq i \leq n. \end{cases} \tag{3.4}$$

Il s'agit d'un problème quadratique convexe dont la fonction objectif est à minimiser. Cette fonction objectif est obtenue par le fait que maximiser la marge revient en fait à minimiser le carré de l'inverse de celle-ci. L'unique contrainte stipule que les données doivent être bien classées et qu'elles ne dépassent pas les hyperplans canoniques.

Remarques

1. Il est souvent plus aisé de minimiser $||w||_2^2$ que $||w||_2$. En effet, pour les méthodes d'optimisation basées sur les dérivées il est préférable de minimiser la norme au carré plutôt que de minimiser simplement la norme pour des raisons de différentiabilité.

2. Cette écriture du problème est appelée *formulation primale.*

Dans le problème d'optimisation (3.4), les variables à déterminer sont donc les composantes $[w]_i$ de w et la constante b. D'un point de vue ML, ces variables correspondent aux composantes [3] $[\alpha]_i$ de la variable $\alpha \in \Lambda$ de la machine d'apprentissage (voir Section 2.1). Le vecteur w est de dimension égale à la dimension de l'espace d'entrée, p. Si nous gardons cette formulation primale du problème, nous souffrons malheureusement des inconvénients classiques des méthodes du ML, à savoir principalement les problèmes de sur-apprentissage et de la malédiction de la dimension. Afin de contourner ces problèmes, il est nécessaire d'introduire la *formulation duale* de ce problème d'optimisation. Rappelons que le problème dual est un problème d'optimisation fournissant la même solution que le problème primal mais dont la formulation est différente. Les variables du problème primal sont appelées *variables primales* et les variables du problème dual, *variables duales.*

Afin d'introduire le problème dual du problème d'optimisation (3.4), nous rappelons pour commencer *les conditions d'optimalité de Karush-Kuhn-Tucker* (KKT) à l'aide du théorème suivant.

3. A ne pas confondre avec les multiplicateurs de Lagrange.

Théorème 3.2.1 *(Conditions d'optimalité de KKT pour les problèmes différentiables convexes)*
Considérons un problème d'optimisation de la forme

$$\begin{cases} min & f(x) \\ s.c. & g_i(x) \geq 0 \quad \forall i = 1, \ldots, n_g \\ & h_j(x) = 0 \quad \forall j = 1, \ldots, n_h \end{cases}$$

avec $f(\cdot)$, $g_i(\cdot)$, $h_j(\cdot)$ convexes et différentiables.

Le Lagrangien est formé comme suit

$$L(x, \alpha, \beta) = f(x) - \sum_{i=1}^{n_g} \alpha_i g_i(x) + \sum_{j=1}^{n_h} \beta_j h_j(x).$$

La solution $\overline{x}$ est optimale si et seulement si il existe $\overline{\alpha} \in \mathbb{R}^{n_g}$ avec $\bar{\alpha}_i \geq 0$ pour tout $i = 1, \ldots, n_g$ et $\overline{\beta} \in \mathbb{R}^{n_h}$ avec $\bar{\beta}_j$ sans restriction de signe (s.r.s) pour tout $j = 1, \ldots, n_h$ tels que

$$\begin{aligned} \frac{\partial L(\overline{x}, \overline{\alpha}, \overline{\beta})}{\partial x} &= \frac{\partial f(\bar{x})}{\partial x} - \sum_{i=1}^{n_g} \bar{\alpha}_i \frac{\partial g_i(\bar{x})}{\partial x} + \sum_{j=1}^{n_h} \bar{\beta}_j \frac{\partial h_j(\bar{x})}{\partial x} = 0 \\ g_i(\bar{x}) &\geq 0 \quad \forall i = 1, \ldots, n_g \\ h_j(\bar{x}) &= 0 \quad \forall j = 1, \ldots, n_h \\ \overline{\alpha_i} g_i(\bar{x}) &= 0 \quad \forall i = 1, \ldots, n_g \quad \textit{condition de complémentarité} \\ \overline{\beta_j} h_j(\bar{x}) &= 0 \quad \forall j = 1, \ldots, n_h \quad \textit{condition de complémentarité.} \end{aligned}$$

Ce théorème fondamental d'optimisation fournit une condition nécessaire et suffisante pour l'optimalité d'une solution dans le cadre de problèmes différentiables convexes [51].

Le problème dual du problème d'optimisation (3.4) est donné grâce à la méthode des *multiplicateurs de Lagrange*, où le *Lagrangien* est la somme de la fonction objectif et d'une combinaison linéaire des contraintes pondérées chacune par une variable duale (voir Théorème 3.2.1)

$$L(w, b; \alpha) = \frac{1}{2}||w||_2^2 - \sum_{i=1}^{n} \alpha_i \underbrace{(y_i(\langle w, x_i \rangle + b) - 1)}_{\text{contrainte}}, \tag{3.5}$$

où les α_i sont les multiplicateurs de Lagrange ou variables duales.

Le problème primal et sa formulation duale ont la même solution qui correspond à un point selle du Lagrangien. Le Lagrangien doit donc être minimisé par rapport aux variables primales w, b et maximisé par rapport aux variables duales α_i. Dès lors, nous pouvons passer du problème primal au problème dual. Celui-ci est donné par

$$\max_{\alpha} \left(\min_{w,b} L(w, b; \alpha) \right).$$

Remarque

Nous pouvons donner une signification physique aux multiplicateurs de Lagrange. La variable α_i représente la "force" avec laquelle la solution appuie sur la contrainte i du problème d'optimisation (3.4). Ainsi, un hyperplan qui violerait la contrainte pour x_i (i.e., il classe cette donnée du mauvais côté) rendrait α_i très grand ce qui ferait fortement augmenter la fonction objectif $L(w, b; \alpha)$. Cette solution n'est donc pas à retenir comme solution optimale.

Nous allons à présent chercher un point selle. Déterminons pour cela les conditions de KKT du problème d'optimisation (3.4). Les dérivées partielles du Lagrangien par rapport aux variables primales doivent pour cela s'annuler. Dès lors, nous obtenons

$$\frac{\partial L(w,b;\alpha)}{\partial b} = 0 \quad \text{et} \quad \frac{\partial L(w,b;\alpha)}{\partial w} = 0,$$

ce qui implique par (3.5) que

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{et} \quad \sum_{i=1}^{n} \alpha_i x_i y_i = w. \tag{3.6}$$

Avec cette formulation, nous pouvons calculer w en déterminant uniquement les n paramètres α_i. L'idée sera donc de formuler un problème dual dans lequel w est remplacé par sa formulation $w = \sum_{i=1}^{n} \alpha_i x_i y_i$. De cette façon, le nombre de paramètres à fixer est relatif au nombre de données de l'ensemble d'apprentissage et non plus à la dimension de l'espace d'entrée. Pour ce faire, nous substituons les égalités (3.6) dans le Lagrangien (3.5)

$$\begin{aligned} L(w,b;\alpha) &= \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{n} \alpha_i (y_i(\langle w, x_i\rangle + b) - 1) \\ &= \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle - \sum_{i,j=1}^{n} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle - \underbrace{\sum_{i=1}^{n} \alpha_i y_i b}_{=0 \text{ par } (3.6)} + \sum_{i=1}^{n} \alpha_i \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle. \end{aligned}$$

Nous obtenons finalement la forme duale du problème d'optimisation (3.4)

$$\begin{cases} \max\limits_{\alpha} & \sum\limits_{i=1}^{n} \alpha_i - \frac{1}{2}\sum\limits_{i,j=1}^{n} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle \\ \text{s.c.} & \alpha_i \geq 0 \quad \forall i : 1 \leq i \leq n \\ & \sum\limits_{i=1}^{n} \alpha_i y_i = 0. \end{cases} \tag{3.7}$$

Remarque

Soit le Lagrangien

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

Selon les conditions de complémentarité de Karush-Kuhn-Tucker, la solution w est telle que

$$\alpha_i \underbrace{(y_i(\langle w, x_i \rangle + b) - 1)}_{\text{contrainte}} = 0, \quad \forall i : 1 \leq i \leq n,$$

d'où, par la complémentarité stricte,

- $\alpha_i = 0$ si la contrainte est satisfaite en tant qu'inégalité stricte

$$y_i(\langle w, x_i \rangle + b) > 1 \quad \text{par (3.3)},$$

- $\alpha_i > 0$ si la contrainte est satisfaite en tant qu'égalité

$$y_i(\langle w, x_i \rangle + b) = 1.$$

Dans ce cas, x_i est un vecteur de support.

Le vecteur solution w admet une formulation, donnée par (3.6), qui dépend des données x_i de l'ensemble d'apprentissage. Plus précisément, cette formulation va en fait dépendre uniquement des vecteurs de support car les multiplicateurs de Lagrange sont non nuls pour ces seuls points comme expliqué à la remarque précédente. D'où,

$$\begin{aligned} w &= \sum_{i=1}^{n} \alpha_i y_i x_i \\ &= \sum_{x_i \in VS} \alpha_i y_i x_i \quad \text{où VS est l'ensemble des vecteurs de support.} \end{aligned}$$

Cette dernière égalité peut encore se justifier de la manière suivante : un grand nombre de termes de la première somme sont nuls. En effet, seuls les α_i correspondant aux données se trouvant sur les hyperplans canoniques sont non nuls (voir remarque ci-dessus). Ces données sont en fait les vecteurs de support. Ces derniers peuvent être vus comme les représentants de leur catégorie car si l'ensemble d'apprentissage n'était constitué que des vecteurs de support, l'hyperplan optimal que l'on trouverait serait identique.

Nous pouvons à présent caractériser la fonction de décision de notre classificateur linéaire. Cette fonction de décision est donnée par

$$h(x) = \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle + b = \sum_{x_i \in SV} \alpha_i y_i \langle x_i, x \rangle + b, \tag{3.8}$$

où les α_i sont les solutions du problème dual et b peut être facilement trouvé à partir de l'équation $y_i(\langle w, x_i\rangle + b) = 1$, avec $x_i \in VS$ et des conditions de KKT, ou encore à l'aide des variables primales

$$b = -\frac{\underset{y_i=-1}{max}(\langle w, x_i\rangle) + \underset{y_i=1}{max}(\langle w, x_i\rangle)}{2}.$$

Pour de plus amples informations concernant l'obtention de cette constante b, le lecteur se reportera à la Section 5.2.4. La résolution du problème dual permet donc de calculer les α_i et d'obtenir le vecteur w à moindre coût. Finalement, le classificateur SVM linéaire est donné par

$$\begin{aligned} f(x) &= sign(h(x)) \\ &= sign\left(\sum_{x_i \in SV} \alpha_i y_i \langle x_i, x\rangle + b\right). \end{aligned}$$

Remarques

- Le problème d'optimisation quadratique (3.7) et la fonction de décision (3.8) n'utilisent que les données de l'ensemble d'apprentissage, x_i, via des produits scalaires.
- La solution ne dépend plus de la dimension de l'espace d'entrée mais du nombre de vecteurs de support.

3.3 Classificateur SVM non-linéaire

Dans les problèmes réels, il est peu probable que les données soient exactement linéairement séparables. De plus, même si une frontière de décision courbe est possible, comme nous le verrons dans la section suivante, séparer exactement les données est souvent non désirable. En effet, s'il y a du bruit ou des erreurs sur les données, l'ensemble des données peut ne pas être linéairement séparable. Ces données sont généralement appelées des *outliers*. Une frontière de décision qui ignore quelques données est souvent meilleure qu'une frontière qui contourne les outliers.

Pour résoudre ce problème, nous mesurons le décalage entre ces outliers et les marges canoniques à l'aide de *variables de relaxation*, $\xi_i \geq 0$ ($\forall i : 1 \leq i \leq n$). En d'autres mots, nous introduisons des variables de relaxation pour assouplir les contraintes du problème primal linéaire et afin de s'assurer que le problème dual soit réalisable (voir Figure 3.7). Nous avons donc de nouvelles contraintes appelées *contraintes de relaxation*,

$$y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i \quad \text{avec} \quad \xi_i \geq 0 \quad \forall i : 1 \leq i \leq n.$$

La Figure 3.7 illustre les variables de relaxation ξ_i et ξ_j pour deux outliers x_i et x_j, respectivement. Ces contraintes de relaxation autorisent qu'un point puisse se trouver à une petite distance euclidienne ξ_i de l'autre côté de l'hyperplan canonique :

- du bon côté par rapport à l'hyperplan séparateur si $\xi_i \leq 1$ (cas de la donnée bleue x_i à la Figure 3.7) ;
- du mauvais côté par rapport à l'hyperplan séparateur si $\xi_i \geq 1$ (cas de la donnée rouge x_j à la Figure 3.7).

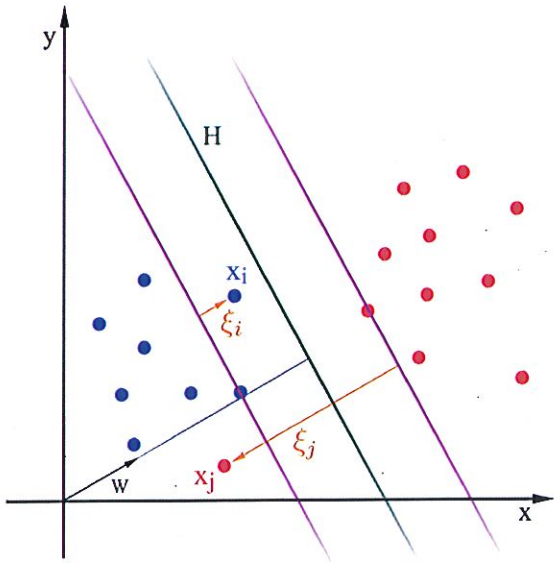


FIGURE 3.7 – Variables de relaxation.

Le problème primal devient alors le problème de la recherche de l'hyperplan impliquant la marge la plus grande et le nombre d'erreurs le plus petit

$$\begin{cases} \min & \underbrace{\frac{1}{2}||w||_2^2}_{\text{marge maximum}} + \underbrace{C\sum_{i=1}^{n}\xi_i}_{\text{erreur sur la marge}} \\ \text{s.c.} & y_i(\langle w, x_i\rangle + b) \geq 1 - \xi_i \quad \forall i : 1 \leq i \leq n \\ & \xi_i \geq 0 \quad \forall i : 1 \leq i \leq n \\ & C > 0, \end{cases} \tag{3.9}$$

où C est un paramètre de contrôle positif servant à pondérer (i.e., équilibrer) l'objectif de maximisation de la marge et le fait que la plupart des données soient bien classées. En d'autres mots, la constante C règle le compromis entre la marge possible et le nombre d'erreurs (i.e., le nombre de points mal classés) admissibles. Sans perte de généralité, nous nous intéressons uniquement aux contraintes $\xi_i \neq 0$ car pour les données correctement classées les ξ_i correspondant sont nulles.

Remarques

1. Le nombre de paramètres ξ_i non nuls sera aussi le nombre d'erreurs fait par un classificateur sur l'ensemble d'apprentissage.

2. La constante C permet de caractériser l'importance que l'on accorde aux erreurs commises sur l'ensemble d'apprentissage par rapport au fait de maximiser la marge. Si les données de l'ensemble d'apprentissage sont très bruitées, nous accordons d'avantage d'importance à la marge en fixant un C petit. Par contre, si les données comportent peu d'outliers ou si l'intérêt se porte plutôt sur les résultats obtenus sur l'ensemble d'apprentissage, nous utilisons une plus grande valeur de C.

En formant le Lagrangien puis en appliquant le théorème de Karush-Kuhn-Tucker nous en déduisons, de la même façon que précédemment, le problème dual qui a la même forme que dans le cas séparable

$$\begin{cases} \max\limits_{\alpha} & \sum\limits_{i=1}^{n}\alpha_i - \frac{1}{2}\sum\limits_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle \\ \text{s.c.} & 0 \leq \alpha_i \leq C \quad \forall i : 1 \leq i \leq n \\ & \sum\limits_{i=1}^{n}\alpha_i y_i = 0. \end{cases} \tag{3.10}$$

(3.11)

Remarque

Nous avons donc maintenant une borne supérieure sur les α_i.

Le calcul de w et b et de la fonction de décision $h(x)$ étant exactement le même que pour le cas linéaire nous ne le développons pas.

3.4 Fonction noyau

Le classificateur à marge maximale que nous avons présenté dans les sections précédentes permet d'obtenir de très bons résultats lorsque les données sont linéairement séparables. Naturellement, de nombreux jeux de données sont non-linéairement séparables. Pour classer ce genre de données nous pourrions utiliser une fonction de décision non-linéaire. Géométriquement, cela reviendrait à avoir une (hyper)courbe qui marquerait la frontière entre les données positives et négatives. Le problème majeur de ce genre de méthode vient du fait que le nombre de paramètres à calculer est très élevé et que l'on est très souvent confronté au problème du sur-apprentissage.

Afin de remédier au problème de l'absence de classificateur linéaire, nous nous appuyons sur une idée apportée par les SVM et introduite à la Section 2.4.3. Au lieu de chercher un hyperplan non-linéaire dans l'espace des entrées, il suffit de reconstituer le problème dans un espace de représentation intermédiaire appelé *espace de redescription*, noté $\mathcal{R}$, de dimension strictement supérieure à l'espace des entrées, éventuellement de dimension infinie, dans lequel les données

sont linéairement séparables. En effet, plus la dimension de l'espace de redescription est grande, plus la probabilité de pouvoir trouver un hyperplan séparateur linéaire optimal entre les données est élevée. Pour cela, nous allons appliquer aux vecteurs d'entrées x_i une transformation non-linéaire Φ de l'espace d'entrée dans l'espace de redescription

$$\begin{array}{rccc} \Phi: & \mathcal{X} = \mathbb{R}^p & \longrightarrow & \mathcal{R} \\ & x_i & \mapsto & \Phi(x_i) \end{array}$$

et nous pouvons ensuite appliquer la même méthode de maximisation de la marge dans $\mathcal{R}$.

Remarque

> Comme nous venons de le préciser, la dimension de l'espace de redescription peut être très élevée, voire de dimension infinie. Cela ne pose aucun problème pour notre classificateur à marge maximale SVM. En effet, grâce à sa formulation duale, le nombre de variables à déterminer ne dépend que de la taille de l'ensemble d'apprentissage.

Comme précédemment, nous recherchons donc dans ce nouvel espace l'hyperplan séparateur

$$h(x) = \langle w, \Phi(x) \rangle + b.$$

Pour cela, nous transformons chaque entrée x_i en $\Phi(x_i)$ dans l'espace de redescription $\mathcal{R}$. Les conditions de classification (i.e., contraintes) deviennent alors

$$y_i(\langle w, \Phi(x_i) \rangle + b) \geq 1 \quad \forall i : 1 \leq i \leq n.$$

L'hyperplan à marge maximale dans ce nouvel espace est la solution du problème quadratique d'optimisation

$$\begin{cases} \min & \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.c.} & y_i(\langle w, \Phi(x_i) \rangle + b) \geq 1 - \xi_i \quad \forall i : 1 \leq i \leq n \\ & \xi_i \geq 0 \quad \forall i : 1 \leq i \leq n \\ & C > 0. \end{cases}$$

En utilisant le même raisonnement que précédemment et en introduisant les multiplicateurs de Lagrange α_i pour chaque contrainte de classification, nous obtenons le Lagrangien

$$L(w, b; \alpha) = \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(\langle w, \Phi(x_i) \rangle + b) - 1). \qquad (3.12)$$

Comme précédemment, nous en déduisons alors le problème d'optimisation dual suivant

$$\begin{cases} \max\limits_{\alpha} & \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \langle \Phi(x_i), \Phi(x_j) \rangle \\ \text{s.c.} & 0 \leq \alpha_i \leq C \quad \forall i : 1 \leq i \leq n \\ & \sum_{i=1}^{n}\alpha_i y_i = 0, \end{cases} \qquad (3.13)$$

où $\langle \cdot, \cdot \rangle$ représente le produit scalaire dans le nouvel espace.

La solution (i.e., l'équation de l'hyperplan séparateur) dans le nouvel espace est donnée comme précédemment par

$$h(x) = \sum_{i=1}^{n} \alpha_i y_i \langle \Phi(x_i), \Phi(x) \rangle + b = \sum_{x_i \in SV} \alpha_i y_i \langle \Phi(x_i), \Phi(x) \rangle + b,$$

où les coefficients α_i et b sont obtenus par résolution du problème d'optimisation (3.13).

Le problème de cette formulation et de sa solution sont qu'elles ne dépendent que des produits scalaires $\langle \Phi(x_i), \Phi(x_j) \rangle$ et $\langle \Phi(x_i), \Phi(x) \rangle$ dans l'espace de redescription de dimension élevée. Ceci devient vite coûteux en terme de calculs quand la dimension de $\mathcal{R}$ augmente, d'autant plus que l'on utilisera des transformations non-linéaires. Cet inconvénient peut rendre le calcul de l'hyperplan optimal fastidieux, voire impossible. C'est pourquoi, plutôt que de travailler avec la transformation non-linéaire

$$\Phi : \mathcal{X} \longrightarrow \mathcal{R},$$

nous déterminons et nous travaillons avec une fonction plus "économique" en terme de calculs

$$k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$$

telle que,

$$k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle = k(x_j, x_i).$$

Une telle fonction est appelée *fonction noyau*.

Propriétés :

1. La fonction noyau est un produit scalaire dans l'espace de représentation intermédiaire ce qui implique qu'elle est linéaire. Cela nous permettra de pouvoir faire le rapprochement avec le cas linéaire des sections précédentes.

2. Le calcul se fait dans l'espace d'origine, ceci est beaucoup moins coûteux qu'un produit scalaire en grande dimension.

3. La fonction noyau traduit la répartition des données dans l'espace de représentation intermédiaire.

4. Lorsque k est bien choisi, nous n'avons pas besoin de connaître explicitement la transformation Φ et a fortiori de calculer la représentation des données dans ce nouvel espace. En effet, seule la fonction noyau intervient dans les calculs. Nous pouvons donc envisager des transformations complexes et même des espaces de redescription de dimension infinie.

Finalement, pour calculer l'hyperplan séparateur optimal dans l'espace de redescription, il suffit de remplacer toutes les occurences du produit scalaire par le noyau. Cette nouvelle approche est introduite dans la Section 3.5 où elle est développée en détail.

3.4.1 Condition de Mercer

Dans cette section, nous établissons une condition permettant de déterminer si une fonction

$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$

correspondant à un produit scalaire dans l'espace de redescription $\mathcal{R}$ est ou n'est pas une fonction noyau. En pratique, nous ne connaissons pas la transformation Φ, introduite dans la section précédente. Nous construisons plutôt une fonction noyau qui doit respecter certaines conditions. Cette fonction noyau doit correspondre à un produit scalaire dans un espace de grande dimension. Le théorème de Mercer explicite les conditions auxquelles la fonction k doit satisfaire pour être un noyau. Avant d'énoncer le théorème de Mercer, nous rappelons la définition de matrice de Gram ainsi que la définition de matrice semi-définie positive qui nous seront utiles par la suite.

Définition 3.4.1 *(Matrice de Gram)*
Soit l'ensemble d'apprentissage

$$\mathcal{S} = \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

la matrice de Gram K est la matrice symétrique de terme général $k(x_i, x_j)$

$$K = \begin{pmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \cdots & k(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n,x_1) & k(x_n,x_2) & \cdots & k(x_n,x_n) \end{pmatrix}.$$

Définition 3.4.2 *(Matrice semi-définie positive)*
Une matrice A de dimension $p \times p$, dont les éléments sont des réels, est semi-définie positive si et seulement si

$$x^T A x \geq 0, \quad \forall x \in \mathbb{R}^p,$$

ce qui revient à exiger que toutes les valeurs propres de A soient positives ou nulles.

Nous pouvons à présent énoncer le théorème de Mercer fournissant une condition nécessaire et suffisante pour qu'une fonction $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ soit un noyau.

Théorème 3.4.1 *(Théorème de Mercer)*
Une fonction $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ est un noyau si, pour un ensemble d'apprentissage $\{(x_1, y_1)..., (x_n, y_n)\}$, la matrice de Gram :

$$K = (k(x_i, x_j))_{i,j=1}^{n}$$

est une matrice semi-définie positive.

En d'autres mots, pour n'importe quel ensemble d'apprentissage

$$\mathcal{S} = \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

la matrice de Gram K est non négative.

Si k vérifie la condition de Mercer, alors il existe un espace de Hilbert[1] $\mathcal{R}$ muni d'un produit scalaire $\langle \cdot, \cdot \rangle$ et une fonction $\Phi : \mathcal{X} \to \mathcal{R}$ tels que

$$k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle = k(x_j, x_i), \quad \forall x_i, x_j \in \mathcal{X}.$$

Remarques

- La preuve de ce théorème peut être trouvée dans [66].
- Nous revenons donc au cas d'un classificateur linéaire sans changement d'espace. L'approche par noyau généralise ainsi l'approche linéaire.
- La condition de Mercer nous indique si une fonction est ou non un noyau. Cependant, elle ne nous donne aucun renseignement explicite sur la transformation Φ et aucune indication en ce qui concerne la construction des noyaux.
- Une fonction $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ générant une matrice semi-définie positive possède donc les trois propriétés fondamentales du produit scalaire :

  1. **Positivité**
  $$k(x_i, x_j) \geq 0, \quad \forall x_i, x_j \in \mathcal{X},$$
  2. **Symétrie**
  $$k(x_i, x_j) = k(x_j, x_i), \quad \forall x_i, x_j \in \mathcal{X},$$
  3. **Inégalité de Chauchy-Schwartz**
  $$|k(x_i, x_j)| \leq \sqrt{k(x_i, x_i)}\sqrt{k(x_j, x_j)}, \quad \forall x_i, x_j \in \mathcal{X}.$$

1. Un espace de Hilbert est un espace de Banach dont la norme découle d'un produit scalaire $\langle \cdot, \cdot \rangle$ par la formule $\|x\|_2 = \sqrt{\langle x, x \rangle}$. C'est en fait la généralisation en dimension quelconque d'un espace euclidien.

3.4.2 Exemples de noyaux

Dans cette section, nous présentons quatre types de noyaux standards. Nous étudions ensuite plus en détail le noyau RBF [4] qui est celui utilisé lors des différents tests des SVM pour la régression (voir Chapitre 6).

3.4.2.1 Les noyaux standards

Outre le noyau RBF Gaussien présenté à la section suivante, les principaux noyaux standards sont donnés dans le tableau suivant :

Noyau	$k(x_i, x_j)$	Hyperparamètres
Linéaire	$\langle x_i, x_j \rangle$	
Polynomial	$(c + \langle x_i, x_j \rangle)^d$	$d \in \mathbb{N}$ et $c \in \mathbb{R}^+$
RBF Gaussien	$e^{\frac{-\|\|x_i - x_j\|\|_2^2}{2\sigma^2}}$	$\sigma > 0$
Sigmoidal	$tanh(c + d\langle x_i, x_j \rangle)$	$c, d \in \mathbb{R}^+$

Tableau 3.1 – Expression des noyaux standards.

Exemple de noyau polynomial

Supposons un ensemble d'apprentissage dont les données sont distribuées, selon une loi uniforme, pour la première classe dans un disque de rayon 1 et pour la deuxième classe dans un anneau, comme illustré à la Figure 3.8. Nous voyons clairement que les deux classes ne sont pas linéairement séparables.

Soit $x = (x_1, x_2)$, considérons la transformation non-linéaire

$$\begin{array}{rccc} \Phi : & \mathbb{R}^2 & \longrightarrow & \mathbb{R}^3 \\ & x = (x_1, x_2) & \mapsto & \Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2). \end{array}$$

Si l'on essaie de séparer les données, notre frontière de décision sera l'hyperplan dans $\mathbb{R}^3$, $(\langle w, \Phi(x) \rangle) + b = 0$, qui sera de la forme

$$w_1x_1^2 + \sqrt{2}w_2x_1x_2 + w_3x_2^2 = 0$$

et est l'équation d'une ellipse. Après transformation, les deux classes deviennent donc linéairement séparables, comme l'illustre le deuxième graphe de la Figure 3.8.
Soit x et $x' \in \mathbb{R}^2$, intéressons-nous maintenant au produit scalaire dans l'espace de redescription, celui-ci est donné par :

$$\begin{aligned} \langle \Phi(x), \Phi(x') \rangle &= [x]_1^2[x]_1'^2 + 2[x]_1[x]_2[x]_1'[x]_2' + [x]_2^2[x]_2'^2 \\ &= ([x]_1[x]_1' + [x]_2[x]_2')^2 \\ &= \langle x, x' \rangle^2 \\ &= k(x, x'). \end{aligned}$$

4. Nous n'étudions que le noyau RBF car c'est le noyau qui donne les meilleurs résultats dans le cas des SVM pour la régression, comme nous le verrons au Chapitre 6.

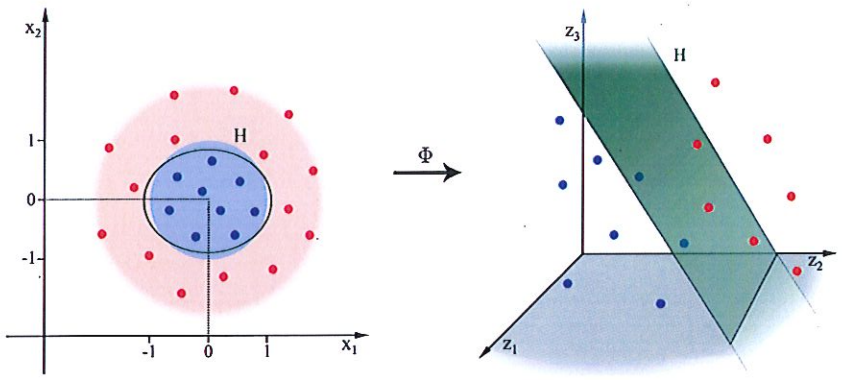


FIGURE 3.8 – Transformation non-linéaire Φ.

Nous pouvons donc calculer $\langle \Phi(x), \Phi(x') \rangle$ sans connaître explicitement Φ à l'aide de la fonction noyau. De plus, cette fonction noyau

$$k(x, x') = \langle x, x' \rangle^2$$

est une instance particulière du noyau polynomial

$$(c + \langle x, x' \rangle)^d$$

où $c = 0$ et $d = 2$.

3.4.2.2 Le noyau RBF Gaussien

Le noyau RBF Gaussien est donc donné par

$$k(x_i, x_j) = e^{\frac{-||x_i - x_j||_2^2}{2\sigma^2}}, \qquad \text{avec } \sigma > 0.$$

La transformation Φ induite par ce noyau est particulière. Elle va en fait "mapper" une donnée sur une fonction gaussienne représentant la similarité de la donnée avec toutes les données de $\mathcal{X}$. La Figure 3.9 illustre la transformation Φ relative au noyaux RBF Gaussien pour deux données x_i et x_j.

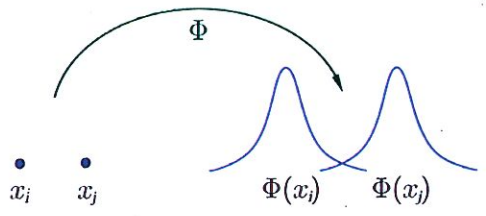


FIGURE 3.9 – Transformation Φ des données x_i et x_j.

L'espace de redescription $\mathcal{R}$ de cette transformation Φ est de dimension infinie, étant donnée que Φ fait correspondre une fonction continue à chaque donnée. Le noyau RBF Gaussien permet donc de calculer des similarités dans un espace de dimension infinie.

Le paramètre σ permet de régler la largeur de la gaussienne. En prenant un σ important, la similarité d'une donnée par rapport à celles qui l'entourent sera grande. Contrairement, en prenant un σ qui tend vers 0, la similarité d'une donnée par rapport à celles qui l'entourent sera faible. La Figure 3.10 illustre cette propriété. En fixant le paramètre σ proche de 0, un classificateur utilisant ce noyau peut arriver à apprendre n'importe quel ensemble d'apprentissage sans commettre d'erreur. Cela signifie qu'il apprendrait toutes les données par coeur. Nous nous trouvons alors dans une situation de sur-apprentissage. Le paramètre σ est donc un hyperparamètre de l'algorithme d'apprentissage qu'il faut fixer avec beaucoup de précaution.

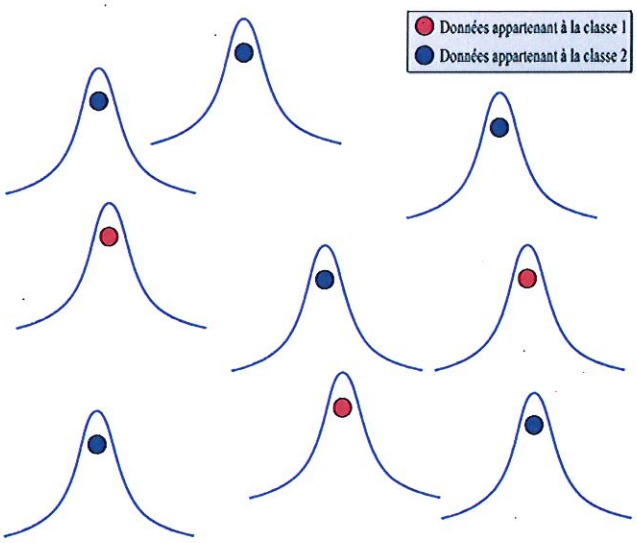


FIGURE 3.10 – Similarité de toutes les données de l'ensemble d'apprentissage vue en utilisant un noyau RBF Gaussien avec un σ tendant vers 0.

3.4.3 Composition de noyaux

Il est possible de créer de nouveaux noyaux à l'aide de noyaux existants. En effet, la composition de noyaux est encore un noyau. Nous pouvons donc énoncer le théorème suivant.

Théorème 3.4.2 *(Composition de noyaux)*
Soient k_1 et k_2, deux noyaux sur $\mathcal{X} \times \mathcal{X}$, k_3 un noyau sur $\mathbb{R}^n \times \mathbb{R}^n$, $\alpha \in \mathbb{R}^+$, $\phi : \mathcal{X} \to \mathbb{R}^n$, une transformation dans $\mathbb{R}^n$ et A une matrice semi-définie positive, alors pour tout $x_i, x_j \in \mathcal{X}$ les fonctions suivantes sont des noyaux :

$$\begin{aligned} k(x_i, x_j) &= k_1(x_i, x_j) + k_2(x_i, x_j) \\ k(x_i, x_j) &= k_1(x_i, x_j) k_2(x_i, x_j) \\ k(x_i, x_j) &= \alpha k_1(x_i, x_j) \\ k(x_i, x_j) &= k_3(\phi(x_i), \phi(x_j)) \\ k(x_i, x_j) &= exp(k_1(x_i, x_j)) \\ k(x_i, x_j) &= x_i^T A x_j. \end{aligned}$$

3.5 Classificateur SVM : formulation générale

3.5.1 Cas non-linéairement séparable : marge fixe

La méthode SVM consiste en un classificateur à marge maximale dans lequel le produit scalaire est remplacé par le noyau. Dès lors, nous pouvons remplacer le produit scalaire $\langle \Phi(x_i), \Phi(x_j) \rangle$ dans (3.13) par le noyau $k(x_i, x_j)$. Nous obtenons alors le problème d'optimisation avec noyau

$$\begin{cases} \max\limits_{\alpha} & \sum\limits_{i=1}^{n} \alpha_i - \frac{1}{2} \sum\limits_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{s.c.} & \sum\limits_{i=1}^{n} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \quad \forall i : 1 \leq i \leq n, \end{cases} \tag{3.14}$$

où les contraintes sont celles du problème (3.7), c'est-à-dire sans les variables de relaxation.

3.5.2 Cas non-linéairement séparable : marge souple

Généralement, l'ensemble d'apprentissage comporte des outliers comme nous l'avons vu à la Section 3.3. La méthode SVM à marge souple doit alors être utilisée afin de pouvoir traiter ces ensembles d'apprentissage non-linéairement séparables contenant des outliers [5]. En effet, même s'il existe une relation linéaire (dans l'espace de redescription $\mathcal{R}$) entre les données et leur catégorie, un classificateur linéaire pourrait commettre des erreurs. Nous pourrions trouver un espace de redescription à l'aide d'un noyau RBF Gaussien avec un σ assez petit de telle manière que les données soient linéairement séparables. Cependant, cela reviendrait à apprendre le bruit, les erreurs des données et donc à perdre une grande partie du pouvoir de généralisation dû au sur-apprentissage. Il est donc plus raisonnable d'admettre que certaines données soient mal classées par notre classificateur. La méthode SVM à marge souple est donc simplement obtenue à partir de (3.10) où les données sont transformées dans un espace de redescription à l'aide d'une fonction noyau. La forme duale de la méthode SVM

5. La notion de linéairement séparable et d'outlier s'applique ici dans l'espace de redescription.

à marge souple est alors

$$\begin{cases} \max\limits_{\alpha} & \sum\limits_{i=1}^{n}\alpha_i - \frac{1}{2}\sum\limits_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j k(x_i,x_j) \\ \text{s.c.} & 0 \leq \alpha_i \leq C \quad \forall i : 1 \leq i \leq n \\ & \sum\limits_{i=1}^{n}\alpha_i y_i = 0, \end{cases} \tag{3.15}$$

et la fonction de décision finale est donnée par

$$h(x) = \sum_{i=1}^{n}\alpha_i y_i k(x_i,x) + b = \sum_{x_i \in SV}\alpha_i y_i k(x_i,x) + b.$$

Finalement, le classificateur SVM non-linéaire est donné comme suit

$$\begin{aligned} f(x) &= sign(h(x)) \\ &= sign\left(\sum_{x_i \in SV}\alpha_i y_i k(x_i,x) + b\right). \end{aligned}$$

Nous terminons en mentionnant que l'on peut déterminer le statut d'une donnée x_i, en appliquant les conditions nécessaires et suffisantes de Karush-Kuhn-Tucker pour la convergence vers un optimum et en regardant sa variable duale α_i. Pour le problème d'optimisation à marge souple, ces conditions sont :

1. $\alpha_i = 0$: la donnée est bien classée et n'est pas sur un des deux hyperplans canoniques. Il ne s'agit donc pas d'un vecteur de support ;

2. $0 < \alpha_i < C$: la donnée est bien classée et se trouve sur un hyperplan canonique. Il s'agit donc d'un vecteur de support ;

3. $\alpha_i = C$: la donnée est mal classée. Elle sera malgré tout considérée comme vecteur de support puisque $\alpha_i > 0$. Il s'agit d'un *outlier*.

Remarques

Les méthodes d'optimisation quadratiques standards pour la résolution du problème d'optimisation (3.15) suffisent généralement. Cependant, lorsque le nombre de données de l'ensemble d'apprentissage devient trop important (généralement $n > 5000$) il faut développer des méthodes d'optimisation spécifiques telles que le chunking [37] ou encore l'algorithme SMO pour la classification [55]. De plus, différents programmes d'optimisation sont disponibles gratuitement sur internet. Les principaux sont : Libsvm, Weka, SVM^{light}, LOQO, MINOS ou encore CPLEX. Un résumé de la plupart des algorithmes pour le problème de la classification implémentés par ces programmes est donné dans [9].

3.6 Etude de l'unicité et de la globalité de la solution

Les différents problèmes d'optimisation rencontrés sont des problèmes d'optimisation convexes. Cette propriété est due au fait que la matrice du noyau est une matrice définie positive. Si la matrice de Gram est définie positive, alors la solution du problème est unique. Si par contre, nous laissons la définition inchangée c'est-à-dire que la matrice de Gram est semi-définie positive, alors la solution du problème n'est pas unique mais toute solution locale est aussi une solution globale. Par conséquent, en ayant trouvé une solution, nous sommes certains qu'il s'agit bien d'une solution optimale. En d'autres mots, la solution est globale mais pas nécessairement unique. En effet, il peut arriver que l'hyperplan déterminé par w et b soit unique mais pas les α_i. Ce qui signifie qu'il peut exister une expression de w qui requiert plus ou moins de vecteurs de support.

3.7 Relation avec l'apprentissage statique

Dans cette section, nous donnons un aperçu général des idées principales et concepts de l'apprentissage statique. Le but n'est pas de décrire toute la théorie de l'apprentissage statique de Vapnik [66] mais de justifier l'utilisation des SVM. Le lecteur peut se référer à [66] pour une étude complète de la théorie de l'apprentissage statique.

3.7.1 Risque empirique et risque fonctionnel

Le but d'un algorithme d'apprentissage est de minimiser le risque fonctionnel (2.1). Cependant, comme nous l'avons vu à la Section 2.1.1, la distribution $p(x, y)$ est inconnue. Le risque fonctionnel ne peut donc pas être calculé. En pratique, un principe de construction de modèle par un algorithme d'apprentissage doit être choisi. Une des meilleures approches est de choisir le principe de minimalisation du risque empirique (MRE) qui permet de retenir la meilleure fonction sur l'ensemble d'apprentissage $\mathcal{S}$ comme modèle final. Cette approche nécessite d'approximer le risque fonctionnel par le risque empirique. Celui-ci est donné pour un certain modèle (i.e, fonction) f par

$$R_{emp}[f] = E_{\mathcal{S}}[f] = \frac{1}{n}\sum_{i=1}^{n} l(f(x), y). \tag{3.16}$$

où

$$l(f(x), y) = \begin{cases} 1 \text{ si } f(x) \neq y, \\ 0 \text{ sinon.} \end{cases}$$

Mais minimiser le risque empirique n'implique pas d'avoir un minimum pour le risque fonctionnel. Nous sommes confrontés une fois de plus au phénomène du sur-apprentissage. Sans contrainte sur la classe $\mathcal{F}$ de fonctions admissibles f, le risque empirique $E_{\mathcal{S}}[f]$ peut toujours être égalé à zéro mais cela n'implique pas pour autant que f ait la possibilité de correctement généraliser (i.e., classer correctement une nouvelle donnée). La théorie d'apprentissage statique étudie les cas dans lesquels le principe du MRE est consistant (i.e., quand il est suffisant de minimiser (3.16) pour avoir un risque fonctionnel faible).

3.7.2 Consistance non triviale et convergence uniforme

La définition de la *consistance* pour le principe MRE est donné comme suit

$$\forall \varepsilon > 0, \quad \lim_{n \to +\infty} P(|R[f] - E_{\mathcal{S}}[f]| \geq \varepsilon) = 0,$$

où, pour l'ensemble d'apprentissage $\mathcal{S}$ de taille n, $f \in \mathcal{F}$ minimise le risque empirique $E_{\mathcal{S}}[f]$. Cette condition de la consistance assure que le risque empirique converge en probabilité vers le risque fonctionnel lorsque la taille de l'ensemble d'apprentissage tend vers l'infini (loi des grands nombres). La Figure 3.11 illustre la propriété de consistance.

Un des buts de la théorie d'apprentissage statique est d'obtenir les conditions de consistance en terme de caractéristique générale de la classe de fonctions admissibles $\mathcal{F}$. Afin d'étudier la consistance uniquement sur base des propriétés générales de $\mathcal{F}$, Vapnik définit la notion de *consistance non triviale* basée sur la consistance de sous-ensembles de $\mathcal{F}$ après avoir retiré les fonctions dont le risque est le plus faible. Cette consistance non triviale peut être vue comme équivalente à la convergence uniforme

$$\forall \varepsilon > 0, \quad \lim_{n \to +\infty} P\left(\sup_{f \in \mathcal{F}} |R[f] - E_{\mathcal{S}}[f]| \geq \varepsilon\right) = 0.$$

Cela signifie que le risque fonctionnel de la plus mauvaise fonction de $\mathcal{F}$ (i.e., la fonction f tel que $R[f]$ est maximal) peut converger vers le risque empirique lorsque n tend vers l'infini. Pour plus d'informations, le lecteur peut se référer à [66, 40].

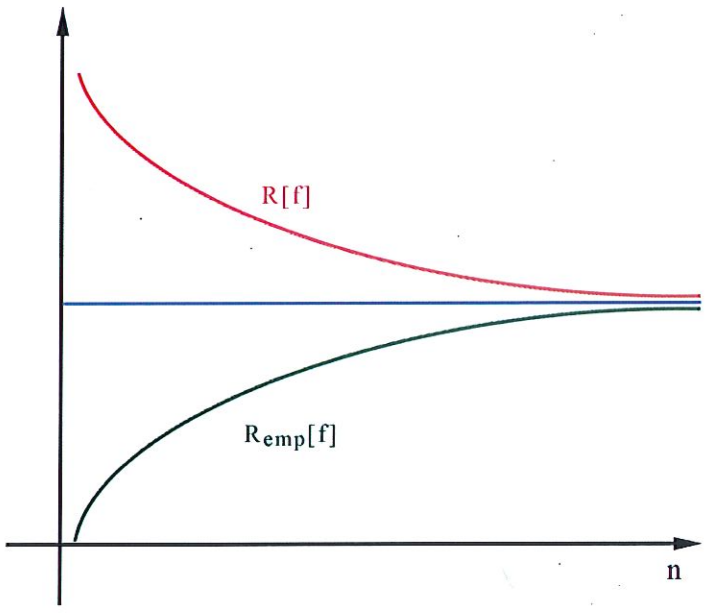


FIGURE 3.11 – Consistance.

3.7.3 Borne sur l'erreur d'apprentissage

La théorie de l'apprentissage statique permet de déterminer des bornes sur le risque fonctionnel $R[f]$ dans le but de garantir une certaine performance pour un algorithme d'apprentissage fixé. L'étude de la vitesse de la convergence uniforme amène à des bornes sur le risque empirique : une vitesse de convergence rapide donne lieu à des bornes étroites pour un n particulier. Nous allons borner le risque fonctionnel $R[f]$ pour la plus mauvaise fonction $f \in \mathcal{F}$ et donc aussi $\forall f \in \mathcal{F}$. Il est clair que les bornes ne peuvent être obtenues pour une classe de fonction $\mathcal{F}$ non contrainte.

3.7.4 Dimension VC

Mesurer la *capacité*[6] d'une classe de fonction $\mathcal{F}$ est crucial. Une mesure de cette capacité est la dimension[7] h de Vapnik-Chervonenkis (VC). Pour la classification binaire, h est le nombre maximum de données qui peuvent être séparées en deux classes parmi toutes les possibilités en utilisant une fonction $f \in \mathcal{F}$ (2^h possibilités).

Exemple

Considérons trois données représentées dans $\mathbb{R}^2$. Supposons que la classe de fonctions $\mathcal{F}$ correspond aux droites de $\mathbb{R}^2$

$$f = ax + b \quad \text{avec } a, b \in \mathbb{R}.$$

La dimension VC de $\mathcal{F}$ est égale à 3. En effet, nous pouvons trouver exactement 8 configurations séparant les 3 données de toutes les façons possibles. D'où $2^h = 8$ implique que la dimension VC, h, est égale à 3. La Figure 3.12 illustre les 8 possibilités.

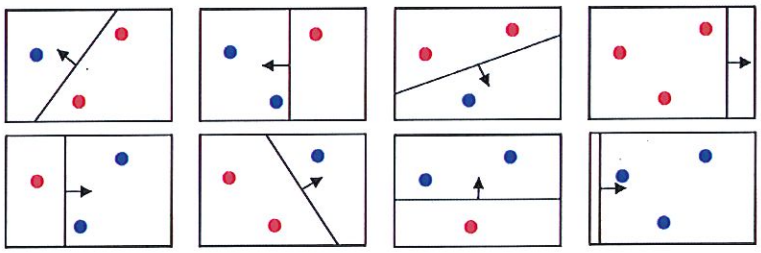


FIGURE 3.12 – Exemple de configuration de 3 données séparables de toutes les manières possibles par les droites de $\mathbb{R}^2$. Les points bleus et rouges représentent respectivement les données assignées positivement et négativement. La flèche représente le côté de la droite où les données seront classées positivement.

6. Par la suite, nous utiliserons indifféremment les termes capacité et complexité.
7. Ne pas confondre la dimension VC, h, avec la fonction de décision, $h(x)$, d'un classificateur.

Grâce à la dimension VC, Vapnik [66] montre que le risque fonctionnel peut être borné avec une probabilité de $1 - \delta$ par

$$R[f] \leq E_{\mathcal{S}}[f] + \sqrt{\frac{h(ln(\frac{2n}{h})) + 1 - ln(\frac{\delta}{4})}{n}}, \tag{3.17}$$

où :

- n est la dimension de l'ensemble d'apprentissage,
- h est la dimension VC,
- $\delta > 0$ est défini par le théorème 2.4 de [66].

L'inégalité (3.17) implique que la capacité de $\mathcal{F}$, h, et le risque empirique, $E_{\mathcal{S}}[f]$, doivent être minimisés pour garantir un risque fonctionnel faible. L'inégalité (3.17) nous permet de définir le terme de capacité.

Définition 3.7.1 *(Terme de capacité)*
Le terme de capacité est défini donné par

$$C(n, h, \delta) = \sqrt{\frac{h\left(ln\left(\frac{2n}{h}\right)\right) + 1 - ln\left(\frac{\delta}{4}\right)}{n}}. \tag{3.18}$$

Remarques

L'inégalité (3.17) fait apparaître deux cas particuliers :

1. Une classe de fonctions restreinte permettant d'expliquer la relation entre les données et leurs classes de manière "grossière" (i.e., sous-apprentissage) fait rapidement décroître le terme de la capacité mais implique un risque empirique très important.

2. Une classe de fonctions très grande (i.e., h élevé) permettant de calculer un modèle complexe risquant d'apprendre le bruit associé aux données d'apprentissage fait rapidement décroître le risque empirique (i.e., sur-apprentissage) mais implique un terme de capacité très important.

La meilleure classe de fonctions est généralement intermédiaire entre ces deux classes extrêmes car nous cherchons une fonction f qui permet d'expliquer correctement les données (i.e., de bien généraliser) tout en ayant un risque empirique faible. Cette idée est illustrée à la Figure 3.13.

3.7.5 Bornes basées sur la marge

La dimension VC d'une classe d'hyperplans séparateurs peut être bornée par une fonction de la marge $\rho(w, b)$ [66]. Le risque fonctionnel peut donc être

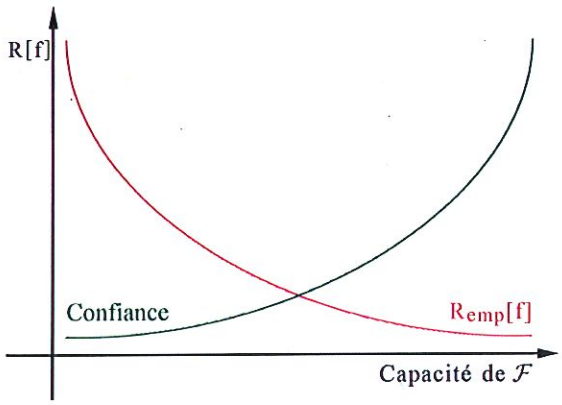


FIGURE 3.13 – Illustration de l'inégalité (3.17). La courbe croissante, appelée *confiance*, correspond à la borne supérieure du terme de capacité.

exprimé en fonction de la marge. La borne sur le risque fonctionnel qui suit est donnée avec une probabilité de $1 - \delta$

$$R[f] \leq \frac{c}{n}\left(\frac{R^2 + ||\xi||_1^2 log_2(\frac{1}{\rho})}{\rho^2} log_2^2 n + log_2 \frac{1}{n}\right), \tag{3.19}$$

où :

- n est le nombre de données de l'ensemble d'apprentissage,
- c est une constante,
- ξ est le vecteur contenant toutes les variables de relaxation ξ_i pour $i = 1, ..., n$ (voir Section 3.3),
- ρ est la marge,
- R est le rayon de la boule minimum englobant les données dans l'espace d'entrée $\mathcal{X}$. (i.e., $\forall x \in \mathcal{X} : ||\Phi(x)||_2 = \sqrt{\langle \Phi(x), \Phi(x) \rangle} \leq R$).

Cette borne implique clairement que maximiser la marge amène à une meilleure capacité de généralisation (i.e., un risque fonctionnel faible).

3.7.6 Minimisation du risque empirique et fonctionnel

Lorsque le nombre de données n de l'ensemble d'apprentissage est grand, le principe MRE est consistant (i.e., bien adapté) et amène à une borne faible pour le risque fonctionnel. Cependant, lorsque le nombre de données est faible $\left(\frac{n}{h} < 20\right)$ alors,

$$C(n, h, \delta) = \sqrt{\frac{h\left(ln\left(\frac{2n}{h}\right)\right) + 1 - ln\left(\frac{\delta}{4}\right)}{n}} \tag{3.20}$$

devient important et minimaliser le risque empirique $E_S[f]$ n'assure plus un risque fonctionnel faible. A contrario, le principe de Minimalisation du Risque Structurel (MRS) propose une stratégie pour minimiser les deux termes en

contrôlant la capacité de la classe de fonctions $\mathcal{F}$. Nous considérons une famille de fonctions imbriquées

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset ... \subset \mathcal{F}_M$$

et les fonctions $f_i \in \mathcal{F}_i$ minimisant le risque empirique $E_{\mathcal{S}}[f_i]$, sur l'ensemble d'apprentissage $\mathcal{S}$. Nous avons donc

$$E_{\mathcal{S}}[f_1] \geq E_{\mathcal{S}}[f_2] \geq ... \geq E_{\mathcal{S}}[f_M]$$

et leurs dimensions VC, h_i, vérifient la propriété suivante illustrée à la Figure 3.14

$$h_1 \leq h_2 \leq ... \leq h_M.$$

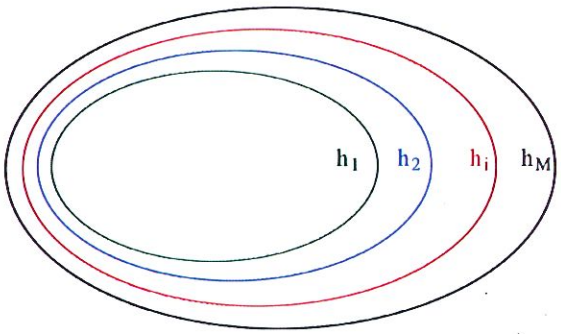


FIGURE 3.14 – Minimisation du risque structurel. Les sous-ensembles sont ordonnés selon leur dimension VC.

Puisque $C(n, h, \delta)$ augmente avec h, cette mesure de la capacité de la classe de fonctions peut être contrôlée. La stratégie MRS est de trouver f_i pour chacune des classes $\mathcal{F}_i$ et choisir la classe, et donc la fonction f_i, qui minimise la borne (3.17). Elle permet donc de trouver le compromis entre la qualité de l'approximation sur l'ensemble d'apprentissage et la complexité de la fonction f_i qui réalise l'approximation. Cette stratégie peut être vue comme une minimisation multi-objectifs du risque empirique et de la capacité. Finalement, le risque structurel est illustré à la Figure 3.15.

3.7.7 Algorithmes d'apprentissage

Comme vu auparavant, le principe MRS exige la minimisation du risque empirique et de la capacité de la machine d'apprentissage afin de minimiser le risque fonctionnel. Sur base de cette considération, deux types d'approches peuvent être formulés :

1. Fixer la capacité de la machine d'apprentissage et minimiser le risque empirique. Les réseaux de neurones sont un exemple de ce type d'algorithmes d'apprentissage.

2. Fixer la valeur du risque empirique et minimiser la capacité. Les SVM font partie de ce type d'algorithme d'apprentissage.

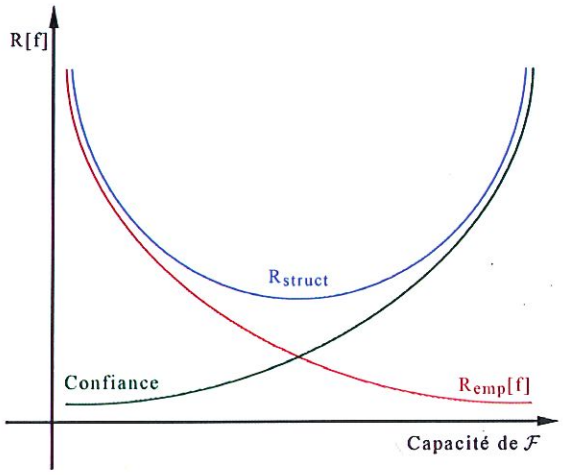


FIGURE 3.15 – Principe de minimisation du risque structurel.

3.7.8 En conclusion

Les deux résultats importants concernant les SVM sont les suivants :

1. La capacité de généralisation d'une machine d'apprentissage dépend plus de la capacité de l'ensemble de fonctions implémentables $\mathcal{F}$ (i.e., de la dimension VC de cet ensemble de fonctions) que de la dimension de l'espace. D'où, une fonction qui décrit les données d'apprentissage correctement et qui appartient à un ensemble de fonctions $\mathcal{F}$ de faible capacité permet de bien généraliser (i.e., de classer correctement de nouvelles données) quelle que soit la dimension de l'espace.

2. Nous avons vu dans la Section 3.4 que pour construire un hyperplan, l'étape la plus importante est l'évaluation du produit scalaire entre les vecteurs d'entrées de l'ensemble d'apprentissage. Cependant, dans un espace de Hilbert, le produit scalaire admet une représentation équivalente sous forme de noyau. Le produit scalaire ne dépend donc également plus de la dimension de l'espace.

Ces deux résultats théoriques nous permettent de construire des classificateurs linéaires de faible capacité dans des espaces de redescription de grande dimension tout en évitant le problème de la malédiction de la dimension. De plus, nous avons vu que la construction d'un classificateur SVM linéaire permet de contrôler la capacité dans l'espace de redescription et que cette construction peut être formulée comme un problème d'optimisation convexe admettant une solution unique. Par ailleurs, nous avons aussi vu que de nombreux algorithmes ont été proposés pour résoudre ce problème. En conclusion, les SVM se fondent donc sur une base théorique solide. Pour de plus amples informations concernant la théorie développée dans cette section, le lecteur peut se référer à [66].

Chapitre 4

Machines à vecteurs de support pour la régression

RÉSUMÉ. *Dans ce chapitre, nous présentons la méthode des SVM avec noyau dans le cadre de la régression. La méthode des SVM pour la classification, présentée dans le Chapitre 3, est donc reformulée pour la régression, donnant lieu à la technique bien connue de la régression par vecteurs de support (SVR pour Support Vector Regression) introduite par Drucker et al [19] en 1997, par Vapnik-Chervonenkis [66] en 1998 et finalement par Smola et Schölkopf [62] en 2004. Nous décrivons principalement la méthode des SVR pour l'approximation de fonctions non-linéaires à l'aide de la méthode des SVR utilisant des noyaux qui donne de très bons résultats (voir Chapitre 6). Ces résultats sont justifiés encore une fois grâce au lien direct entre la théorie de l'apprentissage statique et l'algorithme d'apprentissage des SVR. Ce lien est développé à la fin de ce chapitre.*

4.1 La régression par vecteurs de support (SVR)

Dans cette section, nous nous intéressons aux SVM pour la régression qui permettent de traiter le problème de la régression défini à la Section 2.5. Soit l'ensemble d'apprentissage,

$$\mathcal{S} = \{(x_i, y_i) \text{ tels que } 1 \leq i \leq n,\ x_i \in \mathbb{R}^p \text{ et } y_i \in \mathbb{R}\},$$

les SVM pour la régression ont pour but d'approximer ces données à l'aide d'un modèle f. La description générale du principe de construction d'un modèle linéaire par les SVR est donnée ci-dessous. Nous introduisons ensuite l'astuce du noyau afin d'étendre les SVR au cas non-linéaire.

4.1.1 Le cas linéaire

Comme introduit ci-dessus, les SVR ont pour but de trouver la fonction linéaire f qui soit en plus "la plus plane possible". Celle-ci est définie comme

précédemment par

$$f(x) = \langle w, x \rangle + b,$$

où les paramètres w et b sont à déterminer de telle manière que la déviation [1] par rapport aux données de l'ensemble d'apprentissage soit d'au plus $\varepsilon > 0$, c'est-à-dire que

$$|y_i - f(x_i)| \leq \varepsilon, \quad \forall i : 1 \leq i \leq n. \tag{4.1}$$

Comme la fonction linéaire la plus plane est la fonction constante, le problème est donc de trouver une fonction f qui vérifie (4.1) et dont la norme des poids w est minimum [2], ce qui nous mène au problème d'optimisation suivant

$$\begin{cases} \min & ||w||_2^2 \\ \text{s.c.} & |y_i - \langle w, x_i \rangle - b| \leq \varepsilon \quad \forall i : 1 \leq i \leq n. \end{cases} \tag{4.2}$$

Cette formulation du problème considère qu'il existe une fonction f qui approxime toutes les données avec une précision ε. Cependant, cela pourrait ne pas être le cas ou encore ce modèle f pourrait être confronté au problème du sur-apprentissage simplement dû au fait que certaines données de l'ensemble d'apprentissage peuvent être bruitées. C'est pourquoi il faut mettre en place une technique de détection d'outliers. En pratique, comme pour la classification, nous utilisons une relaxation des contraintes (4.1). Ce qui revient en fait à minimiser la fonction de coût d'insensibilité ε définie ci-dessous.

Définition 4.1.1 *(Fonction de coût d'insensibilité ε)*
La fonction de coût d'insensibilité ε, $l(f(x), y)$, est définie par

$$l(f(x), y) = |y - f(x)|_\varepsilon = \begin{cases} 0 & \text{si } |y - f(x)| \leq \varepsilon, \\ |y - f(x)| - \varepsilon & \text{sinon,} \end{cases} \tag{4.3}$$

où f est une fonction à valeur réelle, $x \in \mathbb{R}^p$ et $y \in \mathbb{R}$.

Comme le montre la Figure 4.1, cette fonction de coût construit un tube d'insensibilité ε. Les points se trouvant en dehors de ce tube sont alors pénalisés. La signification des variables ξ et ξ^* est quant à elle donnée à la Section 4.1.2.

En conclusion, le problème de la régression consiste à trouver la fonction linéaire f qui soit "la plus plane possible" et qui minimise la fonction de coût (4.3). Cependant, cette fonction de coût (4.3) n'est pas directement utilisable. Nous verrons à la Section 4.1.2 comment implémenter celle-ci à l'aide de variables de relaxation jouant le même rôle que dans le cas de la classification.

Remarques

1. L'hyperparamètre ε doit être fixé par l'utilisateur en fonction du niveau d'erreur et de bruit des données de l'ensemble d'apprentissage. Nous verrons comment fixer cet hyperparamètre au Chapitre 5.

1. La déviation ε est fixée par l'utilisateur et est donc un hyperparamètre.
2. Nous montrons à la Section 4.2 que la minimisation du vecteur de poids w implique la minimisation simultanée du risque empirique et de la complexité du modèle.

2. Il existe d'autres fonctions de coût que celle d'insensibilité ε pouvant être utilisées : quadratique, de Laplace ou encore de Huber. Cependant ces fonctions de coût ne permettent pas d'obtenir un ensemble de vecteurs de support clairsemé (voir Section 4.1.1), ni de détecter aussi aisément les erreurs dues au bruit et les outliers dans les données de l'ensemble d'apprentissage que la fonction de coût d'insensibilité ε. Pour de plus amples informations, le lecteur se référera à [30].

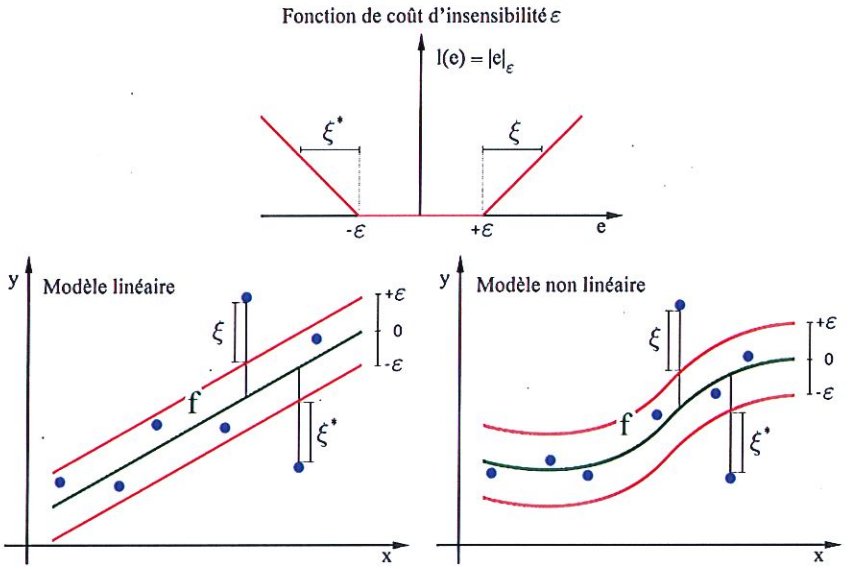


FIGURE 4.1 – La fonction de coût pour $e = y - f(x)$ et le tube d'insensibilité ε pour un modèle linéaire et non-linéaire f.

4.1.2 Le cas non-linéaire

Le développement de la méthode SVR étant similaire à celui étudié au Chapitre 3 pour la classification, nous considérons directement le problème quadratique pour un modèle non-linéaire.

4.1.2.1 Le problème quadratique

Dans le cas de la construction d'un modèle non-linéaire, les données de l'ensemble d'apprentissage sont transformées à l'aide d'une application Φ dans un espace de redescription (voir Section 2.4). Le modèle non-linéaire est alors donné par

$$f(x) = \langle w, \Phi(x) \rangle + b. \tag{4.4}$$

Dans la méthode SVR, l'implémentation de la fonction de coût d'insensibilité ε (4.3) est réalisée à l'aide de variables de relaxation ξ_i et ξ_i^* ajoutées aux contraintes (4.1) et la norme euclidienne des poids w est minimisée dans le but

de maximiser la planitude du modèle f. Le problème quadratique de la méthode SVR est alors donné comme suit

$$\begin{cases} \min & \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{s.c.} & y_i - \langle w, \Phi(x_i)\rangle - b \leq \varepsilon + \xi_i \quad \forall i : 1 \leq i \leq n, \\ & \langle w, \Phi(x_i)\rangle + b - y_i \leq \varepsilon + \xi_i^* \quad \forall i : 1 \leq i \leq n, \\ & \xi_i \geq 0, \xi_i^* \geq 0 \quad \forall i : 1 \leq i \leq n, \end{cases} \tag{4.5}$$

où :

- la constante $C > 0$ détermine le compromis entre la planitude et le nombre d'erreurs tolérées (i.e., le compromis entre le risque empirique et la complexité du modèle). Nous verrons au Chapitre 5 comment sélectionner cet hyperparamètre.

- Les deux contraintes sur les données x_i assurent que l'erreur de prédiction du modèle optimal se situe dans l'intervalle $[-\varepsilon, \varepsilon]$. Les variables de relaxation ξ_i et ξ_i^* permettent cependant de pouvoir relaxer certaines contraintes.

La Figure 4.1 illustre les variables de relaxation ξ et ξ^* pour deux données de l'ensemble d'apprentissage. Ces variables permettent de mesurer les déviations plus grandes que $\pm\varepsilon$ des données de l'ensemble d'apprentissage.

Remarque

Smola montre dans [64] que choisir une fonction la plus plane possible dans l'espace de redescription donne la fonction la plus lisse dans l'espace d'entrée.

Tout comme pour la classification, nous considérons le problème dual du problème d'optimisation (4.5). Celui-ci est donné grâce à la méthode des *multiplicateurs de Lagrange* où le *Lagrangien* est donné comme suit (voir Théorème 3.2.1)

$$\begin{aligned} L(w, b, \xi_i, \xi_i^*) &= \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) - \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + \langle w, \Phi(x_i)\rangle + b) \\ &\quad - \sum_{i=1}^{n}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle w, \Phi(x_i)\rangle - b) - \sum_{i=1}^{n}(\eta_i\xi_i + \eta_i^*\xi_i^*), \end{aligned} \tag{4.6}$$

où les multiplicateurs de Lagrange η_i, η_i^* et α_i, $\alpha_i^* \geq 0$ sont associés respectivement aux contraintes de positivité de ξ_i, ξ_i^* et aux contraintes sur les données d'apprentissage. Déterminons à présent les conditions de KKT [3] du problème d'optimisation (4.5). Annulons pour cela les dérivées partielles du Lagrangien

3. Pour être complet il faudrait aussi donner les conditions de complémentarité de KKT mais dans un premier temps celles-ci ne nous intéressent pas.

par rapport aux variables primales w, b, ξ_i et ξ_i^*. Dès lors, nous obtenons

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\Phi(x_i) = 0, \tag{4.7}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0, \tag{4.8}$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \eta_i = 0, \quad \forall i : 1 \leq i \leq n, \tag{4.9}$$

$$\frac{\partial L}{\partial \xi_i^*} = C - \alpha_i^* - \eta_i^* = 0, \quad \forall i : 1 \leq i \leq n. \tag{4.10}$$

De plus, remarquons que

$$\frac{\partial L}{\partial \eta_i} = \sum_{i=1}^{n}\xi_i = 0, \tag{4.11}$$

$$\frac{\partial L}{\partial \eta_i^*} = \sum_{i=1}^{n}\xi_i^* = 0. \tag{4.12}$$

Ces deux égalités seront nécessaires par la suite afin de simplifier le Lagrangien. Les équations (4.9) et (4.10) permettent de supprimer les variables

$$\eta_i = C - \alpha_i \quad \text{et} \quad \eta_i^* = C - \alpha_i^*.$$

Finalement, par (4.7), le paramètre w est donné par

$$w = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\Phi(x_i). \tag{4.13}$$

Substituons maintenant les égalités obtenues (4.7)-(4.10) dans l'équation du Lagrangien (4.6). Nous obtenons alors le nouveau Lagrangien

$$\begin{aligned} L(w, b, \xi_i, \xi_i^*) = & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle\Phi(x_i), \Phi(x_j)\rangle \\ & - \sum_{i=1}^{n}((C - \alpha_i)\xi_i + (C - \alpha_i^*)\xi_i^*) \\ & - \sum_{i=1}^{n}\alpha_i\left(-y_i + \left(\sum_{j=1}^{n}(\langle\Phi(x_i), \Phi(x_j)\rangle(\alpha_j - \alpha_j^*)) + b\right) + \varepsilon + \xi_i\right) \\ & - \sum_{i=1}^{n}\alpha_i^*\left(y_i - \left(\sum_{j=1}^{n}(\langle\Phi(x_i), \Phi(x_j)\rangle(\alpha_j - \alpha_j^*)) + b\right) + \varepsilon + \xi_i\right) \end{aligned} \tag{4.14}$$

qui peut encore être récrit comme

$$\begin{aligned} L(w,b,\xi_i,\xi_i^*) &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\langle\Phi(x_i),\Phi(x_j)\rangle \\ &\quad -\sum_{i=1}^{n}(C-\alpha_i)\xi_i-\sum_{i=1}^{n}(C-\alpha_i^*)\xi_i^* \\ &\quad -\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i(\alpha_j-\alpha_j^*)\langle\Phi(x_i),\Phi(x_j)\rangle \\ &\quad +\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i^*(\alpha_j-\alpha_j^*)\langle\Phi(x_i),\Phi(x_j)\rangle \\ &\quad +\sum_{i=1}^{n}\alpha_i y_i-\sum_{i=1}^{n}\alpha_i^* y_i \\ &\quad -\sum_{i=1}^{n}\alpha_i\varepsilon-\sum_{i=1}^{n}\alpha_i^*\varepsilon \\ &\quad -\sum_{i=1}^{n}\alpha_i\xi_i-\sum_{i=1}^{n}\alpha_i^*\xi_i^*. \end{aligned} \tag{4.15}$$

En regroupant les deux monômes et en simplifiant les termes semblables, nous obtenons

$$\begin{aligned} L(w,b,\xi_i,\xi_i^*) &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\langle\Phi(x_i),\Phi(x_j)\rangle \\ &\quad +\sum_{i=1}^{n}y_i(\alpha_i-\alpha_i^*) \\ &\quad -\sum_{i=1}^{n}\varepsilon(\alpha_i+\alpha_i^*) \\ &\quad -\sum_{i=1}^{n}(C\xi_i-\alpha_i\xi_i+\alpha_i\xi_i) \\ &\quad -\sum_{i=1}^{n}(C\xi_i^*-\alpha_i^*\xi_i^*+\alpha_i^*\xi_i^*). \end{aligned} \tag{4.16}$$

Finalement, en supprimant les termes semblables et par (4.11) et (4.12), nous avons le Lagrangien final

$$\begin{aligned} L(w,b,\xi_i,\xi_i^*) &= -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\langle\Phi(x_i),\Phi(x_j)\rangle \\ &\quad -\varepsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*)+\sum_{i=1}^{n}y_i(\alpha_i-\alpha_i^*). \end{aligned} \tag{4.17}$$

Le problème dual est alors obtenu en maximisant le Lagrangien (4.17) par rap-

port aux variables duales

$$\begin{cases} \max & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)k(x_i,x_j) \\ & -\varepsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*)+\sum_{i=1}^{n}y_i(\alpha_i-\alpha_i^*) \\ \text{s.c.} & \sum_{i=1}^{n}(\alpha_i-\alpha_i^*)=0, \\ & 0\le\alpha_i\le C \quad \forall i: 1\le i\le n, \\ & 0\le\alpha_i^*\le C \quad \forall i: 1\le i\le n, \end{cases} \tag{4.18}$$

où la fonction noyau

$$k(x_i,x_j)=\langle\Phi(x_i),\Phi(x_j)\rangle$$

a été introduite pour remplacer le produit scalaire (voir Section 2.4). Comme pour la classification, les noyaux sont utilisés afin de traiter les problèmes non-linéaires et éviter le problème de la malédiction de la dimension. Afin de calculer la constante b, nous utilisons les conditions de KKT qui déclarent que les produits entre les variables duales et les contraintes s'annulent à la solution. Nous pouvons donc maintenant écrire

$$\alpha_i(\varepsilon+\xi_i-y_i+\langle w,\Phi(x_i)\rangle+b) = 0 \tag{4.19}$$
$$\alpha_i^*(\varepsilon+\xi_i^*+y_i-\langle w,\Phi(x_i)\rangle-b) = 0 \tag{4.20}$$
$$(C-\alpha_i)\xi_i = 0 \tag{4.21}$$
$$(C-\alpha_i^*)\xi_i^* = 0 \tag{4.22}$$

qui sont les conditions de complémentarité de KKT. A partir de ces égalités, la variable b peut facilement être calculée. En effet, pour un couple particulier de données (x_i,y_i), lorsque le problème d'optimisation (4.18) est résolu, la variable ξ_i est connue et α_i et α_i^* sont donnés par la solution au problème dual d'optimisation (4.18) . Nous étudions au Chapitre 5 l'algorithme "sequential minimal optimization" (SMO) qui permet de déterminer ces multiplicateurs de Lagrange. Il suffit donc de combiner (4.19) et (4.20) avec (4.13) et par (4.21) et (4.22), il s'ensuit que

$$b=y_i-\sum_{j=1}^{n}(\alpha_j-\alpha_j^*)k(x_i,x_j)+\varepsilon \quad \text{si } 0<\alpha_i<C, \tag{4.23}$$

et

$$b=y_i-\sum_{j=1}^{n}(\alpha_j-\alpha_j^*)k(x_i,x_j)-\varepsilon \quad \text{si } 0<\alpha_i^*<C. \tag{4.24}$$

D'où, par (4.23) et (4.24), nous pouvons calculer la constante b à partir d'un ou plusieurs α_i ou α_i^* compris entre 0 et C. Nous verrons au Chapitre 5 comment calculer de manière unique la valeur de la constante b.

Finalement, en substituant (4.13) dans (4.4) et en remplaçant le produit scalaire $\langle\Phi(x),\Phi(x_i)\rangle$ par la fonction noyau, le modèle SVR non-linéaire est donné par

$$f(x)=\sum_{i=1}^{n}(\alpha_i-\alpha_i^*)k(x_i,x)+b. \tag{4.25}$$

Remarques

1. Notons que par les égalités (4.21) et (4.22), seules les données pour lesquelles les multiplicateurs de Lagrange satisfont à

$$\alpha_i = C \quad \text{ou} \quad \alpha_i^* = C$$

peuvent se situer à l'extérieur du tube d'insensibilité ε (i.e., ξ_i et $\xi_i^* \neq 0$). De plus, pour les données se situant à l'intérieur du tube, c'est-à-dire vérifiant

$$|y_i - f(x_i)| < \varepsilon \quad \text{et} \quad \xi_i, \xi_i^* = 0,$$

les conditions (4.19) et (4.20) impliquent que les variables α_i et α_i^* sont égales à zéro. Nous pouvons donc conclure que pour calculer le modèle (4.25), seules sont nécessaires les données dont les variables α_i et α_i^* sont non nulles. Ces données sont appelées, comme pour la classification, *vecteurs de support*, et forment donc un sous-ensemble clairsemé [4] de l'ensemble des données d'apprentissage.

2. Transformer le problème d'optimisation dans sa forme duale présente un avantage important. En effet, le problème d'optimisation dual est un problème d'optimisation quadratique avec contraintes linéaires et la matrice Hessienne est définie positive à condition que la matrice noyau

$$K = (k(x_i, x_j))_{i,j=1}^n$$

le soit aussi. C'est bien le cas dans ce mémoire puisque nous ne considérons que les noyaux définis à la Section 2.4.2. Dans ce cas, nous avons une solution optimale unique.

3. Les données de l'ensemble d'apprentissage apparaissent seulement à travers la fonction noyau $k(x_i, x_j)$. Le noyau de chaque paire de données étant un scalaire, il peut être précalculé [5] et stocké dans la matrice noyau K. De plus, de cette façon, la dimension de l'espace d'entrée n'intervient pas dans le problème d'optimisation évitant ainsi la malédiction de la dimension.

4. D'autres formulations pour la méthode SVR se basant sur les fonctions de coût quadratique ou de Huber, illustrées à la Figure 4.2, ont été proposées [30]. Par exemple, l'utilisation de la fonction de coût quadratique permet de ramener le problème quadratique des SVR à un système d'équations linéaires. Cependant, comme cité précédemment, ces méthodes sont sensibles aux outliers et aux erreurs sur les données de l'ensemble d'apprentissage bien que des procédures complexes aient été proposées pour palier à ce problème. C'est pourquoi ces méthodes ne sont pas étudiées dans ce mémoire. Finalement, une modification du problème quadratique présentée dans ce chapitre, appelée ν-SVR, permettant d'adapter automatiquement l'hyperparamètre ε (i.e., le minimiser) a été proposée dernièrement. Cette

4. Nous n'avons pas besoin de toutes les données x_i pour calculer le modèle f.
5. Ce qui représente un gain de temps important d'un point de vue temps de calcul algorithmique (voir Chapitre 5).

méthode se base sur le niveau de bruit présent dans les données d'apprentissage. Cependant, dans la plupart des applications industrielles, le niveau de bruit est inconnu. Il faut alors introduire deux nouveaux hyperparamètres : ν qui détermine la largeur du tube d'insensibilité ε optimisé par la méthode SVR et le paramètre de régularisation λ qui règle le compromis entre la régularité de la fonction de régression et l'erreur. L'implémentation de cette méthode est assez complexe notamment du fait que ces deux hyperparamètres doivent de nouveau être déterminés. Cette méthode, dépassant le cadre de ce mémoire, n'est pas étudiée. Le lecteur peut se référer à [62, 64] pour de plus amples informations.

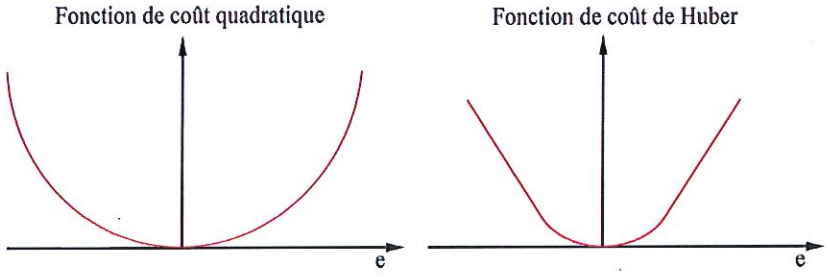


FIGURE 4.2 – Les fonctions de coût quadratique et de Huber.

4.2 Relation avec l'apprentissage statique

La théorie de l'apprentissage statique a été développée au départ pour la classification. Cependant, elle peut aussi être appliquée à la régression afin de développer une borne sur l'erreur de généralisation, similaire à (3.19), pour les modèles à valeurs réelles. Dans cette section, nous nous basons principalement sur [16, 66] afin de déduire les deux bornes principales de la théorie de l'apprentissage statique.

Soit le carré de la fonction de coût

$$l(f(x), y) = (y - f(x))^2,$$

la borne suivante lie le risque fonctionnel $R[f]$ au risque empirique $E_{\mathcal{S}}[f]$ avec une probabilité de $1 - \delta$

$$R[f] \leq E_{\mathcal{S}}[f] \left(1 - c\sqrt{\frac{h(ln(an/h) + 1 - ln(\delta)}{n}}\right)_+^{-1}, \tag{4.26}$$

où :

- n est le nombre de données de l'ensemble d'apprentissage $\mathcal{S}$,
- h est la dimension VC de l'ensemble $\mathcal{F}$ des fonctions admissibles f,
- c est une constante représentant la probabilité d'observer de grandes valeurs du coût,

- a est une constante théorique qui peut être fixée proche de 1 [66],
- $(z)_+$ est égal à z si $z > 0$ et 0 sinon,
- $\delta > 0$ est défini par le théorème 2.4 de [66].

La borne (4.26) décroît avec la complexité du modèle, mesurée par la dimension VC, et lorsque le nombre de données n augmente.

Nous allons maintenant développer une deuxième borne pour la fonction de coût d'insensibilité ε à partir de la borne basée sur la marge pour la classification (3.19). L'idée principale est de considérer une erreur de test (voir Section 2.3) si l'erreur sur une donnée test est plus grande qu'une certaine valeur test θ. Le but est alors de borner la probabilité qu'une donnée test tirée aléatoirement ait une erreur plus petite que θ. Nous pouvons définir une marge (jouant le même rôle que la marge d'un classificateur) pour la régression

$$\gamma = \theta - \varepsilon.$$

Cette marge mesure la différence permise entre la précision test θ et la précision ε et implique qu'une donnée est considérée comme un outlier si son erreur est plus grande que $\theta - \gamma$. En d'autres mots, nous considérons une bande de taille $\pm(\theta - \gamma)$ autour d'un modèle f. Toutes les données se trouvant en dehors de cette bande sont considérées comme des erreurs d'entraînement (i.e., outliers) tandis que des données tests sont considérées comme des erreurs si et seulement si elles se situent en dehors de la bande de taille $\pm\theta$. Vapnik [66] montre alors que nous avons la borne suivante sur le risque fonctionnel avec une probabilité de $1 - \delta$

$$R_\theta[f] \leq \frac{c}{n}\left(\frac{||w||_2^2 R^2 + ||\xi||_1^2 log_2(1/\gamma)}{\gamma^2} log_2^2(n) + log_2\left(\frac{1}{\delta}\right)\right), \tag{4.27}$$

où :

- $R_\theta[f]$ correspond au risque fonctionnel d'un modèle f de fonction de coût d'insensibilité θ,
- ξ est un vecteur contenant toutes les variables de relaxation,
- n est le nombre de données de l'ensemble d'apprentissage $\mathcal{S}$,
- R est le rayon de la boule minimum englobant les données dans l'espace d'entrée $\mathcal{X}$ (i.e., $\forall x \in \mathcal{X} : ||\Phi(x)||_2 = \sqrt{\langle \Phi(x), \Phi(x) \rangle} \leq R$).

Cette borne décroît lorsque le nombre de données n ou la marge γ croît et permet surtout de vérifier que minimiser $||w||_2$ permet de contrôler la capacité d'un modèle f.

Chapitre 5

Implémentation d'un algorithme SVR et l'algorithme SMO

RÉSUMÉ. *Dans ce chapitre, nous présentons le fonctionnement général d'un algorithme SVR ainsi que différents points techniques et astuces. Nous introduisons ensuite différents algorithmes d'optimisation qui permettent de résoudre le problème quadratique des SVR. Nous étudions plus particulièrement l'algorithme Sequential Minimal Optimization (SMO) introduit par John C. Platt pour la classification [55] et adpaté à la régression par Alex J. Smola et Bernhard Schölkopf [11]. Comme nous le verrons au Chapitre 6, les performances de cet algorithme dépendent fortement du choix des hyperparamètres. C'est pourquoi nous présentons une technique de sélection des hyperparamètres basée sur la validation croisée. Ce chapitre se base principalement sur [11].*

5.1 Le fonctionnement général d'un algorithme SVR

Avant d'étudier en détail l'algorithme SMO et son implémentation, nous rappelons brièvement les propriétés importantes d'un algorithme SVR ainsi que son déroulement.

Les différentes étapes d'un algorithme SVR se présentent comme suit. Premièrement, un ensemble de n données d'apprentissage est généré à l'aide d'une technique de plan d'expériences propre à Cenaero (voir Chapitre 1). Deuxièmement, ces données d'apprentissage sont transformées dans un espace de redescription, ce qui correspond à calculer la matrice noyau

$$K = (k(x_i, x_j))_{i,j=1}^{n}.$$

Il faut donc pour cela sélectionner un noyau k. Dans le cadre de ce mémoire, nous avons testé les 4 noyaux proposés à la Section 3.4 de manière empirique,

c'est-à-dire que les noyaux sont simplement testés les uns après les autres. Notons au passage que des méthodes de sélection automatique du noyau existent. Après nos différents tests [1], nous avons remarqué que le noyau RBF Gaussien est celui qui donne en général de meilleurs résultats (voir Chapitre 6). C'est pourquoi nous travaillons uniquement avec ce noyau et ne nous intéressons pas aux méthodes de sélection automatique. Par la suite, nous verrons que la résolution du problème d'optimisation quadratique (4.18) permet de déterminer les multiplicateurs de Lagrange. Nous présentons dans ce chapitre différentes classes d'algorithmes d'optimisation existantes permettant de résoudre ce problème (voir Section 5.2). Nous étudions plus particulièrement l'algorithme SMO (voir Section 5.2.4) que nous avons décidé d'utiliser et d'implémenter dans Minamo. La différence des multiplicateurs de Lagrange est ensuite utilisée, avec les données d'apprentissage, pour calculer les paramètres w et b. Ces deux paramètres sont déterminés respectivement par (4.13) et (4.23) ou (4.24). Finalement, en substituant w et b dans (4.25) nous obtenons le modèle final f qui peut être calculé pour n'importe quelle nouvelle donnée x. La Figure 5.1 illustre les différentes étapes d'un algorithme SVR.

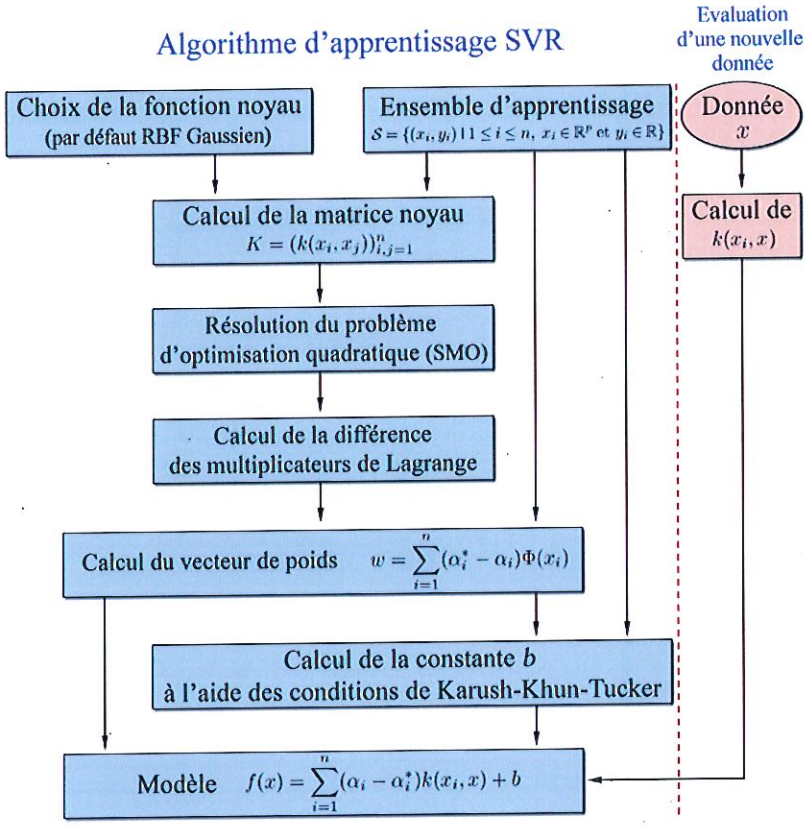


FIGURE 5.1 – Organigramme d'un algorithme SVR.

1. Les différentes fonctions utilisées pour les différents tests sont présentées au Chapitre 6.

5.2 Les différents algorithmes d'optimisation

Nous présentons dans cette section les principaux algorithmes les plus fiables permettant de résoudre le problème des SVM pour la régression pour des ensembles d'apprentissage de taille moyenne (n de l'ordre de 10^3) [2].

5.2.1 L'algorithme de points intérieurs

Les algorithmes de points intérieurs font partie des algorithmes d'optimisation les plus fiables et précis. Ils sont tout particulièrement adaptés pour des problèmes de petite taille. L'idée principale des algorithmes de points intérieurs est de calculer le dual du problème d'optimisation et de résoudre les deux problèmes (primal et dual) simultanément. La résolution se fait alors en vérifiant progressivement les conditions de KKT afin de trouver itérativement une solution admissible. Finalement, il suffit d'utiliser le saut de dualité entre la fonction objectif primale et duale pour déterminer la qualité du modèle. Pour de plus amples informations générales, le lecteur peut se référer à [62, 63] et à [29] pour une implémentation et une étude de la convergence.

5.2.2 L'algorithme du gradient

La plupart des méthodes permettant de résoudre le problème quadratique des SVR se basent sur le problème dual d'optimisation. Cependant, nous pouvons nous demander s'il est possible de considérer une approche s'appuyant sur le problème primal d'optimisation donnant de bons résultats. En effet, il existe plusieurs méthodes découlant de cette approche et basées sur la méthode du gradient. Nous pouvons par exemple citer l'algorithme AdaTron dérivé du célèbre algorithme du Perceptron [16]. Cependant, ce type de méthodes étant encore très peu développées, nous ne les étudions pas. Pour de plus amples informations, le lecteur peut se référer à [62, 33].

5.2.3 L'algorithme de sélection de sous-ensembles

Les algorithmes précédents peuvent être utilisés, sans modification, sur des ensembles d'apprentissage de taille modérée. Historiquement, ces algorithmes ont été créés pour la classification et ont très rapidement atteint leurs limites. En effet, à l'heure actuelle il n'est pas rare de travailler avec des ensembles d'apprentissage de plus de 50000 données, ce qui implique logiquement des problèmes conséquents de mémoire et de CPU. Il a donc fallu inventer de nouveaux algorithmes permettant de contourner ces problèmes : les algorithmes de sélection de sous-ensembles. Ceux-ci ont ensuite été transposés avec succès au problème de la régression.

2. Les problèmes dont la taille de l'ensemble d'apprentissage est extrême (i.e. l'ordre de n est plus grand que 10^3) sont très complexes à résoudre. En effet, l'algorithme peut prendre un temps considérable pour résoudre le problème quadratique ou encore il est impossible de stocker la matrice K. Il faut alors avoir recours à différentes astuces et approximations. Pour de plus amples informations le lecteur peut se référer à [62].

5.2.3.1 Le chunking

La première méthode de sélection de sous-ensembles proposée par Vapnik en 1982 [66] se base sur l'observation que seuls les vecteurs de support sont importants pour le modèle final. En d'autres mots, si nous connaissons et utilisons uniquement les vecteurs de support pour calculer le modèle f, celui-ci est identique à celui obtenu en utilisant toutes les données de l'ensemble d'apprentissage. Donc, si nous connaissons les vecteurs de support, cela permet un gain énorme de mémoire. De plus, nous pouvons directement calculer le modèle avec ces seules données et ainsi traiter n'importe quel ensemble d'apprentissage. Le problème est que nous ne connaissons pas les vecteurs de support avant d'avoir résolu le problème quadratique (4.18). La solution proposée est alors simplement de stocker un sous-ensemble arbitraire de l'ensemble d'apprentissage, appelé chunk, et de calculer le modèle f avec ce chunk. Il suffit alors de garder uniquement dans le chunk les vecteurs de support, de le compléter avec le reste des données et de recalculer le modèle f. La procédure est alors répétée jusqu'à ce que toutes les données aient été utilisées et que les conditions de KKT soient satisfaites.

Historiquement, l'algorithme basique présenté ci-dessus a d'abord été développé pour la classification. Il a ensuite été amélioré par Osuna et al. en 1997 [53] et finalement par Thorsten Joachims en 1999 [36] qui a en plus transposé cette méthode à la régression. Pour de plus amples informations, le lecteur peut se référer à [62, 11].

5.2.3.2 L'algorithme SMO

L'algorithme SMO a été proposé pour la première fois par John Platt en 1999 [55] pour la classification. Cet algorithme se base sur la méthode du chunking. L'idée est de sélectionner des sous-ensembles de dimension 2 et d'optimiser la fonction objectif tout en satisfaisant les contraintes du problème d'optimisation (4.18). L'algorithme SMO présente un avantage important par rapport aux autres algorithmes : il peut gérer des ensembles d'apprentissage de très grande dimension. Le point clé de cet algorithme réside dans le fait que pour un sous-ensemble de taille 2, le sous-problème d'optimisation peut être résolu analytiquement sans devoir recourir à la résolution numérique de problèmes quadratiques. L'algorithme SMO est l'algorithme que nous avons choisi d'implémenter parce qu'il présente de bonnes performances (voir Chapitre 6) et les preuves de convergence ont été démontrées [52]. L'entreprise Cenaero pouvant être amenée dans le futur à traiter des problèmes de grande taille, l'algorithme SMO est donc tout indiqué. L'algorithme SMO pour la régression que nous présentons dans cette section est dérivé de l'algorithme SMO de Platt [55, 56] en suivant simplement le même raisonnement. Le pseudo-code de l'algorithme est donné à titre indicatif à la Section 5.2.4.2.

Le raisonnement proposé ci-après pour implémenter l'algorithme SMO peut uniquement être appliqué au problème des SVR utilisant la fonction de coût d'insensibilité ε. Pour la majorité des autres fonctions de coût convexes, trouver une solution explicite du problème quadratique correspondant est impossible. Nous pourrions bien sûr dériver un problème d'optimisation convexe non-quadratique analogue mais nous devrions le résoudre numériquement.

Développement de l'algorithme SMO

Considérons le problème d'optimisation quadratique (4.18) défini au Chapitre 4

$$\begin{cases} \max & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)K_{ij} \\ & -\varepsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*)+\sum_{i=1}^{n}y_i(\alpha_i-\alpha_i^*) \\ \text{s.c.} & \sum_{i=1}^{n}(\alpha_i-\alpha_i^*)=0, \\ & 0\le\alpha_i\le C \quad \forall i: 1\le i\le n, \\ & 0\le\alpha_i^*\le C \quad \forall i: 1\le i\le n, \end{cases} \tag{5.1}$$

pour deux indices, notons les (i,j). Avant de donner le problème d'optimisation restreint aux deux indices (i,j) et de trouver la solution analytique, nous devons introduire plusieurs concepts importants.

Le but de l'algorithme SMO est de résoudre le problème d'optimisation (5.1). Pour cela, l'idée est de forcer les données d'apprentissage correspondant aux indices (i,j) à se trouver à l'intérieur du tube d'insensibilité ε et de vérifier les conditions de KKT. Comme expliqué auparavant, si les données se situent dans le tube d'insensibilité ε alors les variables α_i ou α_i^* et α_j ou α_j^* correspondantes sont égales à zéro. Ce qui implique que

$$\alpha_i\alpha_i^*=0 \quad \text{et} \quad \alpha_j\alpha_j^*=0. \tag{5.2}$$

De (5.2), nous déduisons 4 cas possibles à considérer pour le problème d'optimisation à deux variables :

- **Cas 1** : (α_i,α_j) qui correspond à $\alpha_i,\ \alpha_j\neq 0$ et $\alpha_i^*,\ \alpha_j^*=0$,
- **Cas 2** : (α_i,α_j^*) qui correspond à $\alpha_i,\ \alpha_j^*\neq 0$ et $\alpha_i^*,\ \alpha_j=0$,
- **Cas 3** : (α_i^*,α_j) qui correspond à $\alpha_i^*,\ \alpha_j\neq 0$ et $\alpha_i,\ \alpha_j^*=0$,
- **Cas 4** : (α_i^*,α_j^*) qui correspond à $\alpha_i^*,\ \alpha_j^*\neq 0$ et $\alpha_i,\ \alpha_j=0$.

De plus, par la contrainte de sommation $\sum_{i=1}^{n}(\alpha_i-\alpha_i^*)=0$, nous avons que

$$(\alpha_i-\alpha_i^*)+(\alpha_j-\alpha_j^*)=\gamma \quad \text{où} \quad \gamma=-\sum_{\substack{q=1\\ q\neq i,j}}^{n}(\alpha_q-\alpha_q^*) \tag{5.3}$$

et avant optimisation (i.e. résolution analytique du problème d'optimisation pour deux indices)

$$\gamma=(\alpha_i^{old}-\alpha_i^{*old})+(\alpha_j^{old}-\alpha_j^{*old}). \tag{5.4}$$

Grâce à (5.3), nous pouvons déduire une borne supérieure et inférieure pour chaque $\alpha_i,\ \alpha_i^*$ et ce pour les 4 cas de telle manière que les contraintes d'inégalités du problème d'optimisation (5.1) soient satisfaites.

Soit

$$(\alpha_i - \alpha_i^*) = \gamma - (\alpha_j - \alpha_j^*) \tag{5.5}$$

alors,

- **Cas 1 :** (α_i, α_j)

  Par (5.5) nous avons que

  $$\alpha_i = \gamma - \alpha_j$$

  et par (5.1)

  $$0 < \alpha_i < C \quad \text{et} \quad 0 < \alpha_j < C$$

  donc

  $$\begin{aligned} 0 < \alpha_i < C &\Leftrightarrow 0 < \gamma - \alpha_j < C \\ &\Leftrightarrow -\gamma < -\alpha_j < C - \gamma \\ &\Leftrightarrow \gamma - C < \alpha_j < \gamma \end{aligned}$$

  d'où, avec $0 < \alpha_j < C$,

  $$L < \alpha_j < H$$

  avec

  $$\begin{aligned} L &= \max(0, \gamma - C), \\ H &= \min(C, \gamma). \end{aligned}$$

- **Cas 2 :** (α_i, α_j^*)

  Par (5.5) nous avons que

  $$\alpha_i = \alpha_j^* + \gamma$$

  et par (5.1)

  $$0 < \alpha_i < C \quad \text{et} \quad 0 < \alpha_j^* < C$$

  d'où

  $$\begin{aligned} 0 < \alpha_i < C &\Leftrightarrow 0 < \alpha_j^* + \gamma < C \\ &\Leftrightarrow -\gamma < \alpha_j^* < C - \gamma \end{aligned}$$

  d'où, avec $0 < \alpha_j^* < C$,

  $$L < \alpha_j^* < H$$

  avec

  $$\begin{aligned} L &= \max(0, -\gamma) \\ H &= \min(C, C - \gamma). \end{aligned}$$

- **Cas 3** : (α_i^*, α_j)

  Par (5.5) nous avons que
  $$\alpha_i^* = \alpha_j - \gamma$$
  et par (5.1)
  $$0 < \alpha_i^* < C \quad \text{et} \quad 0 < \alpha_j < C$$
  d'où
  $$\begin{aligned} 0 < \alpha_i^* < C &\Leftrightarrow 0 < \alpha_j - \gamma < C \\ &\Leftrightarrow \gamma < \alpha_j < C + \gamma \end{aligned}$$
  d'où, avec $0 < \alpha_j < C$,
  $$L < \alpha_j < H$$
  avec
  $$\begin{aligned} L &= \max(0, \gamma) \\ H &= \min(C, C + \gamma). \end{aligned}$$

- **Cas 4** : (α_i^*, α_j^*)

  Par (5.5) nous avons que
  $$\alpha_i^* = -\gamma - \alpha_j^*$$
  et par (5.1)
  $$0 < \alpha_i^* < C \quad \text{et} \quad 0 < \alpha_j^* < C$$
  d'où
  $$\begin{aligned} 0 < \alpha_i^* < C &\Leftrightarrow 0 < -\gamma - \alpha_j^* < C \\ &\Leftrightarrow \gamma < -\alpha_j^* < C + \gamma \\ &\Leftrightarrow -\gamma - C < \alpha_j^* < -\gamma \end{aligned}$$
  d'où, avec $0 < \alpha_j^* < C$,
  $$L < \alpha_j^* < H$$
  avec
  $$\begin{aligned} L &= \max(0, -\gamma - C) \\ H &= \min(C, -\gamma). \end{aligned}$$

La table ci-dessous reprend les bornes L et H pour chacun des quatre cas.

	α_j	α_j^*
α_i	$L = \max(0, \gamma - C)$ $H = \min(C, \gamma)$	$L = \max(0, -\gamma)$ $H = \min(C, C - \gamma)$
α_i^*	$L = \max(0, \gamma)$ $H = \min(C, C + \gamma)$	$L = \max(0, -\gamma - C)$ $H = \min(C, -\gamma)$

Tableau 5.1 – Bornes L et H pour la régression.

Remarque

Dans l'étude des bornes L et H des 4 cas, nous considérons uniquement les inégalités strictes

$$0 < \alpha_i < C \quad \text{et} \quad 0 < \alpha_i^* < C$$

et

$$0 < \alpha_j < C \quad \text{et} \quad 0 < \alpha_j^* < C.$$

La raison pour laquelle α_i, α_i^*, α_j et α_j^* doivent être différentes de 0 et de C vient du fait que nous exigeons que les données correspondantes se trouvent à l'intérieur du tube d'insensibilité ε. D'où, par (4.21) et (4.22), si α_i, α_i^*, α_j et $\alpha_j^* \neq 0$ et $\neq C$ alors les variables de relaxation ξ_i, ξ_i^*, ξ_j et ξ_j^* sont nulles. Il s'ensuit que les données se situent bien dans le tube d'insensibilité ε.

Résolution analytique du problème d'optimisation

L'étape suivante consiste à résoudre analytiquement le problème d'optimisation pour deux variables d'indices (i, j). Pour commencer nous définissons

$$v_i \overset{déf}{=} y_i - \sum_{q \neq i,j} (\alpha_q - \alpha_q^*) K_{iq} + b \tag{5.6}$$

$$= \phi_i + (\alpha_i^{old} - \alpha_i^{*old}) K_{ii} + (\alpha_j^{old} - \alpha_j^{*old}) K_{ij}, \tag{5.7}$$

avec

$$\phi_i = y_i - f(x_i).$$

De (5.7), nous déduisons une égalité qui est nécessaire pour la suite

$$v_i - v_j - \gamma(K_{ij} - K_{jj}) = \phi_i - \phi_j + (\alpha_i^{old} - \alpha_i^{*old})(K_{ii} + K_{jj} - 2K_{ij}). \tag{5.8}$$

Restreignons maintenant le problème (5.1) au problème d'optimisation à deux indices (i, j). Pour cela, nous repartons de (5.1) et nous prenons des nouveaux indices de sommation k et l afin de ne pas confondre avec les indices (i, j)

$$\begin{cases} \max & -\frac{1}{2} \sum_{k=1}^{n} \sum_{l=1}^{n} (\alpha_k - \alpha_k^*)(\alpha_l - \alpha_l^*) K_{kl} \\ & -\varepsilon \sum_{k=1}^{n} (\alpha_k + \alpha_k^*) + \sum_{k=1}^{n} y_k (\alpha_k - \alpha_k^*) \\ \text{s.c.} & \sum_{k=1}^{n} (\alpha_k - \alpha_k^*) = 0, \\ & 0 \leq \alpha_k \leq C \quad \forall k : 1 \leq k \leq n, \\ & 0 \leq \alpha_k^* \leq C \quad \forall k : 1 \leq k \leq n. \end{cases} \tag{5.9}$$

Commençons par développer la fonction objectif du problème d'optimisation (5.9) afin d'isoler les indices (i, j)

$$
\left\{
\begin{array}{ll}
\max & -\frac{1}{2}\sum_{k=1}^{n}\Bigg(\sum_{l\neq i,j}^{n}(\alpha_k-\alpha_k^*)(\alpha_l-\alpha_l^*)K_{kl} \\
& +(\alpha_k-\alpha_k^*)(\alpha_i-\alpha_i^*)K_{ki}+(\alpha_k-\alpha_k^*)(\alpha_j-\alpha_j^*)K_{kj}\Bigg) \\
& -\varepsilon\sum_{k=1}^{n}(\alpha_k+\alpha_k^*)+\sum_{k=1}^{n}y_k(\alpha_k-\alpha_k^*) \\
\text{s.c.} & \sum_{k=1}^{n}(\alpha_k-\alpha_k^*)=0, \\
& 0\leq\alpha_k\leq C \quad \forall k:1\leq k\leq n, \\
& 0\leq\alpha_k^*\leq C \quad \forall k:1\leq k\leq n.
\end{array}
\right. \tag{5.10}
$$

qui est encore équivalent à

$$
\left\{
\begin{array}{ll}
\max & -\frac{1}{2}\Bigg(\sum_{k\neq i,j}^{n}\Bigg(\sum_{l\neq i,j}^{n}(\alpha_k-\alpha_k^*)(\alpha_l-\alpha_l^*)K_{kl}+(\alpha_k-\alpha_k^*)(\alpha_i-\alpha_i^*)K_{ki} \\
& +(\alpha_k-\alpha_k^*)(\alpha_j-\alpha_j^*)K_{kj}\Bigg)+\sum_{l\neq i,j}^{n}(\alpha_i-\alpha_i^*)(\alpha_l-\alpha_l^*)K_{il} \\
& +(\alpha_i-\alpha_i^*)^2K_{ii}+(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)K_{ij} \\
& +\sum_{l\neq i,j}^{n}(\alpha_j-\alpha_j^*)(\alpha_l-\alpha_l^*)K_{jl}+(\alpha_j-\alpha_j^*)(\alpha_i-\alpha_i^*)K_{ji} \\
& +(\alpha_j-\alpha_j^*)^2K_{jj}\Bigg)-\varepsilon\sum_{k=1}^{n}(\alpha_k+\alpha_k^*)+\sum_{k=1}^{n}y_k(\alpha_k-\alpha_k^*) \\
\text{s.c.} & \sum_{k=1}^{n}(\alpha_k-\alpha_k^*)=0, \\
& 0\leq\alpha_k\leq C \quad \forall k:1\leq k\leq n, \\
& 0\leq\alpha_k^*\leq C \quad \forall k:1\leq k\leq n.
\end{array}
\right. \tag{5.11}
$$

Nous avons donc

$$
\left\{
\begin{array}{ll}
\max & -\frac{1}{2}(\alpha_i-\alpha_i^*)^2K_{ii}-\frac{1}{2}(\alpha_j-\alpha_j^*)^2K_{jj}-(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)K_{ij} \\
& -\frac{1}{2}\sum_{k\neq i,j}^{n}\sum_{l\neq i,j}^{n}(\alpha_k-\alpha_k^*)(\alpha_l-\alpha_l^*)K_{kl} \\
& -\frac{1}{2}\sum_{k\neq i,j}^{n}(\alpha_k-\alpha_k^*)(\alpha_i-\alpha_i^*)K_{ki} \\
& -\frac{1}{2}\sum_{k\neq i,j}^{n}(\alpha_k-\alpha_k^*)(\alpha_j-\alpha_j^*)K_{kj} \\
& -\frac{1}{2}\sum_{l\neq i,j}^{n}(\alpha_i-\alpha_i^*)(\alpha_l-\alpha_l^*)K_{il} \\
& -\frac{1}{2}\sum_{l\neq i,j}^{n}(\alpha_j-\alpha_j^*)(\alpha_l-\alpha_l^*)K_{jl} \\
& -\varepsilon\sum_{k\neq i,j}^{n}(\alpha_k+\alpha_k^*)-\varepsilon(\alpha_i+\alpha_i^*+\alpha_j+\alpha_j^*) \\
& +\sum_{k\neq i,j}^{n}y_k(\alpha_k-\alpha_k^*)+y_i(\alpha_i-\alpha_i^*)+y_j(\alpha_j-\alpha_j^*) \\
\text{s.c.} & \sum_{k=1}^{n}(\alpha_k-\alpha_k^*)=0, \\
& 0\leq\alpha_k\leq C \quad \forall k:1\leq k\leq n, \\
& 0\leq\alpha_k^*\leq C \quad \forall k:1\leq k\leq n.
\end{array}
\right. \tag{5.12}
$$

En substituant $(\alpha_j - \alpha_j^*)$ par $\gamma - (\alpha_i - \alpha_i^*)$ (cfr 5.3) et en supprimant les parties indépendantes de α_i et α_j nous avons que

$$\begin{cases} \max & -\frac{1}{2}(\alpha_i - \alpha_i^*)^2 K_{ii} - \frac{1}{2}(\gamma^2 - 2\gamma(\alpha_i - \alpha_i^*) + (\alpha_i - \alpha_i^*)^2) K_{jj} \\ & -\gamma(\alpha_i - \alpha_i^*) K_{ij} + (\alpha_i - \alpha_i^*)^2 K_{ij} - \frac{1}{2}\sum_{k\neq i,j}^{n} (\alpha_k - \alpha_k^*)(\alpha_i - \alpha_i^*) K_{ki} \\ & -\frac{1}{2}\gamma \sum_{k\neq i,j}^{n} (\alpha_k - \alpha_k^*) K_{kj} + \frac{1}{2}\sum_{k\neq i,j}^{n} (\alpha_k - \alpha_k^*)(\alpha_i - \alpha_i^*) K_{kj} \\ & -\frac{1}{2}\sum_{k\neq i,j}^{n} (\alpha_i - \alpha_i^*)(\alpha_k - \alpha_k^*) K_{ik} - \frac{1}{2}\gamma \sum_{k\neq i,j}^{n} (\alpha_k - \alpha_k^*) K_{jk} \\ & +\frac{1}{2}\sum_{k\neq i,j}^{n} (\alpha_i - \alpha_i^*)(\alpha_k - \alpha_k^*) K_{jk} - \varepsilon(\alpha_i + \alpha_i^* + \alpha_j + \alpha_j^*) \\ & +y_i(\alpha_i - \alpha_i^*) + \gamma y_j - y_j(\alpha_i - \alpha_i^*) \\ \text{s.c.} & L \leq (\alpha_i - \alpha_i^*) \leq H \quad \forall i : 1 \leq i \leq n, \end{cases} \tag{5.13}$$

où L et H sont définis dans le Tableau 5.1. En mettant en évidence les termes semblables et en éliminant les termes indépendants de $(\alpha_i - \alpha_i^*)$ nous obtenons le problème d'optimisation suivant

$$\begin{cases} \max & -\frac{1}{2}(\alpha_i - \alpha_i^*)^2 (K_{ii} - 2K_{ij} + K_{jj}) - (\alpha_i - \alpha_i^*)(\gamma(K_{ij} - K_{jj})) \\ & -(\alpha_i - \alpha_i^*) \sum_{k\neq i,j}^{n} (\alpha_k - \alpha_k^*) K_{ki} + (\alpha_i - \alpha_i^*) \sum_{k\neq i,j}^{n} (\alpha_k - \alpha_k^*) K_{kj} \\ & +y_i(\alpha_i - \alpha_i^*) - y_j(\alpha_i - \alpha_i^*) - \varepsilon(\alpha_i + \alpha_i^* + \alpha_j + \alpha_j^*) \\ \text{s.c.} & L \leq (\alpha_i - \alpha_i^*) \leq H \quad \forall i : 1 \leq i \leq n. \end{cases} \tag{5.14}$$

Finalement, en réorganisant les différents termes nous obtenons le problème d'optimisation

$$\begin{cases} \max & -\frac{1}{2}(\alpha_i - \alpha_i^*)^2 (K_{ii} - 2K_{ij} + K_{jj}) - \varepsilon(\alpha_i + \alpha_i^* + \alpha_j + \alpha_j^*) \\ & -(\alpha_i - \alpha_i^*)\Bigg(\gamma(K_{ij} - K_{jj}) + \underbrace{\sum_{k\neq i,j}^{n} (\alpha_k - \alpha_k^*) K_{ki} - y_i - b}_{-v_i} \\ & - \underbrace{\sum_{k\neq i,j}^{n} (\alpha_k - \alpha_k^*) K_{kj} + y_j + b}_{v_j}\Bigg) \\ \text{s.c.} & L \leq (\alpha_i - \alpha_i^*) \leq H \quad \forall i : 1 \leq i \leq n, \end{cases} \tag{5.15}$$

qui est équivalent à

$$\begin{cases} \max & -\frac{1}{2}(\alpha_i - \alpha_i^*)^2 (K_{ii} - 2K_{ij} + K_{jj}) - \varepsilon(\alpha_i + \alpha_i^* + \alpha_j + \alpha_j^*) \\ & -(\alpha_i - \alpha_i^*)(\gamma(K_{ij} - K_{jj}) - v_i + v_j) \\ \text{s.c.} & L \leq (\alpha_i - \alpha_i^*) \leq H \quad \forall i : 1 \leq i \leq n. \end{cases} \tag{5.16}$$

L'étape suivante consiste à trouver le maximum pour chacun des quatre cas. Celui-ci est donné simplement en prenant la dérivée de la fonction objectif de (5.16). Soit la fonction objectif

$$L = -\frac{1}{2}(\alpha_i - \alpha_i^*)^2 \eta - (\alpha_i - \alpha_i^*)(\gamma(K_{ij} - K_{jj}) - v_i + v_j) - \varepsilon(\alpha_i + \alpha_i^* + \alpha_j + \alpha_j^*), \tag{5.17}$$

où $\eta \overset{déf}{=} K_{ii} + K_{jj} - 2K_{ij}$. Considérons maintenant les quatre cas :

- **Cas 1 :** (α_i, α_j)

  La fonction objectif devient

  $$L = -\frac{1}{2}(\alpha_i)^2\eta - \alpha_i(\gamma(K_{ij} - K_{jj}) - v_i + v_j) - \varepsilon(\alpha_i + \alpha_j). \tag{5.18}$$

  Par (5.3), nous avons

  $$\begin{aligned}(\alpha_i - \alpha_i^*) + (\alpha_j - \alpha_j^*) = \gamma \quad &\Leftrightarrow \quad \alpha_i + \alpha_j = \gamma \\ &\Leftrightarrow \quad \alpha_j = \gamma - \alpha_i,\end{aligned}$$

  qui implique que (5.18) est équivalent à

  $$L = -\frac{1}{2}(\alpha_i)^2\eta - \alpha_i(\gamma(K_{ij} - K_{jj}) - v_i + v_j) - \varepsilon\gamma. \tag{5.19}$$

  La dérivée première de (5.19) par rapport à α_i est donnée par

  $$\frac{\partial L}{\partial \alpha_i} = -\alpha_i\eta - (\gamma(K_{ij} - K_{jj}) - v_i + v_j). \tag{5.20}$$

  Au maximum, cette dérivée doit être égale à zéro, d'où

  $$\frac{\partial L}{\partial \alpha_i} = 0 \Leftrightarrow \alpha_i = \frac{v_i - v_j - \gamma(K_{ij} - K_{jj})}{\eta},$$

  ce qui implique, par (5.8), que

  $$\alpha_i = \alpha_i^{old} + \frac{\phi_i - \phi_j}{\eta}.$$

- **Cas 2 :** (α_i, α_j^*)

  La fonction objectif devient

  $$L = -\frac{1}{2}(\alpha_i)^2\eta - \alpha_i(\gamma(K_{ij} - K_{jj}) - v_i + v_j) - \varepsilon(\alpha_i + \alpha_j^*). \tag{5.21}$$

  Par (5.3), nous avons

  $$\begin{aligned}(\alpha_i - \alpha_i^*) + (\alpha_j - \alpha_j^*) = \gamma \quad &\Leftrightarrow \quad \alpha_i - \alpha_j^* = \gamma \\ &\Leftrightarrow \quad \alpha_j^* = \alpha_i - \gamma,\end{aligned}$$

  qui implique que (5.21) est équivalent à

  $$L = -\frac{1}{2}(\alpha_i)^2\eta - \alpha_i(\gamma(K_{ij} - K_{jj}) - v_i + v_j) - 2\varepsilon\alpha_i + \varepsilon\gamma. \tag{5.22}$$

  La dérivée première de (5.22) par rapport à α_i est donnée par

  $$\frac{\partial L}{\partial \alpha_i} = -\alpha_i\eta - (\gamma(K_{ij} - K_{jj}) - v_i + v_j) - 2\varepsilon. \tag{5.23}$$

  Au maximum, cette dérivée doit être égale à zéro, d'où

  $$\frac{\partial L}{\partial \alpha_i} = 0 \Leftrightarrow \alpha_i = \frac{v_i - v_j - \gamma(K_{ij} - K_{jj}) - 2\varepsilon}{\eta},$$

  ce qui implique, par (5.8), que

  $$\alpha_i = \alpha_i^{old} + \frac{\phi_i - \phi_j - 2\varepsilon}{\eta}.$$

- **Cas 3** : (α_i^*, α_j)

  La fonction objectif devient

  $$L = -\frac{1}{2}(\alpha_i^*)^2\eta + \alpha_i^*(\gamma(K_{ij} - K_{jj}) - v_i + v_j) - \varepsilon(\alpha_i^* + \alpha_j). \tag{5.24}$$

  Par (5.3), nous avons

  $$\begin{aligned}(\alpha_i - \alpha_i^*) + (\alpha_j - \alpha_j^*) = \gamma &\Leftrightarrow -\alpha_i^* + \alpha_j = \gamma \\ &\Leftrightarrow \alpha_j = \gamma + \alpha_i^*,\end{aligned}$$

  qui implique que (5.24) est équivalent à

  $$L = -\frac{1}{2}(\alpha_i^*)^2\eta + \alpha_i^*(\gamma(K_{ij} - K_{jj}) - v_i + v_j) - 2\varepsilon\alpha_i^* + \varepsilon\gamma. \tag{5.25}$$

  La dérivée première de (5.25) par rapport à α_i^* est donnée par

  $$\frac{\partial L}{\partial \alpha_i^*} = -\alpha_i^*\eta + (\gamma(K_{ij} - K_{jj}) - v_i + v_j) - 2\varepsilon. \tag{5.26}$$

  Au maximum, cette dérivée doit être égale à zéro, d'où

  $$\frac{\partial L}{\partial \alpha_i^*} = 0 \Leftrightarrow \alpha_i^* = \frac{v_j - v_i + \gamma(K_{ij} - K_{jj}) - 2\varepsilon}{\eta},$$

  ce qui implique, par (5.8), que

  $$\alpha_i^* = \alpha_i^{*old} - \frac{\phi_i - \phi_j + 2\varepsilon}{\eta}.$$

- **Cas 4** : (α_i^*, α_j^*)

  La fonction objectif devient

  $$L = -\frac{1}{2}(\alpha_i^*)^2\eta + \alpha_i^*(\gamma(K_{ij} - K_{jj}) - v_i + v_j) - \varepsilon(\alpha_i^* + \alpha_j^*). \tag{5.27}$$

  Par (5.3), nous avons

  $$\begin{aligned}(\alpha_i - \alpha_i^*) + (\alpha_j - \alpha_j^*) = \gamma &\Leftrightarrow -\alpha_i^* - \alpha_j^* = \gamma \\ &\Leftrightarrow \alpha_j^* = -\alpha_i^* - \gamma,\end{aligned}$$

  qui implique que (5.27) est équivalent à

  $$L = -\frac{1}{2}(\alpha_i^*)^2\eta + \alpha_i^*(\gamma(K_{ij} - K_{jj}) - v_i + v_j) + \varepsilon\gamma. \tag{5.28}$$

  La dérivée première de (5.28) par rapport à α_i^* est donnée par

  $$\frac{\partial L}{\partial \alpha_i^*} = -\alpha_i^*\eta + (\gamma(K_{ij} - K_{jj}) - v_i + v_j). \tag{5.29}$$

  Au maximum, cette dérivée doit être égale à zéro, d'où

  $$\frac{\partial L}{\partial \alpha_i^*} = 0 \Leftrightarrow \alpha_i^* = \frac{v_j - v_i + \gamma(K_{ij} - K_{jj})}{\eta},$$

  ce qui implique, par (5.8), que

  $$\alpha_i^* = \alpha_i^{*old} - \frac{\phi_i - \phi_j + 2\varepsilon}{\eta}.$$

Le tableau suivant reprend le maximum pour chacun des quatre cas

(α_i, α_j)	$\alpha_i = \frac{v_i - v_j - \gamma(K_{ij} - K_{jj})}{\eta} = \alpha_i^{old} + \frac{\phi_i - \phi_j}{\eta}$
(α_i, α_j^*)	$\alpha_i = \frac{v_i - v_j - \gamma(K_{ij} - K_{jj}) - 2\varepsilon}{\eta} = \alpha_i^{old} + \frac{\phi_i - \phi_j - 2\varepsilon}{\eta}$
(α_i^*, α_j)	$\alpha_i^* = \frac{v_j - v_i + \gamma(K_{ij} - K_{jj}) - 2\varepsilon}{\eta} = \alpha_i^{*old} - \frac{\phi_i - \phi_j + 2\varepsilon}{\eta}$
(α_i^*, α_j^*)	$\alpha_i^* = \frac{v_j - v_i + \gamma(K_{ij} - K_{jj})}{\eta} = \alpha_i^{*old} - \frac{\phi_i - \phi_j}{\eta}$

Tableau 5.2 – Maximum du problème d'optimisation SVR en fonction des quatre cas.

avec

$$\eta = K_{ii} + K_{jj} - 2K_{ij}. \tag{5.30}$$

Afin d'optimiser plus de deux variables, nous devons recalculer $\phi_i - \phi_j$. De plus, nous voulons éviter des coûts supplémentaires lorsque nous recalculons la différence de ces erreurs. Nous pouvons pour cela procéder de la manière qui suit

$$\phi_i^{new} - \phi_j^{new} = y_i - f(x_i)^{new} - y_j + f(x_j)^{new}$$

avec

$$\begin{aligned} f(x_i)^{new} &= f(x_i) - (\alpha_i^{old} - \alpha_i^{*old})K_{ii} + (\alpha_i^{new} - \alpha_i^{*new})K_{ii} \\ &\quad -(\alpha_i^{old} - \alpha_i^{*old})K_{ij} + (\alpha_i^{new} - \alpha_i^{*new})K_{ij}. \end{aligned}$$

D'où,

$$\begin{aligned} \phi_i^{new} - \phi_j^{new} &= y_i - f(x_i) - ((\alpha_i^{new} - \alpha_i^{*new}) - (\alpha_i^{old} - \alpha_i^{*old}))(K_{ii} - K_{ij}) \\ &\quad y_j - f(x_j) - ((\alpha_j^{new} - \alpha_j^{*new}) - (\alpha_j^{old} - \alpha_j^{*old}))(K_{ij} - K_{jj}) \\ &= \phi_i^{old} - \phi_j^{old} - (\alpha_i^{new} - \alpha_i^{*new})K_{ii} + (\alpha_i^{old} - \alpha_i^{*old})K_{ii} \\ &\quad +(\alpha_i^{new} - \alpha_i^{*new})K_{ij} - (\alpha_i^{old} - \alpha_i^{*old})K_{ij} \\ &\quad -(\alpha_j^{new} - \alpha_j^{*new})K_{ij} + (\alpha_j^{old} - \alpha_j^{*old})K_{ij} \\ &\quad +(\alpha_j^{new} - \alpha_j^{*new})K_{jj} - (\alpha_j^{old} - \alpha_j^{*old})K_{jj} \\ &= \phi_i^{old} - \phi_j^{old} - ((\alpha_i^{new} - \alpha_i^{*new}) - (\alpha_i^{old} - \alpha_i^{*old}))(K_{ii} - K_{ij}) \\ &\quad +((\alpha_j^{new} - \alpha_j^{*new}) - (\alpha_j^{old} - \alpha_j^{*old}))(K_{jj} - K_{ij}). \end{aligned}$$

Finalement par (5.3) et (5.4), nous avons

$$\phi_i^{new} - \phi_j^{new} = \phi_i^{old} - \phi_j^{old} - \eta((\alpha_i^{new} - \alpha_i^{*new}) - (\alpha_i^{old} - \alpha_i^{*old})). \tag{5.31}$$

Remarque

A la suite d'erreurs numériques, il se peut que $\eta < 0$. Dans ce cas, η doit être fixé à zéro. En effet, une valeur négative de η ne peut pas être tolérée puisque $k(\cdot, \cdot)$ doit vérifier la condition de Mercer (voir Théorème 3.4.1), ce qui implique que la valeur optimale de α_i se trouve sur les frontières L ou H.

Règle de sélection pour la régression

Nous nous intéressons maintenant à une règle de décision permettant de sélectionner les indices (i, j) de telle manière que la fonction objectif soit maximale. L'approche suivie consiste en une double boucle imbriquée permettant de maximiser la fonction objectif. La boucle externe itère sur toutes les données ne vérifiant pas les conditions de KKT, d'abord, sur celles dont les multiplicateurs de Lagrange ne sont pas sur les frontières (i.e., $\alpha_i \neq 0$ et $\neq C$), ensuite sur celles de l'ensemble d'apprentissage violant les conditions de KKT, et ce afin d'assurer la consistence du modèle pour toutes les données de l'ensemble d'apprentissage, ce qui règle le problème du choix de l'indice i. Le choix de l'indice j se base sur l'idée suivante : afin de faire en sorte de se rapprocher rapidement du minimum, nous exigeons de grandes modifications de α_i. Puisque calculer η pour toutes les paires possibles (i, j) coûte très cher en temps de calcul nous choisissons l'indice j qui maximise la valeur absolue de $|\phi_i - \phi_j|$. Si cette heuristique vient à échouer, tous les autres indices j sont alors analysés. En résumé :

1. Nous analysons tous les indices j correspondant aux données d'apprentissage ne se trouvant pas sur les frontières L ou H afin d'en trouver un permettant de progresser.

2. Si aucun indice j n'est satisfaisant, les autres données sont analysées pour en trouver une qui permet de progresser.

3. Si aucune des étapes 1 et 2 ne renvoie un indice j, alors nous passons à l'indice i suivant.

Mise à jour de la constante b

La dernière étape de l'algorithme SMO consiste à déterminer la valeur de la constante b. Malheureusement, l'algorithme SMO ne fournit pas automatiquement la valeur de b qui doit être mise à jour après chaque itération réussie. Nous procédons alors de la façon suivante : si seulement une des variables α_i, α_i^*, α_j ou α_j^* se trouve à l'intérieure des frontières alors nous pouvons utiliser (4.23) ou (4.24) définis au Chapitre 4. Malheureusement, la constante b ne peut pas tout le temps être calculée (e.g., quand tous les vecteurs de support se trouvent en dehors du tube d'insensibilité ε). Dans ce cas, Platt montre qu'il existe un intervalle admissible pour b,

$$[b_i, b_j]$$

où b_i et b_j sont donnés par (4.23) ou (4.24). Il suffit alors de prendre pour b le milieu de cet intervalle :

$$b = \frac{b_i + b_j}{2}.$$

Pseudo-code de l'algorithme SMO

Nous donnons dans cette section le pseudo-code de l'algorithme SMO. Une implémentation [3] de cet algorithme est donnée au Chapitre 7. Ce pseudo-code est entièrement repris dans l'implémentation par soucis de clarté et de facilité de compréhension. Etant donné que les commentaires de notre programme sont en anglais, nous donnons le pseudo-code source également en anglais pour faciliter le travail des différents groupes de Cenaero.

En premier lieu, nous résumons les étapes principales de l'algorithme SMO. L'algorithme SMO est composé de deux parties et se déroule comme suit.

Algorithme 3 Algorithme SMO

1: Sélection des variables α_i et α_j.
Nous utilisons les heuristiques présentés à la Section 5.2.3.3 afin de choisir efficacement les paires des multiplicateurs de Lagrange à optimiser.

2: Optimisation.
Nous fixons tous les α_k $\forall k : 1 \leq k \leq n$ et $k \neq i, j$ et nous calculons la solution analytique du problème quadratique de taille 2.

3. Pour des raisons de confidentialité, le code source sera uniquement disponible le jour de la défense du mémoire.

Algorithme 3 Procedure takeStep(i1,i2)

```
1: if (i1 = i2) then
2:    return 0
3: end if
4: alpha1, alpha1*, alpha2, alpha2* = Lagrange multipliers for i1 and i2
5: y1 = target[i1]
6: y2 = target[i2]
7: phi1 = SVM output on point[i1] - y1
8: phi2 = SVM output on point[i1] - y2
9: k11 = kernel(point[i1],point[i1])
10: k12 = kernel(point[i1],point[i2])
11: k22 = kernel(point[i2],point[i2])
12: eta = - 2*k12 + k11 + k22
13: gamma = alpha1 - alpha1* + alpha2 - alpha2*
14: { % We assume eta > 0. otherwise one has to repeat the complete reasoning similarly }
15: { % (compute objective function for L and H and decide which one is largest)}
16: case1 = case2 = case3 = case4 = finished = 0
17: alpha1old, alpha1old* = alpha1, alpha1*
18: alpha2old, alpha2old* = alpha2, alpha2*
19: deltaphi = phi1 - phi2
20: while !finished do
21:    { % This loop is passed at most three times}
22:    { % Case variables needed to avoid attempting to avoid attempting small changes twice}
23:    if (case1 == 0) && (alpha1 > 0 || (alpha1* == 0 && deltaphi > 0)) && (alpha2 > 0 ||
       (alpha2* == 0 && deltaphi < 0)) then
24:       Compute L, H (wrt. alpha1, alpha2)
25:       if L < H then
26:          a2 = alpha2 - deltaphi/eta
27:          a2 = min(a2, H)
28:          a2 = max(L, a2)
29:          a1 = alpha1 - (a2 - alpha2)
30:          Update alpha1, alpha2 if change is larger than some eps
31:       else
32:          finished = 1
33:       end if
34:       case1 = 1
35:    else if (case2 == 0) && (alpha1 > 0 || (alpha1* == 0 && deltaphi > 2 epsilon)) &&
       (alpha2* > 0 || (alpha2 == 0 && deltaphi > 2 epsilon)) then
36:       Compute L, H (wrt. alpha1, alpha2*)
37:       if L < H then
38:          a2 = alpha2* + (deltaphi - 2 epsilon)/eta
39:          a2 = min(a2, H)
40:          a2 = max(L, a2)
41:          a1 = alpha1 + (a2 - alpha2*)
42:          update alpha1, alpha2* if change is larger than some eps
43:       else
44:          finished = 1
45:       end if
46:       case2 = 1
47:    else if (case3 == 0) && (alpha1* > 0 || (alpha1 == 0 && deltaphi < -2 epsilon)) &&
       (alpha2 > 0 || (alpha2* == 0 && deltaphi < -2 epsilon)) then
48:       Compute L, H (wrt. alpha1*, alpha2)
49:       if L < H then
50:          a2 = alpha2 - (deltaphi - 2 epsilon)/eta
51:          a2 = min(a2, H)
52:          a2 = max(L, a2)
53:          a1 = alpha1* + (a2 - alpha2*)
54:          Update alpha1*, alpha2 if change is larger than some eps
55:       else
56:          finished = 1
57:       end if
58:       case3 = 1
59:    else if (case4 == 0) && (alpha1* > 0 || (alpha1 == 0 && deltaphi < 0)) && (alpha2* >
       0 || (alpha2 == 0 && deltaphi > 0)) then
60:       if L < H then
61:          a2 = alpha2* + deltaphi/eta
62:          a2 = min(a2, H)
63:          a2 = max(L, a2)
64:          a1 = alpha1* - (a2 - alpha2*)
65:          Update alpha1*, alpha2* if change is larger than some eps
66:       else
67:          finished = 1
68:       end if
69:       case4 = 1
70:    else
71:       finished = 1
72:    end if
73:    update deltaphi
74: end while
75: Update threshold to reflect change in Lagrange multipliers
76: Update error cache usinig new Lagrange multipliers
77: if changes in alpha1(*), alpha2(*) are larger than some eps then
78:    return 1
79: else
80:    return 0
81: end if
```

Algorithme 4 Procedure examineExample(i2)

```
y2 = target[i2]
alpha2, alpha2* = Lagrange multipliers for i2
C2, C2* = Constraints for i2
phi2 = SVM output on point[i2] - y2 (in error cache)
if ((phi2 > epsilon && alpha2* < C2*) || (phi2 < epsilon && alpha2* > 0) || (-phi2 >
epsilon && alpha2 < C2) || (-phi2 > epsilon && alpha2 > 0)) then
   if (number of non-zero && non-C alpha > 1) then
      i1 = result of first or second heuristic
      if (takeStep(i1,i2)) then
         return 1
      end if
   end if
   loop {over all non-zero and non-C alpha, with random start}
      i1 = identity of current alpha
      if (takeStep(1i,i2)) then
         return 1
      end if
   end loop
   loop {over all possible i1, with random start}
      i1 = loop variable
      if (takeStep(1i,i2)) then
         return 1
      end if
   end loop
end if
return 0
```

Algorithme 5 main routine

```
initialize alpha and alpha* array to all zero
initialize threshold to zero
numChanged = 0
examineAll = 1
LoopCounter = 0
while (numChanged > 0 || examineAll) do
   LoopCounter ++
   nmChanged =0
   if (examineAll) then
      loop {over i over all training examples}
         numChanged += examineExample(i)
      end loop
   else
      loop {over i over examples where alpha is not 0 and not C}
         numChanged += examineExample(i)
      end loop
   end if
   if (mod(LoopCounter, 2) == 0) then
      MinimumNumChanged = max(1, 0.1*NumSamples)
   else
      MinimumChanged = 1
   end if
   if (examineAll == 1) then
      examineAll = 0
   else if (numChanged < MinimumNumChanged) then
      examineAll = 1
   end if
end while
```

Améliorations de l'algorithme SMO

De nombreuses améliorations de l'algorithme SMO ont été proposées afin d'optimiser le temps de calcul et la précision de la solution calculée. En effet, l'algorithme SMO décrit au Chapitre 5 présente un inconvénient causé par l'utilisation d'une valeur unique pour la mise à jour de la constante b. Parmi toutes les modifications possibles de l'algorithme SMO, nous pouvons citer, comme référence, l'étude : "Improvements to SMO Algorithm for SVM Regression" [65]. Dans cette étude, Shevade, Keerthi et al. développent un nouvel algorithme SMO plus efficace et plus rapide que l'algorithme SMO initial.

5.2.4 Les différentes librairies

Il existe de nombreuses librairies permettant de résoudre le problème quadratique. Les plus connues sont CPLEX, LOQO, Matlab, linprog ou quadprog. Elles sont capables de résoudre les problèmes linéaires ou quadratiques issus des SVM. Néanmoins, comme c'est expliqué précédemment, la taille des problèmes (i.e., la taille de l'ensemble d'apprentissage) a poussé la communauté ML à développer et implémenter des nouveaux algorithmes plus spécifiques basés principalement et en majorité sur l'algorithme SMO. De nouvelles librairies ont alors vu le jour telles que SVMlight, libSVM, HeroSVM ou encore WEKA. Toutes ces librairies sont bien souvent téléchargeables sur Internet. Malheureusement, Cenaero étant un centre de recherche vendant des solutions technologiques, elle ne peut pas utiliser ces librairies. C'est la raison principale qui nous a poussé à développer notre propre code basé sur l'algorithme SMO.

5.3 Procédure de sélection des hyperparamètres

Dans la méthode SVR, de nombreux paramètres doivent être sélectionnés :

- l'hyperparamètre C qui représente le compromis entre la complexité du modèle et l'erreur sur les données d'apprentissage,
- l'hyperparamètre ε qui correspond à la largeur du tube d'insensibilité,
- l'hyperparamètre σ qui est le paramètre de la fonction noyau RBF Gaussien.

Ces hyperparamètres sont en général réglés en fonction d'une estimation de l'erreur de généralisation qui peut être évaluée sur un jeu indépendant de données de validation ou par validation croisée. Cela implique donc la construction d'un modèle pour chaque jeu de données et l'évaluation de celui-ci.

La méthode que nous avons choisi d'implémenter est basée sur une recherche de type grille expliquée à la Section 2.2. Pour cela, nous faisons simplement varier :

- l'hyperparamètre C de 1 à 50 par pas logarithmique,
- l'hyperparamètre σ de 0.1 à 1 par pas de 0.1.

Les modèles obtenus pour chaque jeu d'hyperparamètres sont alors évalués à l'aide d'une procédure de validation croisée expliquée à la Section 2.3.1. Si le

modèle obtenu avec le nouveau jeu d'hyperparamètres est meilleur que l'ancien (i.e. le coefficient de corrélation du modèle obtenu avec le nouveau jeu d'hyperparamètres R_{new} est plus grand que l'ancien coefficient de corrélation R_{old} correspondant à l'ancien modèle) alors nous conservons les nouveaux hyperparamètres et nous continuons la recherche. Un mécanisme permettant d'arrêter la recherche a aussi été implémenté afin d'éviter des calculs superflus. Celui-ci vérifie que si R_{new} est plus grand que 0.99 alors un jeu de très bons hyperparamètres a été trouvé et la recherche est alors arrêtée. Le paramètre [4] k de la validation croisée a été fixé à 5 et l'hyperparamètre [5] ε est quant à lui fixé à 0.001. Finalement, une fois que la recherche des hyperparamètres est terminée, nous construisons le modèle f avec le jeu des meilleurs hyperparamètres obtenus et avec toutes les données de l'ensemble d'apprentissage. La Figure 5.2 illustre la procédure de sélection des hyperparamètres.

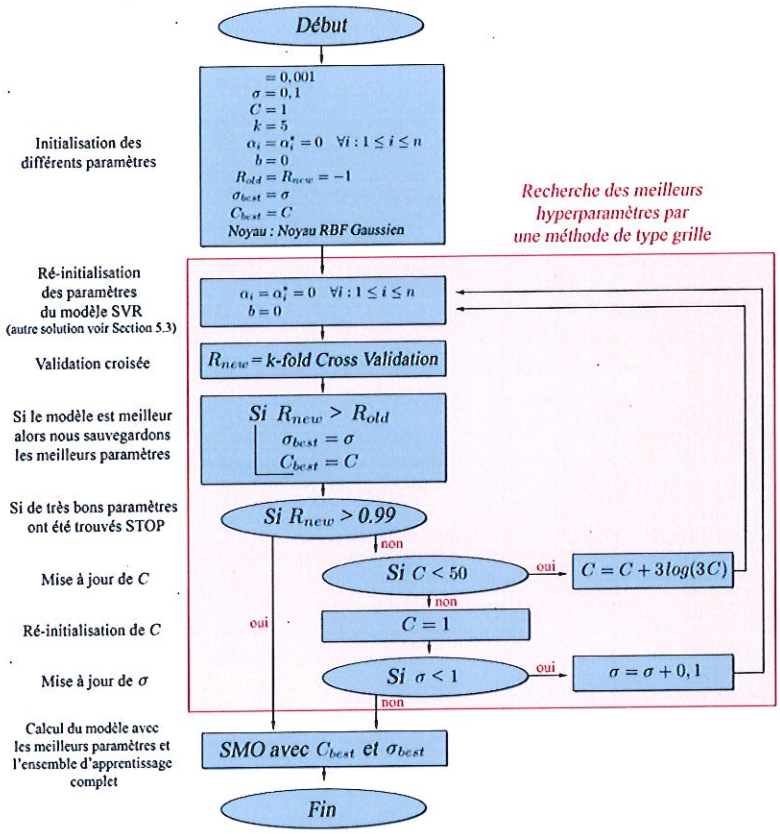


FIGURE 5.2 – Procédure de sélection des meilleurs hyperparamètres.

4. Ne pas confondre le paramètre k de la validation croisée avec le noyau.

5. L'hyperparamètre ε peut aussi être ajouté à la recherche mais nous avons constaté que celui-ci n'améliorait pas énormément la qualité des modèles par rapport à un hyperparamètre ε fixé à 0.001.

D'autres approches permettent d'optimiser le réglage des hyperparamètres. Certaines sont basées sur des méthodes évolutionnistes [26] et d'autres sur une recherche par descente de gradient [13, 4].

5.4 Les différentes astuces et points techniques

L'algorithme SMO que nous avons implémenté fait partie intégrante du programme d'optimisation Minamo, le couplage avec ce dernier étant présenté au Chapitre 1. Des nouvelles fonctionnalités, telle que la validation externe, ont été ajoutées au programme Minamo. Ces nouvelles fonctionnalités et l'algorithme SMO se présentent sous forme de classes ou de fonctions codées en C++. De plus, afin que la nouvelle méthode d'approximation (i.e., l'algorithme SMO) soit compatible avec le reste du programme, nous avons dû adapter la classe SurrogateModel qui comprend les modèles RBF, RBF tuné et Kriging.

Normalisation des données d'apprentissage

Dans Minamo, les données de l'ensemble d'apprentissage obtenues par une technique de plan d'expériences sont normalisées entre -0.5 et 0.5 avant l'exécution d'un algorithme d'apprentissage. La normalisation des données permet généralement d'obtenir des meilleurs résultats et surtout d'améliorer la stabilité numérique.

Stockage de la matrice noyau

Lors de l'apprentissage d'un modèle f par l'algorithme SMO, nous devons évaluer très fréquemment le noyau des données de l'ensemble d'apprentissage. L'idée est donc de pré-calculer et de stocker la matrice noyau

$$K = (k(x_i, x_j))_{i,j=1}^{n}$$

afin de pouvoir l'utiliser sans coût supplémentaire. Cette technique a été implémentée et nous permet un gain de temps de calcul conséquent.

Erreur d'arrondi

Lors de l'apprentissage d'un modèle f par l'algorithme SMO, nous devons très souvent vérifier que certaines variables α_i ou α_i^* sont différentes de 0 ou de C. Afin d'éviter des erreurs d'arrondi, nous vérifions plutôt que

$$|\alpha_i| < mEps \quad \text{et} \quad |\alpha_i^*| < mEps$$

et

$$|\alpha_i - C| < mEps \quad \text{et} \quad |\alpha_i^* - C| < mEps$$

où $mEps$ représente une certaine précision. Généralement, $mEps = 10^{-16}$.

Modification du point de départ pour l'algorithme SMO

Comme expliqué précédemment, lors de la constuction d'un modèle f, la sélection d'hyperparamètres optimaux est une étape cruciale (voir Section 5.3). Lors de la recherche de ces hyperparamètres, nous devons lancer plusieurs fois l'algorithme SMO pour des jeux de paramètres (C, ε, σ, etc.) différents. Une idée permettant d'éviter les temps de calcul trop importants dans la recherche des hyperparamètres optimaux est de relancer l'algorithme d'apprentissage SMO avec des multiplicateurs de Lagrange et une variable b proche de la solution. La méthode proposée est la suivante : si la nouvelle valeur de l'hyperparamètre C à tester C_{new} est proche de l'ancienne C_{old}, alors nous pouvons réajuster les valeurs des multiplicateurs de Lagrange et de la constante b comme suit

$$\alpha_{new} = \frac{C_{new}}{C_{old}}\alpha_{old} \quad \text{et} \quad b_{new} = \frac{C_{new}}{C_{old}}b_{old}. \tag{5.32}$$

Ces nouveaux paramètres peuvent alors être utilisés pour la résolution du nouveau problème d'optimisation par l'algorithme SMO. La même méthode peut également être appliquée pour les paramètres de la fonction noyau. Une justification théorique de cette technique est donnée dans [62].

Nous avons implémenté et testé cette technique avec notre recherche des hyperparamètres. Un gain de temps de calcul a été observé par rapport à un réapprentissage complet. De plus, sur les cas testés, les hyperparamètres trouvés sont identiques à ceux renvoyés lors d'un réapprentissage complet. Cependant, nous avons décidé de désactiver cette technique pour les différents tests du Chapitre 6 afin d'être certain de ne pas biaiser les résultats obtenus. En effet, pour éviter des temps de calcul trop importants lors des différents tests, nous avons décidé de considérer uniquement, lors de la recherche des hyperparamètres, quelques valeurs de C différentes.

Chapitre 6

Tests, résultats et interprétations

RÉSUMÉ. *Dans ce chapitre, nous illustrons les propriétés importantes des SVM pour la régression ainsi que le fonctionnement de la sélection d'un modèle par les SVR. Nous présentons ensuite les différents résultats obtenus. Ces résultats ont pour but de tester et valider la méthode SVR et l'algorithme SMO pour la régression ainsi que de motiver d'éventuelles possibilités de recherches futures à ce mémoire. Finalement, nous comparons notre modèle SVR au modèle RBF "tuné" de Minamo à l'aide de différentes mesures d'erreur.*

6.1 Méthodologie

Avant d'introduire les différentes illustrations et les différents résultats, nous présentons les fonctions analytiques utilisées lors de la phase de tests. Nous définissons ensuite les différents types d'erreurs utilisées pour valider les modèles ainsi que la validation externe. Par la suite, nous décrivons brièvement les deux types de modèles RBF déjà présents dans Minamo. Finalement, nous comparons les résultats obtenus avec le modèle SVR et ceux obtenus avec les modèles RBF implémentés dans Minamo.

6.1.1 Fonctions tests utilisées

Nous présentons dans cette section les différentes fonctions analytiques sur lesquelles le modèle SVR a été testé. L'utilisation de fonctions analytiques connues permet d'obtenir facilement les ensembles d'apprentissage par une technique de plan d'expériences (voir Section 1.2.4). De plus, un bruit gaussien a été implémenté et peut facilement être ajouté à ces fonctions tests afin de tester la validité du modèle SVR en cas de données bruitées.

Fonction à une dimension

Lors de l'implémentation de l'algorithme SMO, nous avons été confrontés au fait que le pseudo-code source proposé par Smola dans [63] comportait des erreurs et de nombreuses imprécisions [1]. En conséquence, nous avons donc dû implémenter personnellement l'algorithme SMO en redéveloppant et vérifiant toutes les étapes et procédures. Afin de débugger notre code-source, nous avons utilisé une fonction proposée dans [14]. L'approximation de cette fonction (i.e., les multiplicateurs de Lagrange) et la matrice noyau données dans cet article nous ont permis de vérifier le bon fonctionnement de notre algorithme. Cette fonction est donnée comme suit

$$f(x) = \sum_{i=1}^{9} a_i (x - 900)^{(i-1)}, \tag{6.1}$$

où $x \in \mathbb{R}$ et

$$a_1 = -659.23, \quad a_2 = 190.22 \quad a_3 = -17.802, \quad a_4 = 0.82691,$$
$$a_5 = -0.021885, \quad a_6 = 0.0003463, \quad a_7 = -3.2446 \times 10^{-6}$$
$$a_8 = 1.6606 \times 10^{-8}, \quad a_9 = -3.5757 \times 10^{-11}.$$

La représentation graphique de cette fonction est donnée à la Figure 6.1

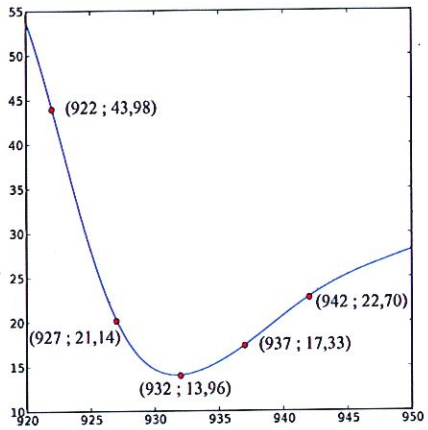


FIGURE 6.1 – Représentation de la fonction test à une dimension.

Ackley

La fonction de Ackley est une fonction continue, multimodale obtenue en modulant une fonction exponentielle avec une fonction cosinus. Cette fonction est donnée comme suit

$$f(x) = -20 \exp\left(-0.2\sqrt{\frac{1}{r}\sum_{i=1}^{r} x_i^2}\right) - \exp\left(\frac{1}{r}\sum_{i=1}^{r}\cos(2\pi x_i)\right) + 20 + e, \tag{6.2}$$

où $r = 2$, $x \in \mathbb{R}^2$ et $x_i \in [-2, 2]$.

1. C'est pourquoi nous avons détaillé entièrement l'algorithme SMO au Chapitre 5.

Le minimum de cette fonction, représentée à la Figure 6.2, se trouve en $(0,0)$.

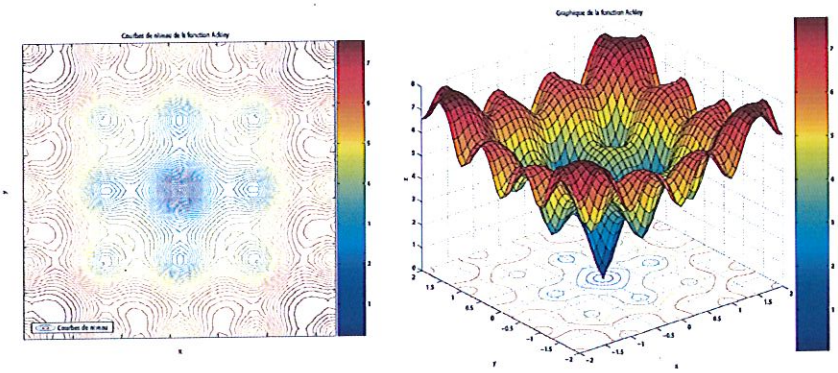


FIGURE 6.2 – Courbes de niveau et graphique de la fonction Ackley.

Ackley 10

La fonction de Ackley en 10 dimensions est simplement donnée par l'équation (6.2) avec $r = 10$ et $x \in \mathbb{R}^{10}$.

Branin

La fonction Branin est donnée comme suit

$$f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6.0\right)^2 + 10\left(1 - \frac{1}{8\pi}\cos(x_1)\right) + 10, \qquad (6.3)$$

où $x \in \mathbb{R}^2$, $x_1 \in [-5, 10]$ et $x_2 \in [0, 15]$.

Cette fonction, représentée à la Figure 6.3, présente 3 minima globaux.

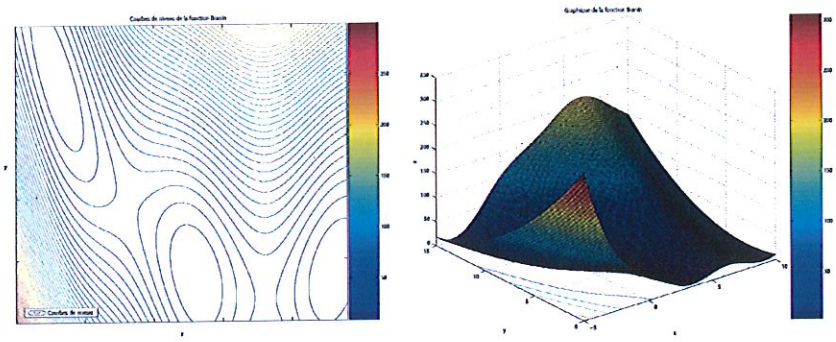


FIGURE 6.3 – Courbes de niveau et graphique de la fonction Branin.

Hosaki

La fonction Hosaki est donnée comme suit

$$f(x) = \left(1 - 8x_1 + 7x_1^2 - \frac{7}{3}x_1^3 + \frac{1}{4}x_1^4\right) x_2^2 e^{-x_2}, \tag{6.4}$$

où $x \in \mathbb{R}^2$, $x_1 \in [0,5]$ et $x_2 \in [0,5]$.

La représentation graphique de cette fonction est donnée à la Figure 6.4.

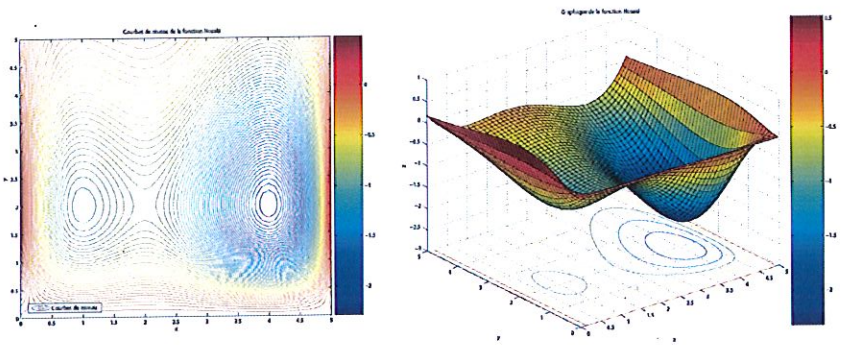


FIGURE 6.4 – Courbes de niveau et graphique de la fonction Hosaki.

La fonction de Hosaki présente un minimum local en $(1,2)$ et un minimum global en $(4,2)$.

Mystery

La fonction Mystery est une fonction multimodale donnée comme suit

$$f(x) = 2 + 0.01(x_2 - x_1^2)^2 + (1 - x_1)^2 + 2(2 - x_2)^2 + 7\sin(0.5x_1)\sin(0.7x_1x_2), \tag{6.5}$$

où $x \in \mathbb{R}^2$, $x_1 \in [-0.5, 5]$ et $x_2 \in [-0.5, 5]$.

Cette fonction, représentée à la Figure 6.5, présente 3 minima dont 1 global.

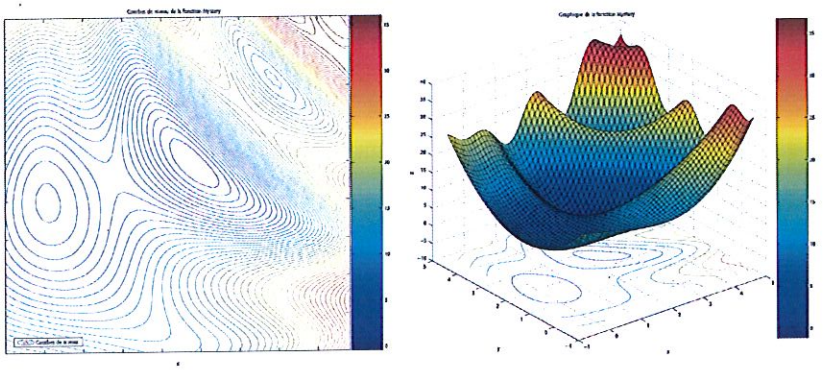


FIGURE 6.5 – Courbes de niveau et graphique de la fonction Mystery.

Rastrigin

La fonction Rastrigin est une fonction non convexe, définie par une somme de cosinus. Elle comprend plusieurs périodes donnant lieu à des vallées étroites et difficiles à interpoler. Elle est donnée comme suit

$$f(x) = \sum_{i=1}^{r} (x_i^2 - 10\cos(2\pi x_i)) + 10r, \tag{6.6}$$

où $r = 2$, $x \in \mathbb{R}^2$, $x_i \in [-1, 1]$.

La représentation graphique de cette fonction est donnée à la Figure 6.6.

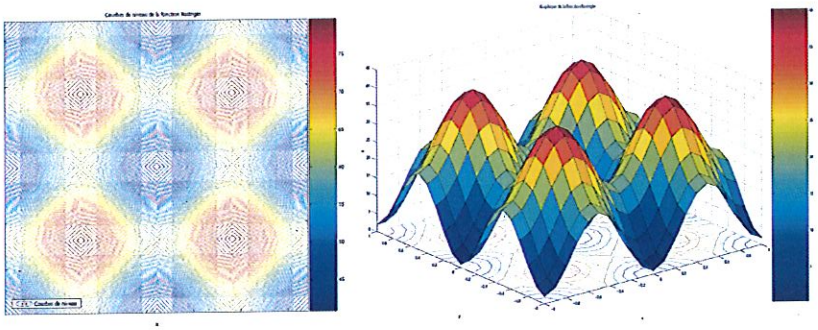


FIGURE 6.6 – Courbes de niveau et graphique de la fonction Rastrigin.

La fonction de Rastrigin présente un minimum global en $(0, 0)$.

Rastrigin 5 et 10

Les fonctions de Rastrigin en 5 et 10 dimensions sont données par (6.6) avec $r = 5$, $x \in \mathbb{R}^5$ et $r = 10$, $x \in \mathbb{R}^{10}$ respectivement.

Rosenbrock

La fonction Rosenbrock est une fonction non convexe, fortement anisotrope et pour laquelle le minimum se trouve dans une vallée étroite. C'est une fonction classique très souvent utilisée pour évaluer les performances d'algorithmes d'optimisation ou, dans notre cas, d'algorithme d'apprentissage. Elle est donnée comme suit

$$f(x) = \sum_{i=1}^{r-1}((1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2), \tag{6.7}$$

où $r = 2$, $x \in \mathbb{R}^2$, $x_i \in [-2, 2]$.

La représentation graphique de cette fonction est donnée à la Figure 6.7.

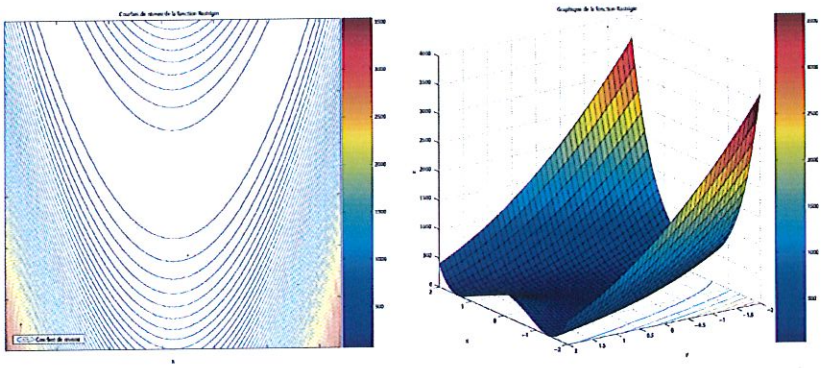


FIGURE 6.7 – Courbes de niveau et graphique de la fonction Rosenbrock.

La fonction Rosenbrock admet un minimum global en $(1, 1)$.

Rosenbrock 10 et 20

Les fonctions de Rosenbrock en 10 et 20 dimensions sont données par (6.7) avec $r = 10$, $x \in \mathbb{R}^{10}$ et $r = 20$, $x \in \mathbb{R}^{20}$ respectivement.

6.1.2 Validation des modèles

Afin d'évaluer la précision et la qualité de nos modèles, nous définissons trois types d'erreurs. Ces erreurs sont ensuite évaluées pour l'algorithme SMO et l'algorithme RBF "tuné" par la validation croisée (LOO) et par la validation externe. Les différents concepts concernant la validation des modèles sont expliqués respectivement à la Section 6.1.2.1 et 6.1.2.2.

6.1.2.1 Types d'erreurs mesurées

1. **Le coefficient de corrélation**

   Le coefficient de corrélation est défini par

$$R = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}\sqrt{\sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2}},$$

   où y_i désigne les valeurs cibles, $\hat{y}_i$ les valeurs prédites par le modèle et n le nombre de données de l'ensemble d'apprentissage. Le coefficient de corrélation permet de juger la qualité des modèles. Rappelons que plus il est proche de 1 meilleur est le modèle. Pour de plus amples informations le lecteur se référera à la Section 2.3.1. Ce coefficient est notamment utilisé lors de la procédure de sélection des meilleurs hyperparamètres (voir Section 5.3).

2. **L'erreur de validation (RMSE)**

   L'erreur de validation RMSE (pour Root Mean Squared Error) est définie par

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}, \tag{6.8}$$

   où y_i désigne les valeurs cibles, $\hat{y}_i$ les valeurs prédites par le modèle et n le nombre de données de l'ensemble d'apprentissage. L'erreur RMSE est donc la racine carrée de la moyenne arithmétique des carrés des écarts. Comme nous calculons le carré de ces écarts, nous augmentons l'importance des grosses erreurs. Dans le cadre de la régression, l'erreur RMSE doit être minimisée. Cette erreur est utilisée lors de la validation externe et LOO.

   Remarque

   Dans le cas de données bruitées, le modèle obtenu à partir d'une interpolation exacte des données bruitées présente une erreur élevée contrairement au modèle obtenu à partir d'une approximation.

3. **L'erreur absolue maximum (MAE)**

   L'erreur absolue maximum MAE (pour Maximum Absolute Error) est définie par

$$MAE = \max_{i=1,\ldots,n} |y_i - \hat{y}_i|, \tag{6.9}$$

   où y_i désigne les valeurs cibles, $\hat{y}_i$ les valeurs prédites par le modèle et n le nombre de données de l'ensemble d'apprentissage. Plus l'erreur MAE est faible, meilleur est le modèle. Cette erreur est utilisée lors de la validation externe et LOO.

6.1.2.2 Les Tableaux de validation

Pour chaque fonction analytique, nous effectuons une moyenne de 50 exécutions (i.e., 50 bases de données) de l'algorithme SMO et de l'algorithme RBF "tuné". Chacun des 50 modèles est alors évalué par validation croisée (LOO) et par validation externe. Les différents résultats sont alors présentés dans un tableau.

La validation LOO

Pour chaque modèle, nous calculons les trois types d'erreurs (voir Section 6.1.2.1) par la procédure LOO. Une fois les erreurs calculées, nous prenons la moyenne de celles-ci. Pour de plus amples informations concernant la procédure LOO le lecteur se référera à la Section 2.3.1.

La validation externe

La validation externe est un moyen efficace de vérifier la qualité d'un modèle et sa capacité de généralisation. Le principe de la validation externe consiste à générer un ensemble test contenant un grand nombre de données (dans notre cas 1000 données). Une fois cet ensemble test généré par une des méthodes de remplissage de l'espace (voir Section 1.2.4), nous construisons ensuite plusieurs modèles (dans notre cas 50 modèles) avec un ensemble d'apprentissage [2] généré par la technique LCVT. Pour chaque modèle, nous calculons les trois types d'erreurs pour l'ensemble test. Une fois les erreurs calculées, il suffit de prendre la moyenne de celles-ci.

La validation externe permet de mesurer précisément les différentes erreurs sur le modèle. Elle a été effectuée sur toutes les fonctions pour le modèle SVR et pour le modèle RBF "tuné" afin de pouvoir comparer les résultats obtenus avec ces deux modèles. Le modèle RBF "tuné" est expliqué à la Section 6.2.

La validation externe est une méthode coûteuse par rapport à LOO. Elle peut donc seulement être effectuée pour des fonctions analytiques mais pas avec de vraies simulations numériques.

6.2 Présentation des modèles RBF

Dans le logiciel d'optimisation Minamo, deux types de modèles RBF sont implémentés.

- *Le modèle par défaut* est un modèle RBF utilisant la fonction de base multiquadratique avec le paramètre largeur σ fixe (voir Section 2.5.2.2).
- *Le modèle RBF "tuné"* est un modèle RBF amélioré utilisant un heuristique basé sur une procédure efficace du LOO [57] qui permet de sélectionner un noyau (multiquadratique ou Gaussien) et une (des) largeur(s) pour les fonctions de base [3]. De plus, ces largeurs peuvent être fixes ou variables et ce, pour chaque fonction de base. Cet heuristique permet donc d'obtenir le meilleur modèle sélectionné en fonction du coefficient de corrélation. Pour de plus amples informations le lecteur peut se référer à [60].

2. L'ensemble d'apprentissage est bien sûr différent de l'ensemble test.
3. Chaque donnée de l'ensemble d'apprentissage est le centre d'une fonction de base radiale.

6.3 Illustrations des concepts relatifs aux SVR

Nous illustrons dans cette section les différents concepts de tube d'insensibilité ε, de vecteurs de support pour la régression et les multiplicateurs de Lagrange présentés aux Chapitres 4 et 5.

6.3.1 Sélection du modèle en fonction du tube d'insensibilité ε

La Figure 6.8 montre comment le modèle SVR sélectionne l'approximation la plus plane possible parmi toutes les approximations possibles des données de l'ensemble d'apprentissage en fonction de la taille du tube d'insensibilité ε. De plus, nous pouvons remarquer que le fait d'exiger que l'approximation soit la plus plane possible dans l'espace de redescription implique bien que le modèle soit lisse dans l'espace d'entrée.

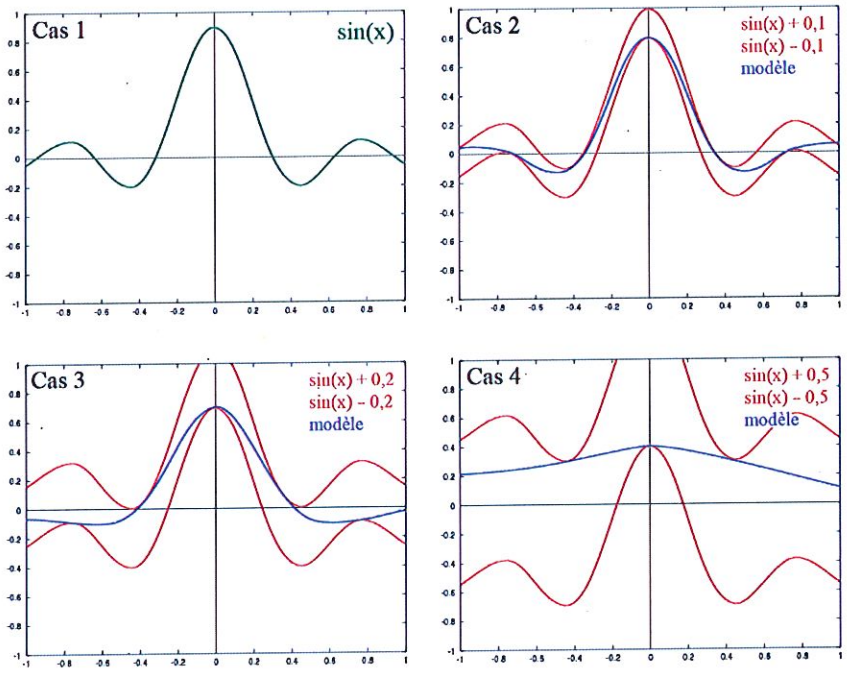


FIGURE 6.8 – Cas 1 : fonction originale $\sin(x)$, Cas 2 : approximation avec $\varepsilon = 0.1$, Cas 3 : approximation avec $\varepsilon = 0.2$ et Cas 4 : approximation avec $\varepsilon = 0.4$. Les courbes rouges indiquent les frontières supérieures et inférieures du tube d'insensibilité ε - source [11].

6.3.2 Relation entre la qualité de l'approximation et les vecteurs de support

La Figure 6.9 illustre la relation entre la qualité de l'approximation et le nombre de vecteurs de support. Nous constatons que moins la précision ε est importante, moins les vecteurs de support sont nombreux. Inversement, plus la précision est importante plus le nombre de vecteurs de support l'est aussi. De plus, quelle que soit la précision, seules quelques données sont considérées comme vecteurs de support (points verts), les points noirs étant les données d'apprentissage. Ceci illustre donc aussi le fait que l'ensemble des vecteurs de support est clairsemé.

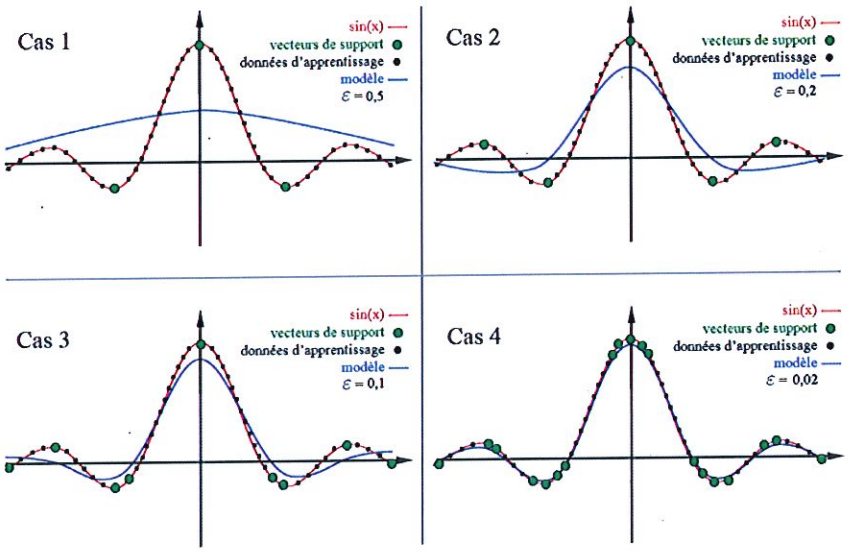


FIGURE 6.9 – Vecteurs de support et approximations (courbe bleue) pour différentes valeurs de ε - source [11].

6.3.3 Illustration de l'action des multiplicateurs de Lagrange

La Figure 6.10 illustre l'action des multiplicateurs de Lagrange agissant comme des "forces" (α_i, α_i^*) tirant et poussant sur l'approximation afin que celle-ci se situe à l'intérieur du tube d'insensibilité ε. Cependant, ces forces s'appliquent seulement sur les données où l'approximation touche ou dépasse les frontières du tube. Cette figure illustre donc les conditions de KKT : le modèle doit se trouver à l'intérieur du tube et, par conséquence, les multiplicateurs de Lagrange sont non nuls lorsque les contraintes sont actives et nuls sinon. Ce qui illustre donc bien les concepts présentés au Chapitre 5.

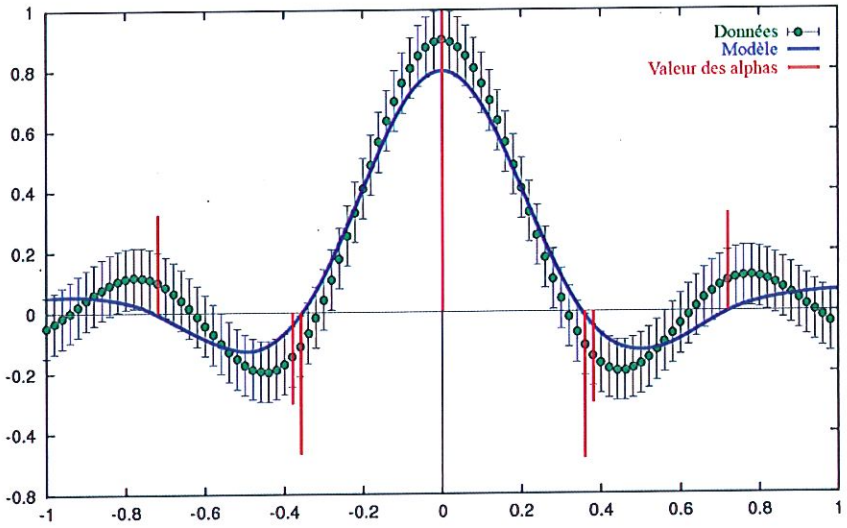


FIGURE 6.10 – Les multiplicateurs de Lagrange agissant comme des "forces" sur le modèle afin que celui-ci se situe à l'intérieur du tube d'insensibilité ε - source [11].

6.4 Résultats

Dans un premier temps, nous étudions la qualité des modèles construits pour des ensembles d'apprentissage bruités. Nous illustrons les rôles des paramètres ε et C et vérifions que les modèles sont bien indépendants des outliers. Deuxièmement, nous présentons les modèles calculés par notre modèle SVR pour chacunes des fonctions analytiques présentées à la Section 6.1.1. Chaque modèle est accompagné d'un tableau de validation externe et LOO (voir Section 6.1.2.2). Nous résumons ensuite les effets des différents hyperparamètres sur la qualité du modèle. Finalement, nous proposons une étude du temps d'exécution de l'algorithme SMO.

6.4.1 Performance en cas de données bruitées

"L'interpolation exacte (e.g., modèle polynomial de degré élevé voir Section 2.5.2.2) fournit un modèle oscillant lorsque les données sont bruitées (i.e., un modèle en sur-apprentissage). Ce bruit peut, par exemple, être dû à des erreurs numériques. En effet, la résolution de problèmes d'optimisation aérodynamique, aéroacoustique ou encore aéromécanique nécessite l'utilisation de simulations CFD ou CSM (Computational Structural Mechanic) qui présentent souvent du bruit suite à des erreurs d'ordre numérique liées à la discrétisation du maillage, au conditionnement du système à résoudre, ..." [58]. Nous émettons alors l'hypothèse qu'un modèle approché non oscillant (e.g., modèle SVR) devrait être plus représentatif de la fonction réelle, non bruitée. Comme nous l'avons vu au Chapitre 4, les SVR permettent de négliger certaines données de l'ensemble d'apprentissage. En effet, grâce aux variables de relaxation, le pro-

blème d'optimisation (4.18) permet d'obtenir un compromis entre la complexité du modèle et les erreurs plus grandes que ε tolérées. Ce compromis est déterminé par la constante C. Plus cette constante C est petite, plus l'approximation est plane (i.e., l'accent est mis davantage sur la minimisation de $\frac{1}{2}||w||_2^2$). A contrario, une constante C importante donne lieu à une approximation plus proche des données (i.e., l'accent est mis davantage sur la minimisation de $\sum_{i=1}^{n}(\xi_i+\xi_i^*)$). L'hyperparamètre C est donc un paramètre important à déterminer afin de construire un modèle ayant une bonne capacité de généralisation. Théoriquement, la méthode SVR permet donc de calculer un modèle robuste ne tenant pas compte des outliers. Afin d'illustrer cette propriété et les implications des hyperparamètres ε et C, nous présentons deux approximations de fonctions de dimension une et deux respectivement, pour des données bruitées et pour des jeux d'hyperparamètres différents. Pour cela, nous générons un ensemble de données d'apprentissage à l'aide de la technique de plan d'expériences LCVT et nous leurs ajoutons ensuite un bruit gaussien (i.e., bruit suivant une loi normale) de moyenne 0 et de variance égale à 0.5. La Figure 6.11 illustre le bruit gaussien pour différentes valeurs de la variance.

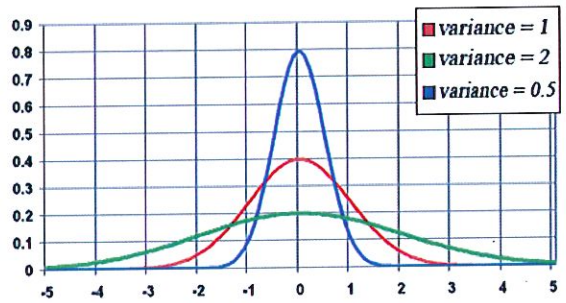


FIGURE 6.11 – Bruit gaussien pour différentes valeurs de la variance.

Remarque

Par la suite, tous les coefficients de corrélation présentés sont obtenus par la procédure de validation croisée (voir Section 2.3.1).

Fonction à une dimension

Nous approximons la fonction à une dimension (6.1) définie à la Section 6.1.1 avec un ensemble d'apprentissage de 40 données bruitées.

- En premier lieu, nous illustrons le rôle du paramètre ε dans le modèle SVR. Nous considérons pour cela un jeu d'hyperparamètres contenant un ε important.

σ	ε	C
0.2	0.1	1

Nous obtenons le modèle représenté à la Figure 6.12, de coefficient de corrélation $R = 0.907$.

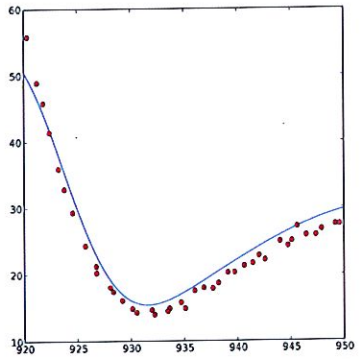


FIGURE 6.12 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.1$ et $C = 1$, pour 40 données bruitées et pour la fonction (6.1).

Le modèle obtenu comporte 10 VS et nous remarquons tout de suite qu'il ne colle quasi pas aux données de l'ensemble d'apprentissage. Un hyperparamètre ε trop important n'est donc pas un bon choix.

- Deuxièmement, nous illustrons le rôle du paramètre C dans le modèle SVR. Nous considérons pour cela un jeu d'hyperparamètres contenant un C important et un ε plus petit.

σ	ε	C
0.2	0.001	100

Nous obtenons le modèle représenté à la Figure 6.13, de coefficient de corrélation $R = 0.995$.

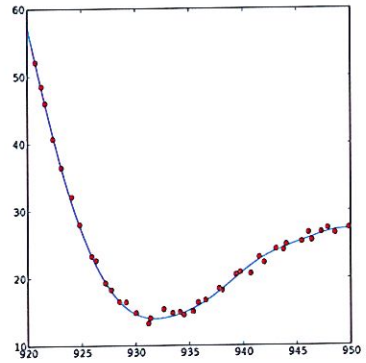


FIGURE 6.13 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 100$, pour 40 données bruitées et pour la fonction (6.1).

Contrairement au modèle précédent, le modèle obtenu comporte 39 VS et, par comparaison avec la vraie fonction à une dimension (Figure 6.1), est très bon malgré la présence de données bruitées. Ceci illustre bien l'importance du choix de l'hyperparamètre C.

- Finalement, notre procédure de sélection des hyperparamètres (voir Section 5.3) nous renvoie le jeu d'hyperparamètres optimal suivant

σ	ε	C
0.1	0.001	1

qui correspond au modèle représenté à la Figure 6.14, de coefficient de corrélation $R = 0.990$.

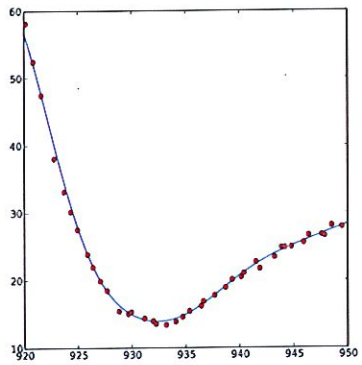


FIGURE 6.14 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.1$, $\varepsilon = 0.001$ et $C = 1$, pour 40 données bruitées et pour la fonction (6.1).

Le modèle calculé comporte 39 VS. Comparé à la vraie fonction (voir Figure 6.1) et au modèle obtenu à l'aide de données non bruitées (voir Section 6.4.2), le modèle construit est très bon. Cependant, nous remarquons que l'hyperparamètre C déterminé est égal à 1 dû au critère d'arrêt (i.e. si le coefficient de corrélation $R > 0.99$ alors la recherche des hyperparamètres est arrêtée). Si nous enlevons ce critère d'arrêt, un hyperparamètre C supérieur serait sélectionné. En effet, nous avons vu qu'avec un hyperparamètre $C = 100$, un meilleur modèle peut être calculé.

Fonction Hosaki

Nous approximons la fonction Hosaki (6.4) définie à la Section 6.1.1 avec un ensemble d'apprentissage de 40 données bruitées.

- Comme précédemment, afin d'illustrer le rôle du paramètre ε, nous considérons un jeu d'hyperparamètres contenant un ε important.

σ	ε	C
0.4	0.1	1

Nous obtenons le modèle représenté à la Figure 6.15, de coefficient de corrélation $R = 0.8375$.

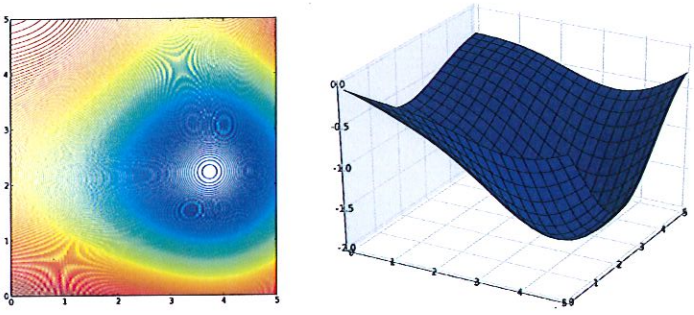


FIGURE 6.15 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.1$ et $C = 1$, pour 40 données bruitées et pour la fonction (6.4).

Le modèle obtenu comporte 24 VS et nous remarquons directement qu'il ne correspond pas à la fonction Hosaki (Figure 6.4). Encore une fois, un hyperparamètre ε trop important n'est pas un bon choix.

- Deuxièmement, afin d'illustrer le rôle du paramètre C, nous considérons un jeu d'hyperparamètres contenant un C important et un ε plus petit.

σ	ε	C
0.4	0.001	20

Nous obtenons le modèle représenté à la Figure 6.16, de coefficient de corrélation $R = 0.9338$.

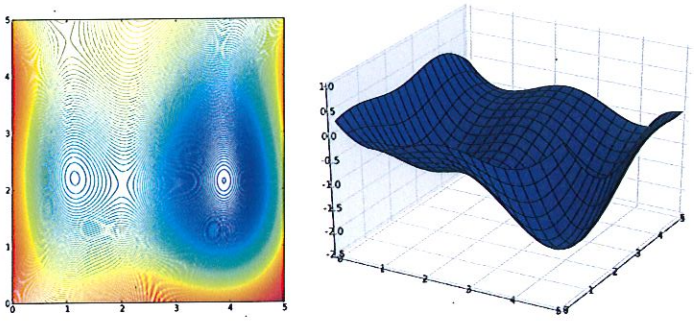


FIGURE 6.16 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 20$, pour 40 données bruitées et pour la fonction (6.4).

Contrairement au modèle précédent, le modèle obtenu comporte 40 VS et, par comparaison avec la vraie fonction Hosaki (Figure 6.4), est bon malgré la présence de données bruitées, illustrant encore une fois l'importance de l'hyperparamètre C.

- Finalement, notre procédure de sélection des hyperparamètres nous renvoie le jeu d'hyperparamètres suivant

σ	ε	C
0.4	0.001	28.1

Nous obtenons le modèle représenté à la Figure 6.17, de coefficient de corrélation $R = 0.947$.

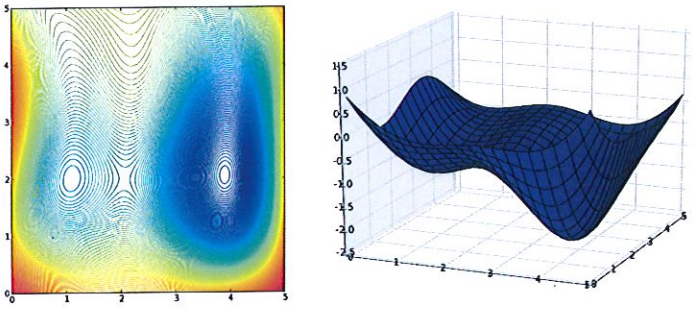


FIGURE 6.17 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 28.1$, pour 40 données bruitées et pour la fonction (6.4).

Comparé à la vraie fonction (voir Figure 6.4) et au modèle obtenu à l'aide de données non bruitées (voir Section 6.4.2), le modèle construit qui contient 39 VS est très bon. Nous remarquons dans cet exemple que l'hyperparamètre C déterminé est supérieur à 1 dû aux données bruitées, ce qui permet de pénaliser plus fortement les outliers et de construire un modèle quasiment parfait.

Tableaux de validation pour des données bruitées

Nous présentons maintenant quelques tableaux de validation externe et LOO pour des modèles calculés à partir d'ensembles d'apprentissage bruités. Les coefficients de corrélation ainsi que les erreurs RMSE et MAE sont mesurés par rapport aux données cibles non bruitées.

1. *Tableaux de validation de la fonction Ackley*

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$8.764e-01$	$9.718e-01$	$2.490e+00$	$8.974e-01$	$9.898e-01$	$4.244e+00$
SVR	$9.084e-01$	$8.369e-01$	$2.074e+00$	$9.229e-01$	$9.246e-01$	$3.422e+00$

Tableau 6.1 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Ackley avec 40 données bruitées.

2. *Tableaux de validation de la fonction Rastrigin*

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$9.191e-01$	$4.012e+00$	$1.375e+01$	$9.791e-01$	$2.051e+00$	$1.504e+01$
SVR	$9.075e-01$	$4.010e+00$	$1.197e+01$	$9.851e-01$	$1.729e+00$	$9.889e+00$

Tableau 6.2 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rastrigin avec 40 données bruitées.

3. *Tableaux de validation de la fonction Rastrigin* 10

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$4.810e-01$	$2.155e+01$	$6.199e+01$	$3.546e-02$	$3.040e+01$	$1.297e+02$
SVR	$4.154e-01$	$2.425e+01$	$6.959e+01$	$3.338e-02$	$3.268e+01$	$1.338e+02$

Tableau 6.3 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rastrigin 10 avec 40 données bruitées.

4. *Tableaux de validation de la fonction Hosaki*

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$9.850e-01$	$9.965e-02$	$4.448e-01$	$9.973e-01$	$4.481e-01$	$8.093e-01$
SVR	$9.387e-01$	$1.998e-01$	$7.074e-01$	$9.807e-01$	$4.609e-01$	$1.071e+00$

Tableau 6.4 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Hosaki avec 40 données bruitées.

Pour les Tableaux 6.1 et 6.2 nous pouvons constater que les coefficients de corrélations obtenus sont plus proches de 1 que ceux obtenus pour le modèle RBF "tuné". De même, les erreurs RMSE et MAE sont plus faibles pour les modèles SVR que pour les modèles RBF "tuné". Nous pouvons donc en conclure que les modèles SVR sont meilleurs que les modèles RBF "tuné". Par contre, pour les Tableaux 6.3 et 6.4, les coefficients de corrélations des modèles SVR sont plus faibles que ceux des modèles RBF "tuné". Les erreurs RMSE et MAE sont aussi un peu plus élevées que celles des modèles RBF "tuné". Cependant, les coefficients de corrélation ainsi que les erreurs RMSE et MAE restent fort proches de ceux du modèle RBF "tuné". Nous pouvons donc dire que les modèles SVR sont un peu moins bons voire quasiment équivalents aux modèles RBF "tuné".

Conclusion

Les approximations de la fonction à une dimension et de la fonction de Hosaki par le modèle SVR pour un ensemble d'apprentissage bruité, ainsi que les différents tableaux de validations illustrent bien la propriété d'indépendance des modèles SVR par rapport aux outliers ainsi que l'importance du choix des hyperparamètres pour la construction de modèles robustes.

6.4.2 Etude des modèles obtenus pour chaque fonction test et de la capacité de généralisation

Dans cette section, nous présentons les modèles obtenus pour toutes les fonctions analytiques tests pour différents ensembles d'apprentissage non bruités. Une validation externe et LOO sont ensuite proposées pour chaque fonction test.

Fonction à une dimension

Nous présentons le modèle obtenu par le modèle SVR pour la fonction à une dimension (6.1) pour un ensemble d'apprentissage de 40 données. Notre procédure de sélection des hyperparamètres nous renvoie le jeu d'hyperparamètres suivant

σ	ε	C
0.1	0.001	9.04

qui correspond au modèle représenté à la Figure 6.18, de coefficient de corrélation $R = 0.996$.

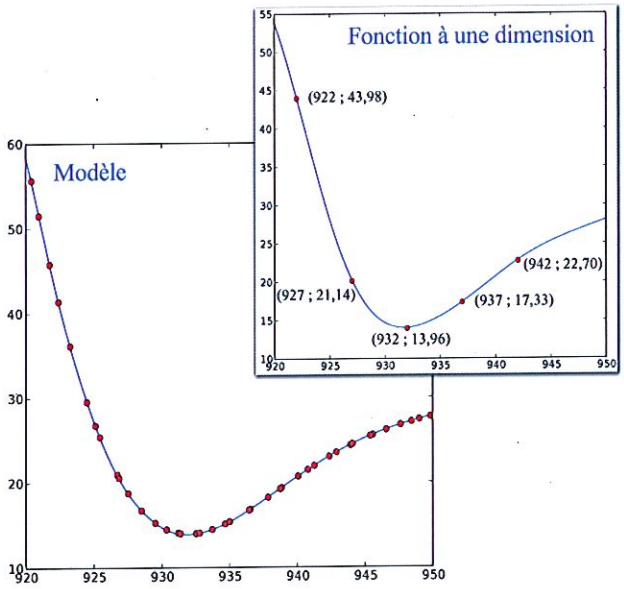


FIGURE 6.18 – Modèle obtenu avec 40 données non bruitées pour le jeu d'hyperparamètres $\sigma = 0.1$, $\varepsilon = 0.001$ et $C = 9.04$, pour 40 données et pour la fonction (6.1).

Comparé à la vraie fonction (Figure 6.1) et au vu du coefficient de corrélation, le modèle calculé, comportant 31 VS, est parfait. De très bons modèles sont également obtenus avec des ensembles de validation contenant moins de données (minimum 10 données).

Ackley

En premier lieu, nous présentons les modèles obtenus pour la fonction Ackley (6.2) par le modèle SVR pour un ensemble d'apprentissage de 40 et 80 données.

- Pour un ensemble d'apprentissage [4] de 40 données, notre procédure de sélection des hyperparamètres nous renvoie le jeu d'hyperparamètres suivant

σ	ε	C
0.4	0.001	28.1

Nous obtenons le modèle représenté à la Figure 6.19, de coefficient de corrélation $R = 0.745$.

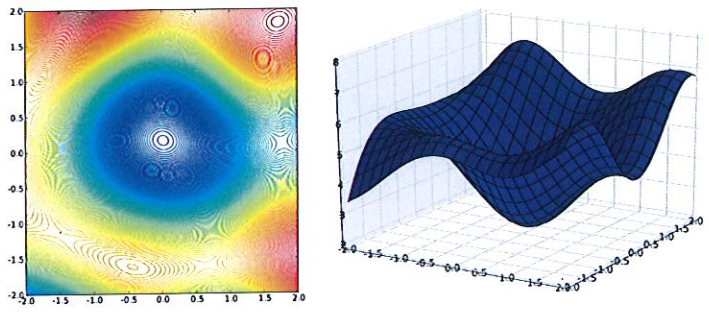


FIGURE 6.19 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 28.1$, pour 40 données et pour la fonction (6.2).

Le modèle obtenu comporte 40 VS. Comparé à la vraie fonction (Figure 6.2), le modèle construit n'est pas optimal. Cependant, l'allure générale de la fonction est respectée. Pour obtenir un meilleur modèle il suffit simplement de considérer un ensemble d'apprentissage contenant plus de données.

- Suite aux remarques précédentes, nous considérons un ensemble d'apprentissage de 80 données. Notre procédure de sélection des hyperparamètres nous renvoie alors le nouveau jeu d'hyperparamètres suivant

σ	ε	C
0.1	0.001	52.83

Nous obtenons le modèle représenté à la Figure 6.20, de coefficient de corrélation $R = 0.921$.

4. Exceptionnellement, les données d'apprentissage sont générées par la procédure LHS (voir Section 1.2.4). Pour obtenir des résultats corrects avec la procédure LCVT, il faut considérer un ensemble d'apprentissage d'au moins 100 données.

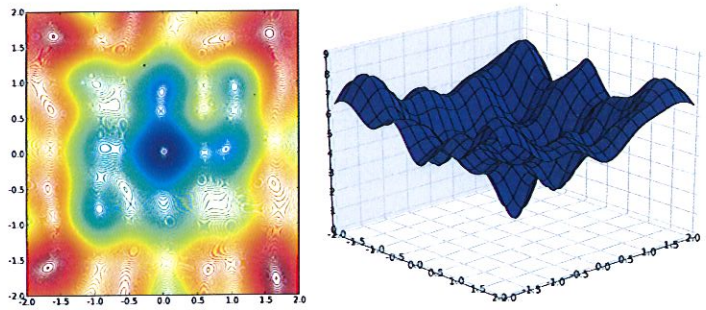


FIGURE 6.20 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.1$, $\varepsilon = 0.001$ et $C = 52.83$, pour 80 données et pour la fonction (6.2).

Le modèle obtenu comporte 79 VS. Comparé à la vraie fonction (Figure 6.2), le modèle construit est bon. En effet les courbes de niveau respectent l'allure de celles de la fonction Ackley.

Finalement, afin de vérifier la qualité des modèles construits par le modèle SVR, nous effectuons une validation externe et LOO. Les résultats obtenus sont donnés dans le Tableau 6.5.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$8.773e-01$	$9.796e-01$	$2.357e+00$	$8.955e-01$	$8.750e-01$	$3.681e+00$
SVR	$9.151e-01$	$8.076e-01$	$2.033e+00$	$9.227e-01$	$7.988e-01$	$3.106e+00$

Tableau 6.5 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Ackley avec 40 données.

Nous pouvons directement constater que les coefficients de corrélation obtenus par le modèle SVR sont plus proches de 1 que ceux obtenus par l'algorithme RBF "tuné" et sont donc meilleurs. De plus, les erreurs RMSE et MAE sont plus faibles pour le modèle SVR, ce qui implique que le modèle SVR construit un modèle plus robuste pour cette fonction.

Ackley 10

Pour le cas de la fonction Ackley en 10 dimensions, nous analysons uniquement les statistiques (R, RMSE, MAE) via LOO et via la validation externe. Les résultats obtenus sont présentés dans le Tableau 6.6.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$5.145e-01$	$4.958e-01$	$1.568e+00$	$6.483e-01$	$5.223e-01$	$2.337e+00$
SVR	$5.468e-01$	$4.840e-01$	$1.568e+00$	$6.488e-01$	$5.204e-01$	$2.381e+00$

Tableau 6.6 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Ackley 10 avec 40 données.

Nous remarquons que les résultats sont assez mauvais du fait du peu de données d'apprentissage. En effet, les coefficients de corrélation sont assez faibles. C'est pourquoi, nous considérons directement un nouvel ensemble d'apprentissage de 100 données. Le Tableau 6.7 reprend les nouveaux résultats.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
100 données						
RBF "tuné"	$7.576e-01$	$3.726e-01$	$1.671e+00$	$8.377e-01$	$3.786e-01$	$1.455e+00$
SVR	$7.378e-01$	$3.916e-01$	$1.579e+00$	$8.052e-01$	$4.097e-01$	$1.470e+00$

Tableau 6.7 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Ackley 10 avec 100 données.

Nous remarquons que les coefficients de corrélation obtenus pour la validation externe sont beaucoup plus proches de 1 et sont donc bien meilleur que ceux obtenus au Tableau 6.6. De même, les erreurs RMSE et MAE sont plus faibles que celles obtenues au Tableau 6.6 pour le modèle construit avec 40 données. Finalement, nous pouvons constater que les coefficients de corrélation et les erreurs RMSE et MAE obtenus par le modèle SVR et par l'algorithme RBF "tuné" sont quasiment équivalents, ce qui implique que le modèle SVR construit un modèle aussi bon que l'algorithme RBF "tuné".

Hosaki

En premier lieu, nous présentons les modèles obtenus pour la fonction Hosaki (6.4) par le modèle SVR pour un ensemble d'apprentissage de 20 et 40 données.

- Pour un ensemble d'apprentissage de 20 données, notre procédure de sélection des hyperparamètres nous renvoie le jeu d'hyperparamètres suivant

σ	ε	C
0.4	0.001	28.1

  Nous obtenons le modèle représenté à la Figure 6.21, de coefficient de corrélation $R = 0.844$.

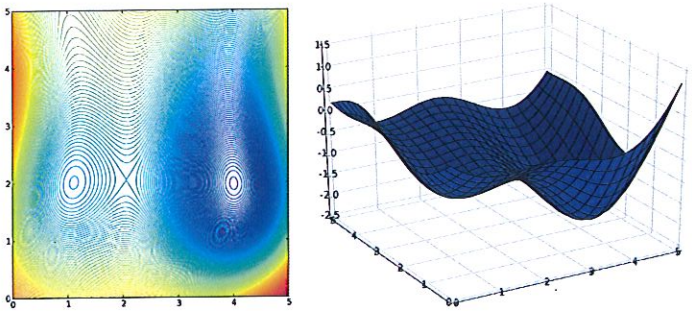


FIGURE 6.21 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 28.1$, pour 20 données et pour la fonction (6.4).

Le modèle obtenu comporte 20 VS. Comparé à la vraie fonction (Figure 6.4), à ses courbes de niveau et compte tenu du fait que l'ensemble d'apprentissage ne contient que 20 données, le modèle construit est déjà assez bon.

- Nous considérons maintenant un ensemble d'apprentissage de 40 données, notre procédure de sélection des hyperparamètres nous renvoie alors le nouveau jeu d'hyperparamètres suivant

σ	ε	C
0.4	0.001	28.1

qui correspond au modèle représenté à la Figure 6.22, de coefficient de corrélation $R = 0.956$.

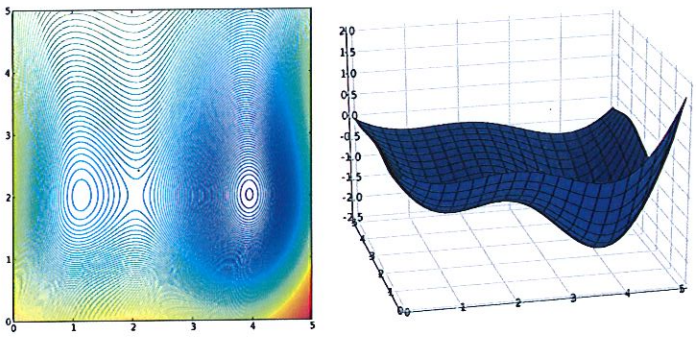


FIGURE 6.22 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.4$, $\varepsilon = 0.001$ et $C = 28.1$, pour 40 données et pour la fonction (6.4).

Le modèle obtenu comporte 39 VS. Comparé à la vraie fonction (Figure 6.4), le modèle construit est très bon suite à l'ajout de nouvelles données d'apprentissage. En effet, les courbes de niveau respectent quasiment parfaitement l'allure de celles de la vraie fonction de Hosaki.

Finalement, afin de vérifier la qualité des modèles construits par le modèle SVR, nous effectuons une validation externe et LOO. Les résultats obtenus pour un ensemble d'apprentissage de 30 et 40 données sont présentés respectivement dans les Tableaux 6.8 et 6.9.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
30 données						
RBF "tuné"	$9.277e-01$	$2.241e-01$	$7.435e-01$	$9.812e-01$	$1.071e-01$	$1.040e+00$
SVR	$8.904e-01$	$2.709e-01$	$8.635e-01$	$9.627e-01$	$1.564e-01$	$1.096e+00$

Tableau 6.8 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Hosaki avec 30 données.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$9.843e-01$	$1.100e-01$	$4.490e-01$	$9.960e-01$	$4.523e-02$	$4.582e-01$
SVR	$9.443e-01$	$1.915e-01$	$6.801e-01$	$9.813e-01$	$1.144e-01$	$7.969e-01$

Tableau 6.9 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Hosaki avec 40 données.

Les résultats obtenus pour la validation externe avec 30 et 40 données pour le modèle SVR sont bons et très proches de ceux de l'algorithme RBF "tuné". En effet, les coefficients de corrélation sont très proches de 1 et ce, quels que soient le modèle et le nombre de données d'apprentissage. Nous pouvons donc dire que le modèle SVR construit un modèle quasiment aussi robuste que l'algorithme RBF "tuné".

Branin

En premier lieu, nous présentons les modèles obtenus pour la fonction Branin (6.3) par le modèle SVR pour un ensemble d'apprentissage de 40 et 80 données.

- Pour un ensemble d'apprentissage de 40 données, notre procédure de sélection des hyperparamètres nous renvoie le jeu d'hyperparamètres suivant

σ	ε	C
0.2	0.001	1

qui correspond au modèle représenté à la Figure 6.23, de coefficient de corrélation $R = 0.936$.

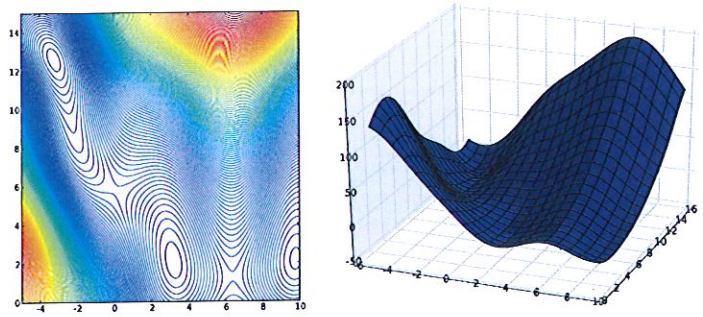


FIGURE 6.23 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 1$, pour 40 données et pour la fonction (6.3).

Le modèle construit comporte 40 VS. Comparé à la vraie fonction (Figure 6.3) et à ses courbes de niveau, le modèle construit est déjà assez bon.

- Nous considérons maintenant un ensemble d'apprentissage de 80 données, notre procédure de sélection des hyperparamètres nous renvoie alors le nouveau jeu d'hyperparamètres suivant

σ	ε	C
0.2	0.001	9.04

qui correspond au modèle représenté à la Figure 6.24, de coefficient de corrélation $R = 0.997$.

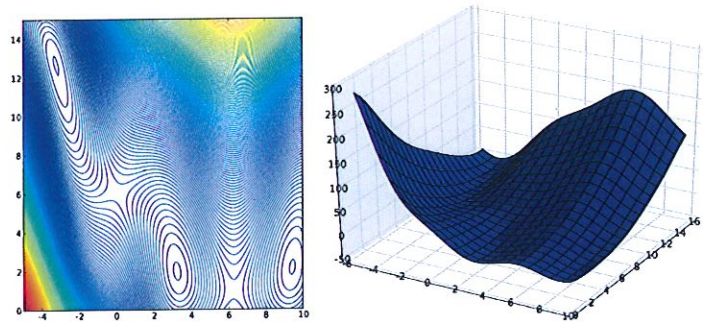


FIGURE 6.24 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 9.04$, pour 80 données et pour la fonction (6.3).

Le modèle obtenu comporte 73 VS. Comparé à la vraie fonction (Figure 6.3), le modèle construit est très bon suite à l'ajout de nouvelles données d'apprentissage. En effet, les courbes de niveau respectent, encore une fois, quasiment parfaitement, l'allure de celles de la fonction de Branin.

Finalement, afin de vérifier la qualité des modèles construits par la méthode SVR, nous effectuons une validation externe et LOO. Les résultats obtenus pour un ensemble d'apprentissage de 30 et 40 données sont représentés respectivement dans les Tableaux 6.10 et 6.11.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
30 données						
RBF "tuné"	$9.478e-01$	$1.549e+01$	$4.795e+01$	$9.842e-01$	$8.906e+00$	$7.852e+01$
SVR	$9.437e-01$	$1.659e+01$	$5.089e+01$	$9.773e-01$	$1.226e+01$	$8.599e+01$

Tableau 6.10 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Branin avec 30 données.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$9.886e-01$	$7.290e+00$	$2.452e+01$	$9.977e-01$	$3.321e+00$	$3.367e+01$
SVR	$9.657e-01$	$1.302e+01$	$4.071e+01$	$9.856e-01$	$9.574e+00$	$6.624e+01$

Tableau 6.11 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Branin avec 40 données.

Les résultats obtenus pour la validation externe avec 30 et 40 données pour le modèle SVR sont bons et fort proches de ceux de l'algorithme RBF "tuné". En effet, les coefficients de corrélation sont fort proches de 1. Encore une fois, nous pouvons donc dire que le modèle SVR construit un modèle quasiment aussi bon que l'algorithme RBF "tuné".

Mystery

En premier lieu, nous présentons les modèles obtenus pour la fonction Mystery (6.5) par le modèle SVR pour un ensemble d'apprentissage de 40 et 80 données.

- Pour un ensemble d'apprentissage de 40 données, notre procédure de sélection des hyperparamètres nous renvoie le jeu d'hyperparamètres suivant

σ	ε	C
0.3	0.001	9.04

qui correspond au modèle représenté à la Figure 6.25, de coefficient de corrélation $R = 0.815$.

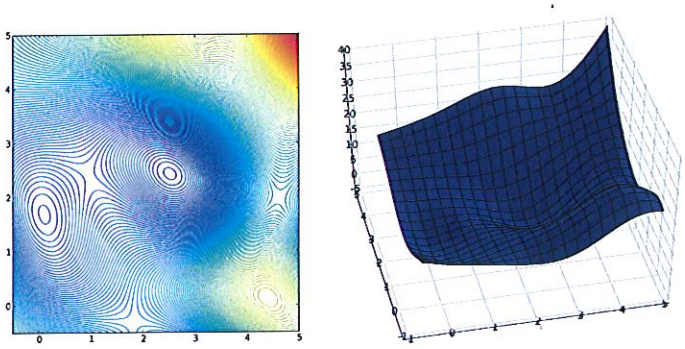


FIGURE 6.25 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.3$, $\varepsilon = 0.001$ et $C = 9.04$, pour 40 données et pour la fonction (6.5).

Le modèle obtenu comporte 40 VS. Comparé à la vraie fonction (6.5) et à ses courbes de niveau, le modèle calculé est assez bon. La fonction Mystery étant complexe à approximer, nous devons considérer un nombre de données d'apprentissage plus important afin d'améliorer nos résultats.

- Nous considérons maintenant un ensemble d'apprentissage de 80 données, notre procédure de sélection des hyperparamètres nous renvoie alors le nouveau jeu d'hyperparamètres suivant

σ	ε	C
0.1	0.001	1

qui correspond au modèle représenté à la Figure 6.26, de coefficient de corrélation $R = 0.956$.

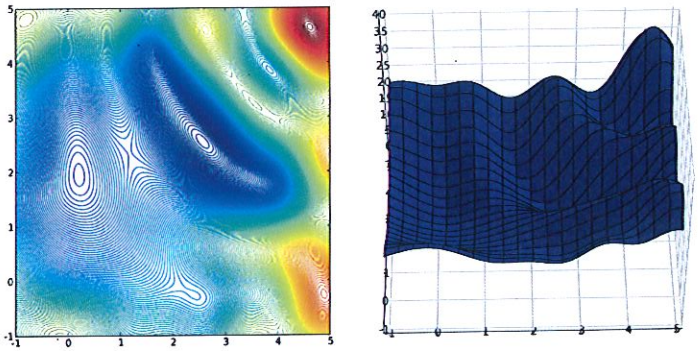


FIGURE 6.26 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.1$, $\varepsilon = 0.001$ et $C = 1$, pour 80 données et pour la fonction (6.5).

Comparé à la vraie fonction (6.5), le modèle construit comportant 78 VS est bon suite à l'ajout de nouvelles données d'apprentissage. En effet, les courbes de niveau respectent, encore une fois quasiment parfaitement, l'allure de celles de la fonction Mystery.

Finalement, afin de vérifier la qualité des modèles construits par le modèle SVR, nous effectuons une validation externe et LOO. Les résultats obtenus pour un ensemble d'apprentissage de 40 données sont inscrits dans le Tableau 6.12.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$8.167e-01$	$4.629e+00$	$1.452e+01$	$9.481e-01$	$2.638e+00$	$2.273e+01$
SVR	$7.943e-01$	$5.012e+00$	$1.268e+01$	$8.846e-01$	$4.373e+00$	$1.266e+01$

Tableau 6.12 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Mystery avec 40 données.

Les résultats obtenus pour la validation externe pour le modèle SVR sont bons et fort proches de ceux de l'algorithme RBF "tuné". Remarquons cependant que l'erreur MAE est plus faible avec la méthode SVR. Une fois de plus, nous pouvons conclure que le modèle SVR calcule un modèle quasiment aussi bon que l'algorithme RBF "tuné".

Rastrigin

En premier lieu, nous présentons les modèles obtenus pour la fonction Rastrigin (6.6) par le modèle SVR pour un ensemble d'apprentissage de 40 et 60 données.

- Pour un ensemble d'apprentissage de 40 données, notre procédure de sélection des hyperparamètres nous renvoie le jeu d'hyperparamètres suivant

σ	ε	C
0.2	0.001	52.83

qui correspond au modèle représenté à la Figure 6.27, de coefficient de corrélation $R = 0.934$.

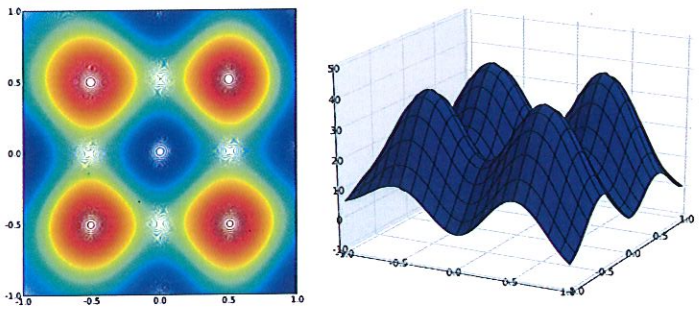


FIGURE 6.27 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 52.83$, pour 40 données et pour la fonction (6.6).

Le modèle obtenu comporte 40 VS. Comparé à la vraie fonction (Figure 6.6), à ses courbes de niveau et au vu du coefficient de corrélation, le modèle construit est déjà très bon.

- Nous considérons maintenant un ensemble d'apprentissage de 60 données, notre procédure de sélection des hyperparamètres nous renvoie alors le nouveau jeu d'hyperparamètres suivant

σ	ε	C
0.2	0.001	9.04

qui correspond au modèle représenté à la Figure 6.28, de coefficient de corrélation $R = 0.986$.

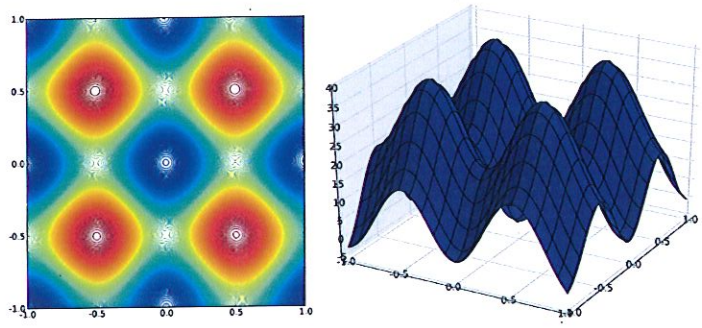


FIGURE 6.28 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.2$, $\varepsilon = 0.001$ et $C = 9.04$, pour 60 données et pour la fonction (6.6).

Le modèle calculé comporte 59 VS. Comparé au modèle précédent, l'ajout de nouvelles données d'apprentissage permet d'obtenir un modèle encore plus précis. En effet, les courbes de niveau respectent, parfaitement l'allure de celles de la vraie fonction de Rastrigin (Figure 6.6).

Finalement, afin de vérifier la qualité des modèles construits par le modèle SVR, nous effectuons une validation externe et LOO. Les résultats obtenus pour un ensemble d'apprentissage de 40 données sont notés dans le Tableau 6.13.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$9.133e-01$	$4.121e+00$	$1.299e+01$	$9.761e-01$	$2.156e+00$	$1.622e+01$
SVR	$9.044e-01$	$4.090e+00$	$1.181e+01$	$9.850e-01$	$1.688e+00$	$1.020e+01$

Tableau 6.13 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rastrigin avec 40 données.

Les coefficients de corrélation ainsi que les valeurs des erreurs RMSE et MAE obtenus pour la validation externe pour le modèle SVR sont meilleurs que ceux de l'algorithme RBF "tuné". Nous pouvons conclure que les modèles construits par le modèle SVR sont meilleurs que ceux de l'algorithme RBF "tuné".

Rastrigin 5

Pour le cas de la fonction Rastrigin en 5 dimensions, nous analysons uniquement les statistiques (R, RMSE, MAE) de la validation externe et LOO. Les résultats obtenus pour un ensemble d'apprentissage de 40 et 100 données sont présentés respectivement dans les Tableaux 6.14 et 6.15.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$5.555e-02$	$3.829e+01$	$9.613e+01$	$-3.098e-02$	$3.524e+01$	$1.472e+02$
SVR	$-2.342e-02$	$2.907e+01$	$7.269e+01$	$1.534e-02$	$2.523e+01$	$8.257e+01$

Tableau 6.14 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rastrigin 5 avec 40 données.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
100 données						
RBF "tuné"	$-1.401e-02$	$3.647e+01$	$1.313e+02$	$1.471e-02$	$3.009e+01$	$1.628e+02$
SVR	$-6.947e-02$	$2.241e+01$	$6.834e+01$	$-2.966e-02$	$2.164e+01$	$8.140e+01$

Tableau 6.15 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rastrigin 5 avec 100 données.

Malheureusement, les résultats obtenus pour l'approximation de la fonction de Rastrigin en 5 dimensions sont mauvais car les coefficients de corrélation sont négatifs et ce, quels que soit le modèle utilisé. Nous pouvons cependant remarquer que les erreurs RMSE et MAE sont plus faibles que pour le modèle SVR. Afin d'obtenir de meilleurs résultats il faudrait considérer un ensemble d'apprentissage contenant beaucoup plus de données.

Rastrigin 10

Pour le cas de la fonction Rastrigin en 10 dimensions, nous ne présentons que les résultats de la validation externe et LOO. Les résultats obtenus pour un ensemble d'apprentissage de 40 et 150 données sont notés respectivement dans les Tableaux 6.16 et 6.17.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$3.600e-01$	$2.684e+01$	$6.803e+01$	$3.732e-02$	$3.118e+01$	$1.080e+02$
SVR	$1.947e-01$	$2.718e+01$	$7.525e+01$	$2.919e-02$	$2.960e+01$	$9.919e+01$

Tableau 6.16 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rastrigin 10 avec 40 données.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
150 données						
RBF "tuné"	$5.026e-01$	$2.161e+01$	$6.164e+01$	$3.902e-02$	$3.094e+01$	$1.330e+02$
SVR	$4.306e-01$	$2.463e+01$	$6.820e+01$	$3.382e-02$	$3.349e+01$	$1.388e+02$

Tableau 6.17 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rastrigin 10 avec 150 données.

Nous pouvons directement constater que les résultats obtenus pour l'ensemble d'apprentissage de 40 données ne sont pas bons du fait du peu de données. Pour 150 données d'apprentissage, les résultats sont un peu meilleurs mais les performances du modèle SVR sont un peu inférieures à celles de l'algorithme RBF "tuné".

Rosenbrock

En premier lieu, nous présentons les modèles obtenus pour la fonction Rosenbrock (6.7) par le modèle SVR pour un ensemble d'apprentissage de 40 et 80 données.

- Pour un ensemble d'apprentissage de 40 données, notre procédure de sélection des hyperparamètres nous renvoie le jeu d'hyperparamètres suivant

σ	ε	C
0.5	0.001	52.83

qui correspond au modèle représenté à la Figure 6.29, de coefficient de corrélation $R = 0.980$.

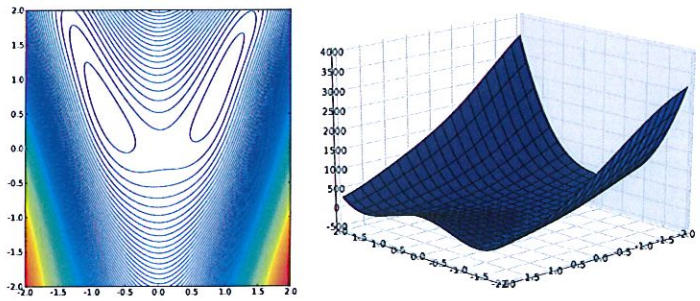


FIGURE 6.29 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.5$, $\varepsilon = 0.001$ et $C = 52.83$, pour 40 données et pour la fonction (6.7).

Le modèle calculé comporte 38 VS. Comparé à la vraie fonction (Figure 6.7), à ses courbes de niveau et au vu du coefficient de corrélation, le modèle construit est déjà très bon.

- Nous considérons maintenant un ensemble d'apprentissage de 80 données, notre procédure de sélection des hyperparamètres nous renvoie alors le nouveau jeu d'hyperparamètres suivant

σ	ε	C
0.3	0.001	9.04

qui correspond au modèle représenté à la Figure 6.30, de coefficient de corrélation $R = 0.997$.

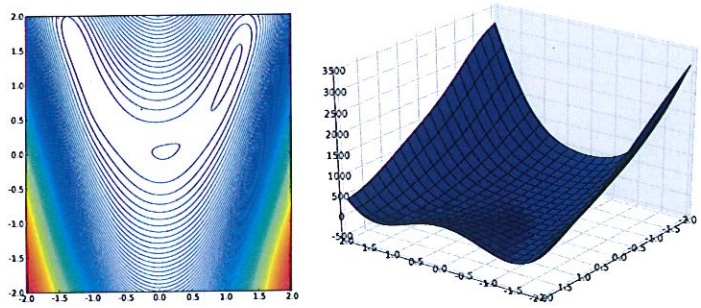


FIGURE 6.30 – Modèle obtenu pour le jeu d'hyperparamètres $\sigma = 0.3$, $\varepsilon = 0.001$ et $C = 9.04$, pour 80 données et pour la fonction (6.7).

Comparé au modèle précédent, l'ajout de nouvelles données d'apprentissage permet d'obtenir un modèle comportant 64 VS encore plus précis. En effet, les courbes de niveau respectent l'allure de celles de la vraie fonction de Rosenbrock (Figure 6.7).

Finalement, afin de vérifier la qualité des modèles construits par le modèle SVR, nous effectuons une validation externe et LOO. Les résultats obtenus pour un ensemble d'apprentissage de 30 et 40 données sont présentés respectivement dans les Tableaux 6.18 et 6.19.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
30 données						
RBF "tuné"	$9.983e-01$	$3.498e+01$	$1.299e+02$	$9.998e-01$	$1.209e+01$	$9.577e+01$
SVR	$9.850e-01$	$1.143e+02$	$4.408e+02$	$9.956e-01$	$6.613e+01$	$4.868e+02$

Tableau 6.18 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rosenbrock avec 30 données.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$9.998e-01$	$9.503e+00$	$4.051e+01$	$9.999e-01$	$4.366e+00$	$4.198e+01$
SVR	$9.927e-01$	$7.747e+01$	$3.349e+02$	$9.971e-01$	$5.328e+01$	$4.863e+02$

Tableau 6.19 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rosenbrock avec 40 données.

Nous pouvons constater que les coefficients de corrélation obtenus par le modèle SVR et par l'algorithme RBF "tuné" sont équivalents et très proches de 1. Bien que les erreurs RMSE et MAE soient un peu plus élevées nous pouvons quand même conclure que le modèle SVR construit un modèle quasiment aussi bon que l'algorithme RBF "tuné". De plus, sans le critère d'arrêt, nous pourrions certainement construire un modèle encore meilleur.

Rosenbrock 10

Pour le cas de la fonction Rosenbrock en 10 dimensions, nous analysons uniquement les statistiques (R, RMSE, MAE) de la validation externe et LOO. Les résultats obtenus pour un ensemble d'apprentissage de 40 et 100 données sont notés respectivement dans les Tableaux 6.20 et 6.21.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$4.672e-01$	$1.413e+03$	$3.760e+03$	$5.372e-01$	$1.512e+03$	$7.336e+03$
SVR	$4.274e-01$	$1.422e+03$	$3.826e+03$	$4.711e-01$	$1.584e+03$	$7.957e+03$

Tableau 6.20 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rosenbrock 10 avec 40 données.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
100 données						
RBF "tuné"	$6.462e-01$	$1.198e+03$	$3.542e+03$	$7.531e-01$	$1.155e+03$	$6.120e+03$
SVR	$5.816e-01$	$1.303e+03$	$3.856e+03$	$6.777e-01$	$1.288e+03$	$6.854e+03$

Tableau 6.21 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rosenbrock 10 avec 100 données.

Nous pouvons observer que les résultats obtenus pour l'ensemble d'apprentissage de 40 données ne sont pas bons (i.e., coefficients de corrélations faibles et les erreurs élevées) du fait du peu de données et ce pour les deux méthodes. Pour l'ensemble d'apprentissage de 100 données, les résultats sont un peu meilleurs bien que les performances du modèle SVR soient legèrement inférieures à celles de l'algorithme RBF "tuné".

Rosenbrock 20

Pour le cas de la fonction Rosenbrock en 20 dimensions, nous analysons uniquement les statistiques (R, RMSE, MAE) de la validation externe et LOO. Les résultats obtenus pour un ensemble d'apprentissage de 40 et 200 données sont inscrits respectivement dans les Tableaux 6.22 et 6.23.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
40 données						
RBF "tuné"	$4.518e-01$	$2.147e+03$	$7.247e+03$	$6.307e-01$	$1.782e+03$	$7.173e+03$
SVR	$4.748e-01$	$2.175e+03$	$7.788e+03$	$6.231e-01$	$1.837e+03$	$7.894e+03$

Tableau 6.22 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rosenbrock 20 avec 40 données.

Technique	Leave-one-out			Validation externe		
	R	RMSE	MAE	R	RMSE	MAE
200 données						
RBF "tuné"	$6.715e-01$	$1.611e+03$	$4.854e+03$	$6.855e-01$	$1.729e+03$	$7.401e+03$
SVR	$6.645e-01$	$1.756e+03$	$5.786e+03$	$6.222e-01$	$1.928e+03$	$1.010e+04$

Tableau 6.23 – Résultats des différents tests du modèle SVR et RBF "tuné" pour la fonction de Rosenbrock 20 avec 200 données.

Les conclusions sont identiques au cas de Rosenbrock à 10 dimensions.

6.4.2.1 En conclusion

Au vu des différents résultats obtenus, nous pouvons dire que la méthode SVR permet de construire des modèles assez robustes notamment lors de la présence de bruit. Cependant, la construction de modèles par l'algorithme SMO peut certainement être améliorée. Nous proposons au Chapitre "Conclusion et Perspectives" plusieurs modifications et améliorations de l'algorithme SMO constituant des perspectives de recherches futures intéressantes.

6.4.3 Etude des différents hyperparamètres de la méthode SVR

Suite à nos différents tests, nous résumons dans cette section les effets des différents hyperparamètres du modèle SVR sur la qualité du modèle et sur le temps de calcul. Le Tableau 6.24 résume les différentes constations (dans ce tableau, les symboles $\nearrow$ et $\searrow$ signifient respectivement que le paramètre augmente ou diminue).

Paramètre	Variation	Temps de calcul	Qualité du modèle - Remarques
ε	$\nearrow$	$\searrow$	le modèle est plus plat
	$\searrow$	$\nearrow$	le modèle est plus précis
C	$\nearrow$	$\nearrow$	le modèle est plus précis et la pénalisation des outliers est plus importante
	$\searrow$	$\searrow$	le modèle est moins précis et la pénalisation des outliers est moins importante
σ	$\nearrow$	$\nearrow$	le modèle est plus plat et il y a un risque de sous-apprentissage
	$\searrow$	$\searrow$	le modèle est plus précis et il y a un risque de sur-apprentissage
Dimension	$\nearrow$	$\searrow$	nous pouvons donc considérer un nombre plus important de données d'apprentissage
	$\searrow$	$\nearrow$	nous devons donc considérer un nombre moins important de données d'apprentissage
Données	$\nearrow$	$\nearrow$	le modèle est plus précis
	$\searrow$	$\searrow$	le modèle est moins précis
Bruit	$\nearrow$	$\nearrow$	il faut considérer un C important et éventuellement un ε un peu plus grand
	$\searrow$	$\searrow$	nous pouvons considérer des hyperparamètres C et ε plus faible

Tableau 6.24 – Etudes des hyperparamètres.

Ce tableau illustre donc bien le fait que la sélection des hyperparamètres est une tâche complexe et cruciale ayant des répercutions à la fois sur le temps de calcul (voir Section 6.6) et sur la précision du modèle. De plus, cette sélection dépend de la dimension du problème, de la taille de l'ensemble d'apprentissage et du bruit.

6.4.4 Etude du temps d'exécution de l'algorithme SMO

La durée du processus d'apprentissage est un critère très important pour Cenaero. La durée de construction d'un modèle par notre algorithme SMO, pour un jeu de données d'apprentissage quelconque, est de l'ordre de la minute [5]. Cependant, la recherche des hyperparamètres optimaux couplée à la procédure de validation croisée nécessitent la construction de nombreux modèles. Il est donc important d'analyser le temps de calcul pour la construction d'un modèle optimal (i.e., recherche des hyperparamètres optimaux). Pour cela, nous considérons la fonction **Rosenbrock** de dimensions 2, 5, 10, 15 et 20 et nous construisons respectivement, pour chaque dimension, un modèle optimal avec 40, 75, 100, 150, 200 et 300 données d'apprentissage. Nous mesurons alors le temps de calcul pour la construction de chaque modèle. Les différents résultats obtenus sont présentés à la Figure 6.31. La dimension de la fonction Rosenbrock est donnée en abscisse tandis que le temps est présenté en ordonnée, en secondes et selon une échelle logarithmique. Les différentes courbes correspondent aux ensembles d'apprentissage de différentes tailles utilisés.

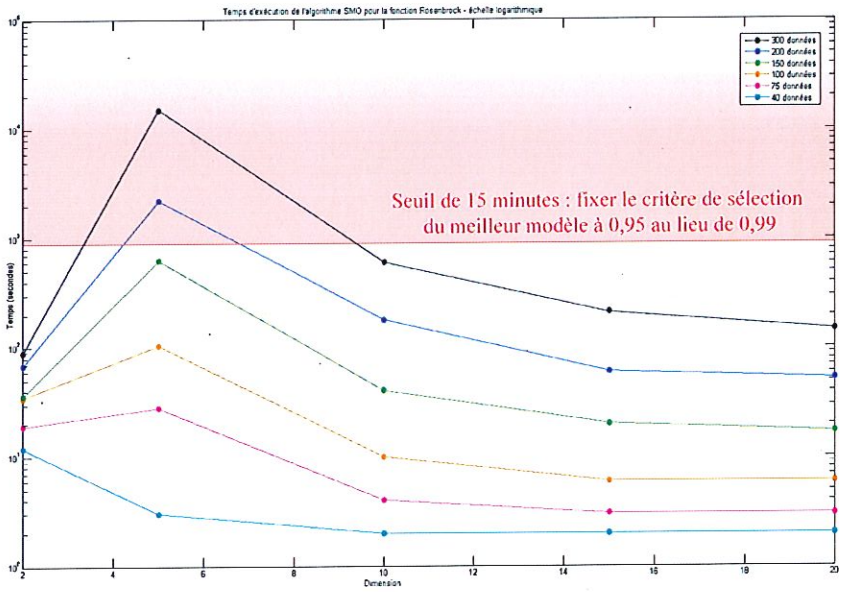


FIGURE 6.31 – Temps d'exécution de l'algorithme SMO pour la fonction Rosenbrock - Echelle logarithmique.

Le critère d'arrêt permettant de stopper l'apprentissage à savoir, si un modèle de coefficient de corrélation plus grand que 0.99 est obtenu, permet d'économiser un temps de calcul conséquent pour les problèmes de dimensions inférieures à 5. Pour les problèmes de dimensions supérieures à 5, il est très compliqué d'obtenir un coefficient de corrélation supérieur à 0.99. L'algorithme d'ap-

5. La durée variant selon les hyperparamètres, une étude de ces derniers et de leurs effets sur le temps de calcul et sur la précision du modèle est donnée à la Section 6.5.

prentissage doit donc tester tous les jeux d'hyperparamètres possibles, et comme nous l'avons vu à la Section 5.5, plus les paramètres C et σ augmentent, plus le temps de calcul est élevé ce qui explique les temps de calcul importants pour les modèles de Rosenbrock en 5 dimensions. Globalement, nous remarquons que les temps d'exécution diminuent fortement lorsque la dimension du problème augmente (e.g., les problèmes de dimensions supérieures à 10 demandent un temps d'exécution inférieur à 10 minutes). Cette constatation peut peut-être se justifier[6] par le fait qu'il est beaucoup plus simple de déterminer un hyperplan (i.e., un modèle) dans un espace d'entrées de dimension élevée et par le fait que SMO peut gérer des ensembles d'apprentissage importants. Cette propriété permet de considérer des ensembles d'apprentissage de plusieurs centaines de données pour des problèmes de dimensions importantes (e.g., la construction du modèle pour la fonction Rosenbrock en 20 dimensions avec 300 données d'apprentissage ne prend que 2 minutes et 26 secondes). Afin d'optimiser le temps de calcul, il serait intéressant d'essayer de fixer le critère d'arrêt en fonction de la dimension du problème.

Remarque

Des temps d'exécution identiques sont observés pour la fonction Rastrigin.

6.4.5 Test du modèle SVR sur la base de données LPT

Afin de tester l'efficacité du modèle SVR, nous avons testé celui-ci sur la base de données LPT (pour Low Pression Turbines). Ces données venant d'une étude confidentielle sur les turbines basse pression, nous ne pouvons expliquer la nature de ces dernières. Nous pouvons uniquement signaler que la base de données LPT contient 8 variables d'entrée. Nous avons évalué le coefficient de corrélation par la procédure du LOO pour les différents modèles obtenus pour deux ensembles de données.

- Le premier ensemble de données constitue le plan d'expériences de départ et contient 83 données générées par la technique LHS. Les Figures 6.32, 6.33 et 6.34 illustrent les résultats obtenus avec les différents modèles. En abscisse, nous avons les valeurs exactes issues de la simulation numérique et en ordonnée, les valeurs prédites par le modèle. Il en résulte que, plus les points rouges se situent sur la droite, meilleur est le modèle.

6. La justification est une question ouverte. Nous avons contacté Alexander Smola, Sathiya Keerthi et Chiranjib Bhattacharyya qui sont les principaux initiateurs de l'algorithme SMO afin d'obtenir des explications complémentaires sur le temps d'exécution. A ce jour, nous sommes toujours en attente d'une réponse.

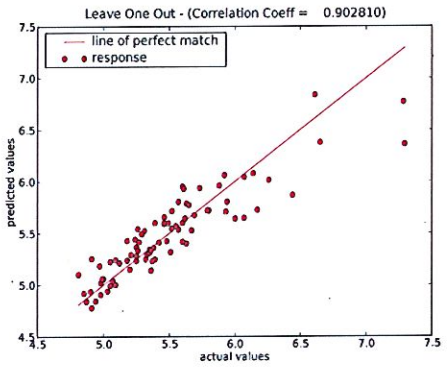


FIGURE 6.32 – Résultat de la procédure LOO pour le modèle RBF par défaut pour le premier ensemble de données.

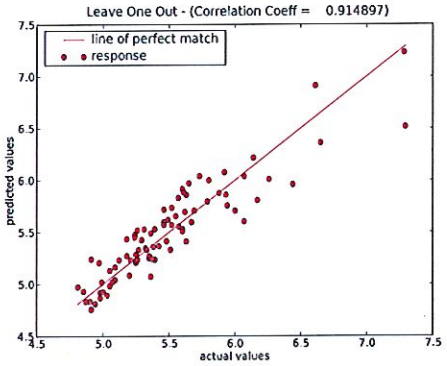


FIGURE 6.33 – Résultat de la procédure LOO pour le modèle RBF "tuné" pour le premier ensemble de données.

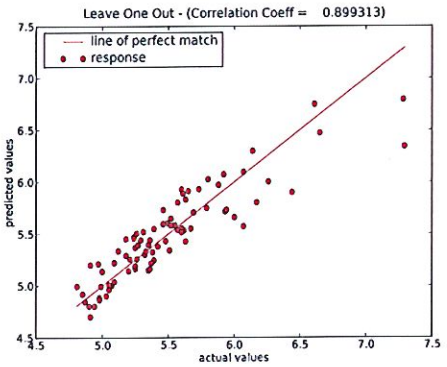


FIGURE 6.34 – Résultat de la procédure LOO pour le modèle SVR pour le premier ensemble de données.

Nous pouvons constater que les trois modèles calculés sont assez bons. Cela signifie donc que cette base de données est suffisante pour lancer une optimisation.

- Le deuxième ensemble de données est constitué du plan d'expériences contenant les 83 données auxquelles ont été ajoutés des points au cours de l'optimisation. En fin d'optimisation, la base de données contient 157 données (optimisation réalisée avec RBF par défaut) contenant 157 données[7]. Les Figures 6.35, 6.36 et 6.37 illustrent les résultats obtenus pour la construction des différents modèles sur cette base de données complétée.

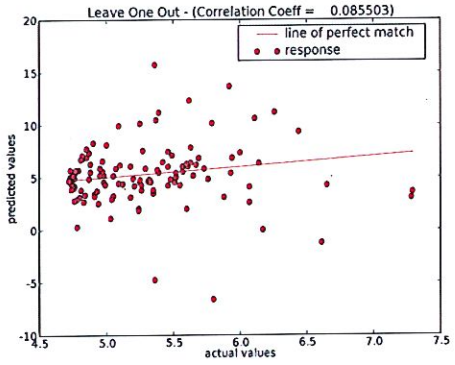


FIGURE 6.35 – Résultat de la procédure LOO pour le modèle RBF par défaut pour le deuxième ensemble de données.

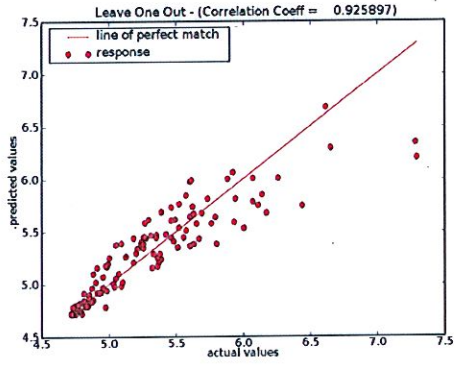


FIGURE 6.36 – Résultat de la procédure LOO pour le modèle RBF "tuné" pour le deuxième ensemble de données.

7. Comme expliqué au Chapitre 1, des données sont ajoutées au cours de l'optimisation, notamment dans la zone de l'optimum.

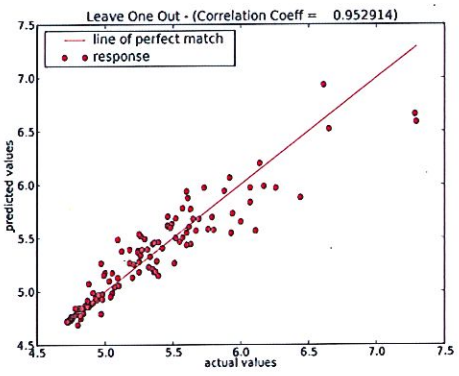


FIGURE 6.37 – Résultat de la procédure LOO pour le modèle SVR pour le deuxième ensemble de données.

Nous pouvons remarquer que l'approximation calculée par le modèle RBF par défaut est très mauvaise. Cela s'explique par le fait que les largeurs des fonctions de base radiales sont fixes pour ce modèle. L'approximation obtenue par le modèle RBF "tuné" est bon et légèrement améliorée par rapport au modèle RBF "tuné" sur le premier ensemble de données. L'approximation construite par le modèle SVR est encore meilleure. Le modèle SVR permet donc d'assimiler facilement les nouvelles données d'apprentissage ajoutées en cours d'optimisation et notamment les nouveaux points ajoutés dans le voisinage de l'optimum.

6.4.6 Couplage avec l'optimisation

Le principe de l'optimisation assistée par modèle est de construire un modèle du problème d'origine. Ensuite, ce modèle peut être exploité dans une boucle d'optimisation en lieu et place du modèle numérique original. En effet, le temps pour effectuer une seule simulation exacte peut prendre plusieurs heures. Le modèle peut être rapidement évalué et, par conséquent, le nombre d'évaluations du modèle par l'algorithme d'optimisation ne constitue plus un problème critique.

1. En premier lieu, nous illustrons l'efficacité de l'optimisation assistée par modèle. La Figure 6.38 illustre l'historique de convergence de l'optimisation pour la fonction Ackley en 5 dimensions. Les itérations sont données en abscisse tandis que le logarithme [8] de la valeur de la fonction objectif est donné en ordonnée.

8. Le logarithme de la fonction objectif est utilisé afin de bien mettre en valeur la différence entre l'algorithme génétique avec et sans modèle.

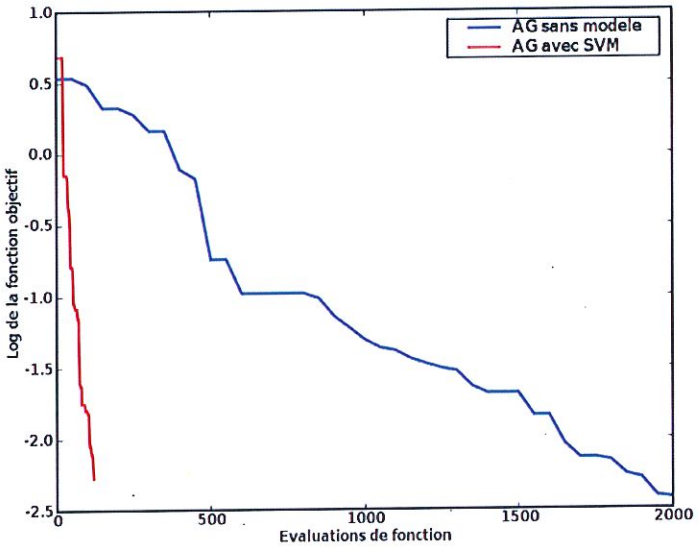


FIGURE 6.38 – Historique de convergence pour la fonction Ackley pour l'algorithme génétique avec et sans modèle.

Dans un premier temps, nous avons effectué une optimisation de la fonction Ackley en 5 dimensions avec l'algorithme génétique "pur" (i.e., sans modèle), avec une population de 50 individus et 40 générations ce qui correspond donc à 2000 évaluations de fonctions. Ce résultat est comparé avec celui obtenu avec l'algorithme génétique (AG) assisté par un modèle SVR. La base de données initiale comporte 20 points générés par la technique de plan d'expériences LCVT. Pour l'algorithme génétique assisté, nous avons effectué seulement 100 itérations (i.e., au total 120 itérations de fonction). La Figure 6.38 indique clairement que pour un nombre fixé d'évalutations de la vraie fonction objectif, l'approche combinant l'algorithme génétique avec un modèle approché est plus efficace qu'un algorithme génétique "pur" utilisant uniquement la vraie évaluation de la fonction objectif.

2. Deuxièmement, nous avons testé l'optimisation pour différents modèles. Les graphes présentés ci-après sont une moyenne de 50 exécutions (i.e., 50 bases de données différentes).

   - **Rastrigin**

     La Figure 6.39 illustre l'historique de convergence pour la fonction Rastrigin pour 40 itérations (i.e., 1 itération est équivalente à 1 optimisation du modèle SVR par l'algorithme génétique). La base de données initiale comporte 20 données générées par la technique LCVT.

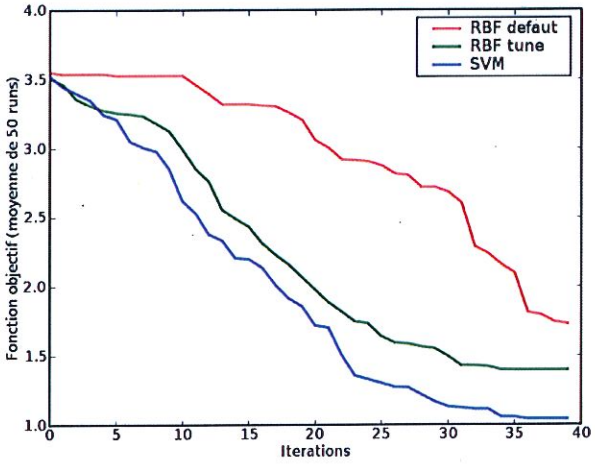


FIGURE 6.39 – Historique de convergence pour la fonction Rastrigin.

La valeur optimale de Rastrigin à l'optimum $(0, 0)$ est de 0. Nous remarquons que le modèle SVR permet une convergence plus précise vers la valeur optimale de Rastrigin.

- **Ackley**

La Figure 6.40 illustre l'historique de convergence pour la fonction Ackley pour 100 itérations (i.e., 1 itération est équivalente à 1 optimisation du modèle SVR par l'algorithme génétique). La base de données initiale comporte 20 données générées par la technique LCVT.

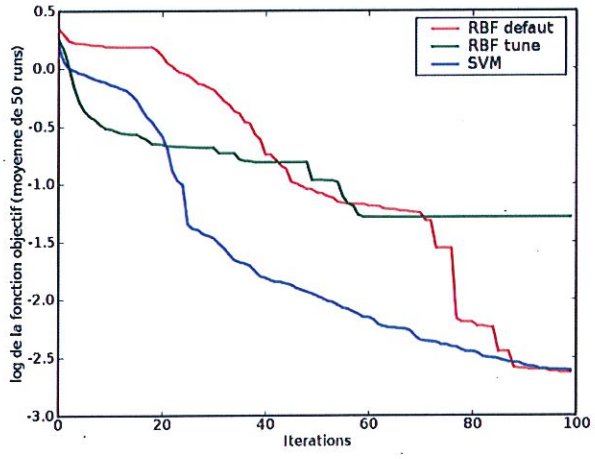


FIGURE 6.40 – Historique de convergence pour la fonction Ackley.

La valeur optimale de Ackley à l'optimum $(0, 0)$ est de 0. Nous remarquons encore une fois que le modèle SVR permet une convergence plus précise tendant vers la valeur optimale de Ackley.

- **Branin**

La Figure 6.41 illustre l'historique de convergence pour la fonction Rastrigin pour 25 itérations (i.e., 1 itération est équivalente à 1 optimisation du modèle SVR par l'algorithme génétique). La base de données initiale comporte 12 données générées par la technique LCVT.

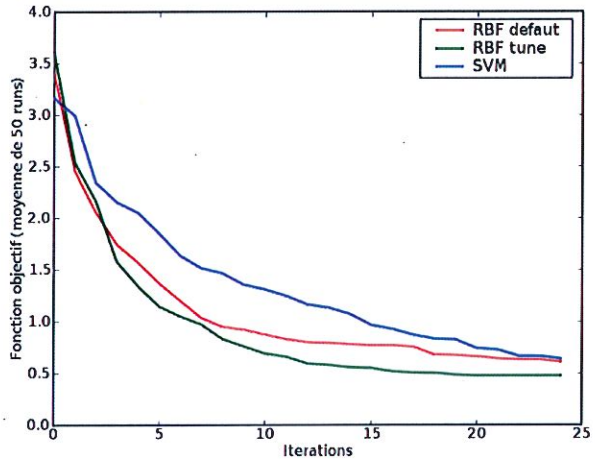


FIGURE 6.41 – Historique de convergence pour la fonction Branin.

La valeur optimale de Branin aux optima est de 0.397887. Dans ce cas, le modèle SVR est un peu moins bon que les modèles RBF par défaut et RBF "tuné" et ne permet donc pas une convergence plus précise vers l'optimum de Branin.

Remarque générale pour Rastrigin, Ackley et Branin

L'algorithme génétique assisté par le modèle SVR converge vers un optimum du modèle des fonctions. Cet optimum ne correspond pas forcément à l'optimum de la vraie fonction. Afin de converger plus précisément vers l'optimum de la vrai fonction, il faut considérer des bases de données comportant un nombre de données d'apprentissage plus important.

Nous pouvons conclure que le modèle SVR couplé à l'algorithme d'optimisation ne se comporte pas trop mal par rapport aux autres modèles, mais de toute façon, nous ne pourrons jamais avoir un seul et unique modèle qui soit le meilleur pour tous les types de fonction.

6.4.7 Avantages du modèle SVR par rapport au modèle de réseaux de neurones

Pour conclure cette section, nous rappelons les différents avantages de la méthode SVR. Bien que les méthodes d'apprentissage restent différentes, il existe une certaine équivalence entre les SVM et les réseaux de neurones [22]. Cependant, le modèle SVR apporte encore plusieurs avantages par rapport au modèle par réseaux de neurones à savoir principalement :

1. La sélection des centres γ_k (voir Section 2.5.2.2, Tableau 2.1) est automatiquement réalisée par l'algorithme.

2. L'apprentissage correspond à un problème d'optimisation convexe (voir Section 3.6).

3. Le modèle obtenu est plat (smooth en anglais).

4. Le problème de la malédiction des données est évité à l'aide des noyaux [5, 8].

5. Une bonne capacité de généralisation peut être obtenue même à partir d'un nombre de données d'apprentissage peu élevé.

6. De nombreuses améliorations restent possibles (voir Chapitre "Conclusion et Perspectives").

Conclusion et perspectives

Ce mémoire de fin d'études propose une investigation d'une méthode d'apprentissage, à savoir les machines à vecteurs de support (SVM), pour la classification, et plus particulièrement pour la régression. Pour commencer, nous avons étudié le cas de la classification par les SVM afin de se familiariser avec les différents concepts importants. Nous nous sommes ensuite intéressés à la régression.

Cette investigation résulte d'une demande formulée par Cenaero. Ce centre d'excellence en technologies de simulations numériques comporte quatre groupes dont le groupe Optimisation et Méthodes Numériques qui a encadré ce mémoire. Ce groupe se concentre principalement sur le développement et l'utilisation de la plateforme d'optimisation multi-disciplinaire Minamo. Les simulations numériques des systèmes à optimiser étant généralement très coûteuses en temps de calcul, le processus d'optimisation par algorithmes génétiques permettant de résoudre des problèmes mono- et multi-objectif(s) complexes est accéléré grâce à l'utilisation de méta-modèles (i.e., des approximations peu coûteuses des fonctions à optimiser) sur lesquels l'optimisation est effectuée. Ces itérations sont donc utilisées pour choisir itérativement les nouvelles évaluations à réaliser.

Le logiciel Minamo mis au point par Cenaero permet d'effectuer l'interpolation de fonctions inconnues à l'aide de réseaux RBF ou de la méthode Kriging. L'algorithme d'apprentissage RBF "tuné" implémenté dans Minamo donne d'excellents résultats. Néanmoins, cet algorithme tel qu'il est implémenté pour le moment, présente quelques inconvénients notamment lors de la présence de bruit qui peut entacher les données d'apprentissage et impliquer dans certains cas une capacité de généralisation médiocre du modèle construit. L'objectif de ce mémoire est de proposer un nouveau type de modèle afin d'élargir l'éventail des modèles disponibles dans Minamo.

L'implémentation de la nouvelle méthode SVR d'approximation a été réalisée à l'aide de l'algorithme SMO et implantée dans Minamo. La méthode SVR a ensuite été testée sur un ensemble de fonctions tests. Ces fonctions tests sont des fonctions analytiques connues permettant de générer rapidement, par des techniques de plan d'expériences, des ensembles d'apprentissage. Les différents résultats ont montré que les modèles SVR obtenus sont en général aussi bons que ceux construits par la méthode RBF "tuné". La capacité de généralisation des modèles, qui a été mesurée à l'aide de la validation croisée mais aussi par validation externe et LOO, s'est également avérée bonne et équivalente à celle des modèles RBF "tuné". Les performances du modèle SVR, face à des données d'apprentissage bruitées, ont aussi pu être testées. Pour la majorité des fonctions testées, les erreurs mesurées (RMSE et MAE), par rapport aux données cibles sont équivalentes et parfois meilleures qu'avec la méthode RBF "tuné". En outre, le modèle SVR a pu être testé et validé, pour un cas test industriel concernant

les turbines basse pression. Les résultats ont montré que le modèle SVR permet de construire correctement un modèle à partir d'un ensemble d'apprentissage comprenant des points ajoutés en cours d'optimisation.

Le modèle SVR d'approximation de fonctions est englobé dans le logiciel Minamo, utilisant des algorithmes génétiques fortement accélérés grâce à l'utilisation de modèles, il a donc pu également être testé afin de connaître son impact sur la recherche de l'optimum d'une fonction. Les résultats ont montré que l'optimum est atteint plus rapidement avec la méthode SVR et est plus proche de l'optimum théorique que celui atteint par simulation exacte, tout comme pour les autres modèles RBF. Ensuite, le couplage entre l'algorithme génétique et le modèle SVR a ensuite été comparé au couplage entre l'algorithme génétique et les modèles RBF par défaut et RBF "tuné". Nous avons constaté que le modèle SVR couplé à l'algorithme génétique est parfois meilleur, équivalent ou moins bon par rapport aux autres modèles, mais de toute façon, nous ne pourrons jamais avoir un seul et unique modèle qui soit le meilleur pour tous les types de fonctions. Comme nous avons élargi l'éventail des modèles possibles, nous pourrions imaginer qu'en fonction du problème, le type de modèle le plus adapté serait choisi.

Finalement, la plus grande difficulté de la méthode SVR est de déterminer les hyperparamètres optimaux. En effet, la qualité des modèles dépend fortement de ces hyperparamètres. Nous avons pour cela implémenté une procédure de sélection des hyperparamètres basée sur la validation croisée.

Pour conclure, nous rappelons les différentes perspectives de recherches futures. Il serait intéressant d'améliorer la recherche des hyperparamètres en incorporant l'hyperparamètre ε dans la procédure de sélection des hyperparamètres. Ceci impliquerait une sorte de lissage lors de données bruitées. Par la suite, une fois les hyperparamètres optimaux obtenus par notre procédure de sélection, il serait aussi judicieux de tenter de les affiner en relançant celle-ci dans la région des meilleurs hyperparamètres trouvés. Finalement, il serait aussi intéressant d'implémenter une autre technique de recherche des hyperparamètres basée sur des méthodes évolutionnaires [26] ou sur une recherche par descente de gradient [13, 4]. Une autre façon d'améliorer le choix du paramètre ε serait d'analyser et implémenter le problème d'optimisation ν-SVR [62], permettant d'adapter automatiquement l'hyperparamètre ε. Nous pourrions également tester d'autres noyaux tel que le noyau spline ou B-spline [30] donnant des résultats prometteurs. Les modèles pourraient aussi être améliorés par l'incorporation de connaissances a priori dans l'apprentissage SVM pour la régression, comme par exemple, des valeurs particulières connues en certains points ou des bornes sur les dérivées de la fonction inconnue. Ces connaissances peuvent être incluses par simple ajout de contraintes linéaires en les paramètres du problème d'optimisation linéaire correspondant à l'apprentissage SVR. L'apprentissage incorporant les informations a priori reste donc un problème d'optimisation convexe [43, 44, 50]. Finalement, la dernière amélioration concernerait la résolution du problème quadratique de façon différente. Nous pourrions pour cela étudier et implémenter l'algorithme proposé dans l'article "Improvements to SMO Algorithm for SVM Regression" [65] ou encore implémenter un algorithme de type points intérieurs [29]. Enfin, d'autres méthodes permettant de quantifier la capacité de généralisation du modèle pourraient être étudiées.

Les possibilités d'améliorations ne manquent donc pas et ce serait dommage de s'arrêter en si bon chemin. Le mien s'arrête ici et je passe le témoin à Cenaero.

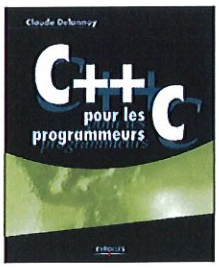
Annexe A

Codes sources

Etant donné que les activités de l'entreprise Cenaero en matière de recherches, développement et production de logiciels, algorithmes, méthodes, processus, etc. sont de nature confidentielles, les codes sources ne sont pas publiés dans ce mémoire. Le lecteur intéressé par le code source peut prendre contact avec Madame le Docteur Caroline Sainvitu - caroline.sainvitu@cenaero.be. Néanmoins, les codes sources seront disponibles à la lecture le jour de la défense de ce mémoire.

Remarque sur l'implémentation

Les différents codes sources ont été implémentés en cpp. Afin de maîtriser ce nouveau langage, nous avons utilisé le livre "C++ pour les programmeurs C" [17].



Afin de visualiser les résultats obtenus avec l'algorithme SMO, différents scripts python ont été mis au point pour les fonctions à une et deux dimensions respectivement.

Finalement, les différentes figures de la Section 6.1 ont été réalisées avec Matlab.

Bibliographie

[1] *Introduction Aux Machines à Vecteurs de Support (SVM)*, Alctra R&D, http ://www.alctra.fr/intro_SVM.pdf.

[2] F. Abdallah, *Les séparateurs à vaste marge (SVM)*, HEUDIASYC, Université de Technologie de Compiègne, Centre de recherche Royallieu, BP 20529.

[3] D. Andre, *Exploring Online Support Vectro Regression*, Computer Sciences Division, University of California, Berkeley, CA 94720-1776, 1999.

[4] N. E. Ayat, M. Cheriet, C. Y. Suen., *Optimizing of the SVM kernels using an empirical error minimization scheme*, Lecture Notes in Computer Science, 2002, Volume 2388/2002, pages 257-271, 2002.

[5] R. E. Bellman, *Adaptive Control Processes*, Princeton University Press, 1990.

[6] C. Bernard, *Ondelettes et problèmes mal posés : la mesure du flot optique et l'interpolation irrégulière - Apprentissage et interpolation*, Académie de Montpellier, École Polytechnique, Centre de mathématiques appliquées, 1999.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[8] Bishop C., *Neural Networks for Pattern Recognition*, Oxford University Press, New York, 1995.

[9] L. Bottou, O. Chapelle, D. DeCoste, J. Weston, *Large Scale Kernel Machines*, MIT Press, 2007.

[10] M. D. Buhmann, *Radial Basis Functions : Theory and Implementations*, Cambridge University Press, 2003.

[11] C. J.C. Burges, *A tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery, Volume 2, Number 2, pages 121-167, 1998

[12] J. Callut, *Implémentation efficace des Support Vector Machines pour la classification*, Master's Thesis in Computer Science, 2003.

[13] O. Chapelle, V. N. Vapnik, O. Bousquet, S. Mukherjee, *Choosing multiple parameters for support vector machines*, AT&T Research Labs, 2002.

[14] S. M. Clarke, J. H. Griebsch, T. W. Simpson, *Analysis of Support Vector Regression for Approximation of Complex Engineering Analyses*, Journal of mechanical design, Volume 127, Number 6, pages 1077-1087, 2005.

[15] A. Cornuéjols, L.Miclet, *SVM : Support Vector Machines - Séparateurs à Vaste Marge*, IRISA, 2002.

[16] N. Cristianini, J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, 2000.

[17] C. Delannoy, *C++ pour les programmeurs C*, Paris Eyrolles, 6ème édition, 2007.

[18] N.R. Draper, H. Smith, *Applied Regression Analysis*, Wiley Series in Probability and Statistics, 1998.

[19] Drucker H., Burges C., Kaufman L., Smola A., Vapnik V., *Support Vector Regression Machines, Neural Information Processing Systems*, MIT Press, 1997.

[20] P. Dupont, *Support Vector Machines*, International Joint Conference on Neural Networks, pages 1443-1448, 2005.

[21] P. Erästö, *Support Vector Machines - Backgrounds and Practice*, Academic Dissertation for the Degree of Licentiate of Philosophy, Rolf Nevanlinna Institute, 2001.

[22] T. Evgeniou, M. Pontil, T. Poggio, *Regularization networks and support vector machines*, Advances in Computational Mathematics, Volume 13, pages 1-50, 200.

[23] M. Fauvel, *Machines à Support Vecteurs : Application à la Classification d'Images Hyperspectrales*, gipsa-lab, Grenoble, 2007.

[24] A.I.J. Forrester, A.J. Keane, *Recent advances in surrogate-based optimization*, Progress in Aerospace Sciences, Volume 45, Issues 1-3, Pages 50-79, 2009.

[25] A.I.J. Forrester, A. Sóbester, A.J. Keane, *Engineering Design via Surrogate Modelling : A Practical Guide*, John Wiley & Sons, Chichester, 240 pages, 2008.

[26] F. Friedrichs, C. Igel, *Evolutionary tuning of multiple SVM parameters*, In Proc. of the 12th Europ. Symp. on Artificial Neural Networks (ESANN), 2004.

[27] G. Gasso, K. Zapien, S. Canu, *Chemins de régularisation pour la régression ν-SVR*, GRETSI, 2007.

[28] N. Gilardi, A. Gammerman, M. Kanevski, M. Maignan, T. Melluish, C. Saunders, V. Vovk, *Application des méthodes d'apprentissage pour l'étude des risques de pollution dans le Lac Léman*, Université de Genève, 2000.

[29] M. D. Gonzalez-Lima, W. W. Hager, H. Zhang, *An affine-scaling interior-point method for continuous knapsack constraints*, AMS, subject classifications, 2005.

[30] S. R. Gunn, *Support Vector Machines for Classification and Regression*, University of Southampton, Faculty of Engineering, Science and Mathematics - School of Electronics and Computer Science, Technical Report, 1998.

[31] M. Hasan, F. Boris, *SVM : Machines à Vecteurs de Support ou Séparateurs à Vastes Marges*, BD Web, ISTY3, Versailles St Quentin,2006.

[32] R. L. Haupt , S. E. Haupt, *Practical Genetic Algorithms*, Wiley-Interscience, 2004.

[33] Te-M. Huang, V. Kecman, I. Kopriva, *Kernel Based Algorithms for Mining Huge Data Sets : Supervised, Semi-supervised, And Unsupervised Learning*, 2006.

[34] V. Iliopoulou, I. Lepot, P. Geuzaine, *Design optimization of a HP compressor blade and its hub endwall*, ASME Paper GT2008-50293, Berlin, 2006.

[35] V. Iliopoulou, T. Mengistu, I. Lepot, *Non Axisymmetric Endwall Optimization Applied to a High Pressure Compressor Rotor Blade*, AIAA-2008-5881, 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization, Conference, Victoria, British Columbia, Canada, 2008.

[36] T. Joachims, *Making Large-Scale SVM Learning Practical*, publié dans "Advances in Kernel Methods - Support Vector Learning", MIT Press, Cambridge, USA, 1999.

[37] L. Kaufman, *Solving the quadratic programming problem arising in support vector classification*, MIT Press, 1999.

[38] A. Keane, P. Nair, *Computational Approaches for Aerospace Design*, Wiley, 2005.

[39] M. Hue, *Méthodes à noyaux et Machines à Vecteurs de Support (SVMs)*, Cours Master Isifar, 2007.

[40] J. Kharroubi, *Etude de Techniques de Classement "Machines à Vecteurs Supports" pour la Vérification Automatique du locuteur*, ENST, Doctorat Signal et Image, Signal et Images, 2000.

[41] F. Lauer, G. Bloch, *Méthodes SVM pour l'identification*, Centre de Recherche en Automatique de Nancy (CRAN UMR 7039), Université de Nancy, CNRS, 2006.

[42] F. Lauer, *Machines à Vecteurs de Support et Identification de Systèmes Hybrides*, Ph. D., Université Henri Poincaré, Centre de Recherche en Automatique, Nancy, UMR CNRS 7039, 2008.

[43] F. Lauer, G. Bloch, *Incorporating prior knowledge in support vector regression*, Machine Learning, 70(1), pages 89-118, 2008.

[44] M. Lazaro, I. Santamaria, F. Pérez-Cruz, A. Artés-Rodriguez, *Support vector regression for simultaneous learning of a multivariate function ond its derivatives*, Neurocomputing,, 69, pages 42-61, 2005.

[45] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86(11), pages 2278-2324, 1998.

[46] F. Liese, K-J. Miescke, *Statistical Decision Theory : Estimation, Testing, and Selection*, Springer Series in Statistics, 2008.

[47] L. Lukas, *Least Squares Support Vector Machines Classification Applied to Brain Tumour Recognition Using Magnetic Resonance Spectroscopy*, Ph. D., Katholieke Universiteit Leuven, Faculteit Toegepaste Wetenschappen Departement Elektrotechniek, 2003.

[48] P. Mahé, *Noyaux pour graphes et Support Vector Machines pour le criblage virtuel de molécules*, Rapport de stage, INRIA, centre de recherche Sophia Antipolis, 2003.

[49] G. Mak, *The implementation of support vector machines using the sequential minimal optimization algorithm*, Ph. D., School of Computer Science McGill University, Montreal, 2000.

[50] O. L. Mangasarian, E. W. Wild, *Nonlinear knowledge in kernel approximation*, Technical Report 05-05, Data Mining Institute, University of Wisconsin, 2005.

[51] J. Nocedal, S. J. Wright, *Numerical Optimization*, Springer, 2006.

[52] T. Norikazu, J. Guo, N.i Tetsuo, *Global Convergence of SMO Algorithm for Support Vector Regression*, IEEE transactions on neural networks, Volume 19, Number 6, pages 971-982, 2008.

[53] E. Osuna, R. Freund, F. Girosi, *An Improved Training Algorithm for Support Vector Machines*, Neural Networks for Signal Processing VII. Proceedings of the IEEE Workshop, 1997.

[54] D. Pfanzagl, *Parametric Statistical Theory*, Walter de Gruyter & Co, 1994.

[55] J. C. Platt, *Fast training of support vector machines using sequential minimal optimization*, MIT Press, 1999.

[56] J. C. Platt, *Using Analytic QP and Sparseness to Speed Training of Support Vector Machines*, Advances in Neural Information Processing Systems 11, MIT Press, 1999.

[57] S. Rippa, *An algorithm for selecting a good value for the parameter c in radial basis function interpolation*, Advances in Computational Mathematics, Volume 11, Number 2-3, 1999.

[58] H. Robaye, *Investigation et amélioration de méthodes d'approximations par réseaux de fonctions à base radiale généralisées*, Université Libre de Bruxelles, Mémoire d'Ingénieur civil en Electromécanique, 2006.

[59] D. Ross, T. Most, J. Unger, J. Will, *Advanced surrogate models within the robustness evaluation*, Neural Networks for Signal Processing VII. Proceedings of the IEEE Workshop, 1997.

[60] C. Sainvitu, V. Iliopoulou, I. Lepot, *Global Optimization with Expensive Functions - Sample Turbomachinery Design Application*, BFG Springer book, Volume "Recent Advances in Optimization and its Applications in Engineering", 2010.

[61] Y. Saka, M. Gunzburger, J. Burkardt, *Latinized, improved LHS, and CVT point sets in hypercubes*, International Journal of Numerical Analysis and Modeling, Volume 4, Number 3-4, Pages 729-743, 2007.

[62] A. J. Smola, B. Schölkopf, *Learning with kernels*, MIT Press, 2002.

[63] A. J. Smola, B. Schölkopf, *A Tutorial on Support Vector Regression*, NeuroCOLT Technical Report TR-98-030, 2003.

[64] A. J. Smola, B. Schölkopf, K. R. Muller, *The connection Between Regularization Operators and Support Vector Kernels*, Neural Networks, 1998.

[65] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K Murthy, *Improvements to SMO Algorithm for SVM Regression.*

[66] V. N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, Second Edition, 2000.

[67] E. Viennet, *Reconnaissance des formes et Machines à Vecteurs de Support*, University of Singapore, Dept of Mechanical and Production Engineering, 1998.

[68] *Cenaero - Annual Report 2008*, http ://cms.horus.be/files/99936/MediaArchive/pdf/CENAERO-AR2008.pdf.

[69] *http ://www.dice.ucl.ac.be/ verleyse/lectures/gbio2020/slides/classification v1-0 bw 2spp.pdf.*

[70] *http ://www.kernel-machines.org.*