From Support Vector Machines to Hybrid System Identification

Fabien Lauer

Nancy-Université
Université
Henri Poincaré

Ecole doctorale IAEM Lorraine
Département de formation doctorale en automatique
UFR ESSTIN

Machines à Vecteurs de Support et Identification de Systèmes Hybrides

—

From Support Vector Machines to Hybrid System Identification

THÈSE

présentée et soutenue publiquement le 1er octobre 2008
pour l'obtention du

Doctorat de l'Université Henri Poincaré – Nancy 1

spécialité automatique

par

Fabien Lauer

Membres du Jury

Stéphane Canu	Prof., Institut National des Sciences Appliquées de Rouen, France (*rapporteur / reviewer*)
Joos Vandewalle	Prof., Katholieke Universiteit Leuven, Belgique (*rapporteur / reviewer*)
Gérard Dreyfus	Prof., Ecole Supérieure de Physique et de Chimie Industrielles, France
Alain Richard	Prof., Université Henri Poincaré – Nancy 1, France
Christoph Schnörr	Prof., Heidelberg University, Allemagne
René Vidal	Ass. Prof., Johns Hopkins University, Etats-Unis
Gérard Bloch	Prof., Université Henri Poincaré – Nancy 1, France (*directeur de thèse / advisor)*

CRAN

Centre de Recherche en Automatique de Nancy, UMR CNRS 7039, Nancy–Université

Cette thèse est co-financée par le Centre National de la Recherche Scientifique (CNRS) et la Région Lorraine dans le cadre d'une Bourse de Doctorat pour Ingénieurs (BDI), pour la période d'octobre 2005 à septembre 2008.

This thesis is co-sponsored by the French National Center for Scientific Research (CNRS) and the Région Lorraine for the time period from October 2005 to September 2008.

Remerciements

Je tiens à remercier en premier lieu mon directeur de thèse, Gérard Bloch, sans qui je n'en serais certainement pas là. Je lui dois mon introduction à la recherche scientifique, un certain goût pour l'écriture et une rigueur à toute épreuve.

Je suis reconnaissant envers M. Stéphane Canu, directeur du Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes (LITIS), et M. Joos Vandewalle, directeur de la division SCD de la Katholieke Universiteit Leuven, d'avoir accepté d'être les rapporteurs de cette thèse. Je remercie également M. Gérard Dreyfus, directeur du Laboratoire d'Electronique de l'ESPCI, M. Christoph Schnörr, professeur à l'Université de Heidelberg, M. René Vidal, directeur du Vision, Dynamics and Learning Lab at Johns Hopkins University, et M. Alain Richard, directeur du Centre de Recherche en Automatique de Nancy (CRAN), d'avoir accepté de faire partie du jury. Je remercie notamment l'ensemble des membres du jury pour leurs commentaires constructifs et les discussions qui ont suivi la soutenance de thèse.

Je remercie les membres du CRAN qui m'ont permis de travailler dans un cadre scientifique encourageant, et notamment Marion Gilson-Bagrel, Hugues Garnier et Jamal Daafouz pour m'avoir invité à présenter mes travaux dans divers groupes de travail et colloques nationaux.

Je remercie plus généralement l'ensemble des personnels de l'ESSTIN et en particulier du CRAN pour leur bonne humeur. J'ajouterais une attention particulière pour Sylvie Ferrari qui était toujours disponible pour m'aider à gérer mes déplacements et autres formalités administratives.

Merci aussi à tous les personnels du département GEII de l'IUT Nancy–Brabois, où j'ai effectué mon monitorat, pour avoir fait de ces trois années une expérience positive.

Un grand merci à ma famille et en particulier à mes parents qui m'ont toujours soutenu dans mes choix.

Le mot de la fin sera pour Caroline, ma fiancée. Merci à toi d'être et d'avoir toujours été là pour moi.

Résumé

Contexte

Automatique et modélisation

En automatique, l'obtention d'un modèle peut être vue comme le goulot d'étranglement de l'ensemble de la procédure de contrôle. Quand les connaissances physiques sur le processus ne suffisent pas ou quand les lois physiques sont trop complexes pour permettre la synthèse ou l'implémentation temps-réel, les modèles sont souvent construits à partir de données expérimentales. Ce problème, sur lequel la thèse se concentre, peut être vu de manière plus générale comme la détermination d'un modèle d'un système sur la base d'observations de son comportement entrée–sortie. Formulé ainsi, le problème est transposable à de nombreux champs de recherche et a été étudié par diverses communautés selon le type de système à modéliser, la nature des entrées–sorties, et le modèle considéré. La nature des sorties permet de définir deux grand types de problèmes : la classification, pour laquelle les sorties ne prennent que des valeurs discrètes, et la modélisation, pour laquelle les sorties sont habituellement des nombres réels. La modélisation peut ensuite se diviser en régression d'une part, et identification de systèmes dynamiques d'autre part.

Apprentissage statistique

Parmi toutes les approches existantes pour traiter ces problèmes, l'apprentissage statistique possède des caractéristiques intéressantes, comme de solides fondements théoriques, un cadre de travail unifié pour la classification et la régression, et des algorithmes efficaces en pratique. En particulier, les Machines à Vecteurs de Support (*Support Vector Machines*, SVMs) cristallisent tous ces avantages et ont été la source d'une recherche intensive au cours des quinze dernières années. Reconnaissant les performances de premier rang obtenues avec les SVMs sur un grand nombre d'applications, la thèse place ces machines d'apprentissage au cœur de l'étude. Le but est de promouvoir une approche interdisciplinaire, dans laquelle des méthodes et des outils développés par la communauté de chercheurs en apprentissage statistique peuvent aider à résoudre des problèmes typiquement étudiés par la communauté des automaticiens.

Optimisation

Dans cette thèse, les résultats sont souvent présentés sous la forme de problèmes d'optimisation, pour lesquels des solutions peuvent être trouvées grâce à des optimiseurs adéquats, plutôt que sous la forme de théorèmes. Ici aussi, l'idée est de pouvoir bénéficier d'avancées dans d'autres champs de recherche. Une tendance similaire peut être observée dans d'autres domaines de l'automatique, où des problèmes de contrôle conduisent souvent à des inégalités matricielles linéaires (*Linear Matrix Inequalities*, LMI). La différence principale est qu'une LMI constitue un problème de faisabilité, alors que cette thèse considère des problèmes de minimisation.

Les nombreux avantages de l'optimisation convexe, devenue maintenant une technologie « presse-bouton », peuvent expliquer ce phénomène. Des problèmes d'optimisation qui étaient considérés comme des impasses gagnent maintenant le statut de résultats. Cependant, l'optimisation non-convexe ne doit pas être considérée comme impossible, mais simplement comme plus difficile. La majeure partie de la thèse traite d'optimisation convexe avant de s'attaquer dans le dernier chapitre à un problème non-convexe : l'identification de systèmes hybrides.

Approche boîte grise

Se concentrant sur des systèmes non-linéaires, la thèse présente principalement l'utilisation de modèles non-paramétriques de nature *boîte noire*. Ces modèles boîte noire sont déterminés par apprentissage à partir de données entrées–sorties. Cette approche est à l'opposé de la modélisation *boîte blanche*, basée sur des lois physiques et pour laquelle les paramètres ont souvent une signification réelle. La thèse promeut l'usage d'une troisième approche, la modélisation *boîte grise*. Dans cette approche, les algorithmes d'apprentissage sont modifiés pour prendre en compte des connaissances *a priori* sur le problème. Ceci permet de bénéficier des données expérimentales et des capacités des machines d'apprentissage ainsi que d'informations supplémentaires souvent disponibles en pratique. Des exemples de connaissances *a priori* rencontrées dans différents contextes sont presentés ci-dessous tout en introduisant les trois grands problèmes étudiés dans la thèse.

Classification. Le but de la classification est de trouver un modèle capable d'assigner un objet à une classe, c'est-à-dire de reconnaître l'objet représenté par un ensemble de caractéristiques. Dans ce cadre, les sorties du modèle, ici le classifieur, ne prennent que des valeurs discrètes. Un exemple typique d'application est la reconnaissance de caractères manuscrits, dans lequel le modèle doit pouvoir donner en sortie le caractère représenté par l'image d'entrée. En reconnaissance de formes, l'invariance de classe est la forme de connaissances *a priori* le plus souvent rencontrée. L'invariance de classe signifie que la sortie du classifieur doit restée inchangée si une transformation particulière est appliquée à la forme en entrée. Par exemple, l'invariance aux translations et rotations de l'image est souvent considérée en reconnaissance de caractères manuscrits.

Régression. Le but de la régression est de trouver un modèle capable de prédire une sortie à valeur réelle à partir des entrées. Les applications typiques incluent la modèlisation de processus statiques, l'interpolation et l'approximation de fonctions. De plus, en ingénierie, les modèles sont souvent destinés à être embarqués, par exemple dans des contrôleurs temps-réel. Dans ce cas, le but de la modélisation est d'obtenir un modèle à faible complexité. En régression, les connaissances *a priori* peuvent prendre différentes formes comme des points particuliers à valeurs connues ou des informations sur les dérivées de la fonction à estimer. De plus, dans de nombreuses situations pratiques, les données expérimentales peuvent être difficiles et coûteuses à obtenir, mais des modèles complexes, plus ou moins précis, peuvent exister et fournir des connaissances *a priori* sous la forme de données de simulation.

Identification. L'identification de systèmes s'intéresse aux processus dynamiques et diffère en cela de la régression. En effet, pour un système dynamique, la sortie à valeur réelle dépend non seulement des entrées, mais aussi de l'état du système, c'est-à-dire d'actions ou d'évenements antérieurs. Pour une classe particulière de systèmes, le problème d'identification peut être approché en considérant les entrées et sorties retardées du système comme des entrées du modèle. Les connaissances *a priori* en identification sont de différentes natures, incluant des bornes sur des variables physiques ou des propriétés dynamiques comme la stabilité. Dans certains cas, une structure de modèle prédéfinie peut être imposée. Cette structure peut correspondre soit à la structure réelle du système ou simplement être avantageuse pour l'utilisation du modèle, comme par exemple le calcul d'une commande optimale. Dans ce cadre, la thèse s'intéresse particulièrement à l'identification de systèmes hybrides, où le système est considéré comme commutant entre différents modes de fonctionnement. Pour une classe de systèmes, ce problème peut être vu comme l'approximation de fonctions non-regulières (*non-smooth*) ou la régression par morceaux.

Outre l'approche interdisciplinaire, la thèse propose de suivre deux approches complémentaires, bien que contradictoires, pour la modélisation.

- *Apprentissage de type boîte noire.* Les techniques d'apprentissage en aveugle à partir de données peuvent être rapidement appliquées à un problème particulier. Ces techniques doivent être étudiées dans le but d'obtenir les meilleures performances possibles avec le moins de connaissances *a priori* sur le problème et à partir d'un nombre de données minimum.

- *Connaissances a priori.* Les connaissances disponibles sur une application particulière constituent la meilleure aide pour résoudre le problème et doivent être utilisées au maximum. Ainsi, les méthodes de type boîte noire doivent être transformées en méthodes de type *boîte grise* permettant d'inclure le plus de connaissances le plus efficacement possible, tant d'un point de vue calculatoire que pratique.

Chapitre 1 – Introduction à l'apprentissage statistique

Le chapitre 1 fournit une introduction concise au problème d'apprentissage d'un modèle à partir de données expérimentales. Dans ce contexte, le but est de minimiser l'erreur de généralisation, c'est-à-dire l'erreur commise par le modèle sur des données inconnues lors de l'apprentissage. En pratique, cette erreur de généralisation, aussi appelée « risque », n'est pas calculable et doit être estimée. Ainsi, un algorithme d'apprentissage cherchera à minimiser l'erreur empirique, c'est-à-dire l'erreur sur l'ensemble des données disponibles, appelé *base d'apprentissage.* Cependant, en modélisation non-linéaire, minimiser l'erreur empirique peut conduire à un phénomène connu sous le nom de « sur-apprentissage » (*overfitting*) : un modèle peut apprendre « par cœur » l'ensemble de la base d'apprentissage, sans pour autant être capable de généraliser.

Le chapitre décrit de manière formelle les différents problèmes étudiés dans la thèse (la classification, la régression et l'identification de systèmes) en donnant des exemples illustratifs du phénomène de sur-apprentissage.

Chapitre 2 – Classification par machines à vecteurs de support

Les machines à vecteurs de support (SVMs) sont introduites dans le chapitre 2 pour la classification, qui est aussi leur application originelle. Trois variantes du problème d'apprentissage des SVMs sont présentées avec leurs avantages et inconvénients. La première conduit à un problème d'optimisation quadratique, alors que la seconde à un problème d'optimisation linéaire. La troisième, connue sous le nom de *Least Squares SVM* (LS-SVM), est une simplification de la méthode conduisant à résoudre un système d'équations linéaires.

Une courte partie du chapitre s'attache à expliquer quelques bases de la théorie permettant de formaliser les caractéristiques des SVMs qui peuvent se résumer ainsi :

- *Théorie.* Les SVMs reposent sur la théorie de l'apprentissage statistique, laquelle permet de formuler des bornes sur l'erreur de généralisation, c'est-à-dire de garantir une certaine performance du classifieur.

- *Interprétation.* Les SVMs bénéficient d'une interprétation géométrique permettant une introduction rapide aux concepts fondamentaux des SVMs.

- *Optimisation convexe.* Le problème d'apprentissage SVM peut se formuler comme un programme d'optimisation convexe, qui a une solution unique et pour lequel des optimiseurs efficaces existent. Les problèmes classiques rencontrés en apprentissage, comme les minima locaux, sont ainsi évités. Il faut cependant noter que le problème général d'apprentissage prenant en compte le réglage des hyperparamètres, comme la pondération de l'erreur empirique ou les paramètres internes des noyaux, n'est pas convexe.

Le chapitre 2 propose aussi une catégorisation des méthodes permettant l'incorporation de connaissances *a priori* dans l'apprentissage de classifieurs SVM. Cette revue de la littérature met en évidence la quantité de travaux existant dans ce domaine. En particulier, il est à noter que la grande majorité des travaux concernent l'incorporation d'invariances de classe. Le chapitre se termine par une unification d'un grand nombre des méthodes sous la forme d'un problème d'optimisation SVM, étendu par des contraintes additionnelles et dans lequel un terme est ajouté au critère à minimiser.

Publications liées au chapitre 2

Précédents travaux sur la classification :

[C1] F. Lauer, M. Bentoumi, G. Bloch, G. Millerioux, and P. Aknin. Ho–Kashyap with early stopping versus soft margin SVM for linear classifiers – an application. Dans : *Advances in Neural Networks, Proc. of the Int. Symp. on Neural Networks (ISNN), Dalian, Chine*, vol. 3173 de *LNCS*, pages 524–530, 2004.

[J1] F. Lauer and G. Bloch. Ho–Kashyap classifier with early stopping for regularization. *Pattern Recognition Letters*, 27(9):1037–1044, 2006.

[J2] F. Lauer, C. Y. Suen, and G. Bloch. A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40(6):1816–1824, 2007.

La revue et catégorisation des méthodes pour l'incorporation de connaissances a priori dans l'apprentissage SVM pour la classification a été présentée dans

[J3] F. Lauer and G. Bloch. Incorporating prior knowledge in support vector machines for classification: a review. *Neurocomputing*, 71(7-9):1578–1594, 2008.

Chapitre 3 – Régression par machines à vecteurs de support

Le chapitre 3 présente les machines à vecteurs de support dans le cadre de la régression. Le but principal de ce chapitre est d'introduire les notions nécessaires pour la suite du manuscrit. Par ailleurs, le chapitre décrit certains avantages théoriques des SVMs pour la régression avant de les illustrer sur quelques exemples numériques. En particulier, la régression par SVMs est appliquée à l'identification de systèmes non-linéaires. Les avantages suivants des SVMs peuvent être cités.

- Les SVMs pour la régression sont basés sur la théorie de l'apprentissage statistique, qui a montré son efficacité sur de nombreuses applications en classification, et qui fournit des bornes non-asymptotiques sur l'erreur de généralisation. Le principal enseignement est que le contrôle de la capacité (ou de la complexité) du modèle, inclus dans l'apprentissage SVM, est crucial. Il permet notamment au modèle de généraliser à partir de peu de données.
- Les SVMs pour la régression utilisent des fonctions à noyaux (*kernel functions*) pour traiter l'approximation de fonctions non-linéaires, ce qui permet au modèle de généraliser dans des espaces à grande dimension, éventuellement infinie. Grâce aux fonctions à noyaux, le problème ne repose plus sur la dimension des données mais sur leur nombre. En effet, le modèle peut s'exprimer comme une somme pondérée des fonctions à noyaux calculées en chaque point de la base d'apprentissage.
- Les SVMs pour la régression utilisent une fonction d'erreur robuste aux valeurs aberrantes.
- Comme pour la classification, l'apprentissage SVM pour la régression revient à résoudre un problème d'optimisation convexe et peut donc être efficacement implémenté avec la garantie d'obtenir une solution globale. Cependant, la nécessité de régler des hyperparamètres conduit ici aussi à un problème non-convexe dans son ensemble.
- L'apprentissage SVM conduit à des modèles parcimonieux par une sélection automatique des vecteurs de support nécessaires pour résoudre le problème.

Publications liées au chapitre 3

La régression par SVMs a été appliquée à l'identification de systèmes non-linéaires dans

[C2] F. Lauer and G. Bloch. Méthodes SVM pour l'identification. Dans : *Journées Identification et Modélisation Expérimentale (JIME), Poitiers, France*, 2006.

Chapitre 4 – Incorporation de connaissance *a priori* dans l'apprentissage SVM pour la régression

Bien qu'un grand nombre de méthodes soient disponibles pour l'incorporation de connaissances *a priori* dans l'apprentissage SVM en classification, peu sont directement transférables au cas de la régression. Le chapitre 4 propose donc une méthode générale permettant d'inclure différentes formes de connaissances dans l'apprentissage SVM pour la régression, comme des valeurs particulières connues en certains points ou des bornes sur les dérivées de la fonction à estimer. De plus, cette méthode permet de considérer des connaissances sous forme de relations entre plusieurs fonctions à estimer, et étend donc la régression par SVM au cas multi-sorties. Toutes ces connaissances peuvent être incluses par simple ajout de contraintes linéaires en les paramètres au problème d'optimisation linéaire correspondant à l'apprentissage SVM. L'apprentissage incorporant les informations *a priori* reste donc un problème d'optimisation convexe, limitant ainsi l'accroissement de la complexité.

Un certain nombre d'exemples illustrant les différentes possibilités offertes par la méthode proposée sont donnés tout au long du chapitre. La méthode est finalement testée sur une application réelle : l'estimation de la fraction de gaz résiduels dans un cylindre de moteur. Dans ce contexte applicatif, les expériences sont complexes et coûteuses. Cependant, un simulateur, construit à partir de connaissances physiques, est disponible. Celui-ci peut fournir des données de simulation dans des zones non explorées lors des expériences. Le problème est donc de combiner ces données, éventuellement biaisées, avec très peu de données expérimentales afin d'obtenir un modèle précis et suffisamment simple pour pouvoir être embarqué. Diverses méthodes pour effectuer cette combinaison ont été testées et la plus performante utilise des informations sur la forme de la fonction suggérée par les données de simulation plutôt que sur les valeurs exactes de ces données. Cette approche nécessite deux étapes. Un premier modèle SVM est estimé par un premier apprentissage sur les données de simulation uniquement, puis utilisé pour calculer des valeurs de la dérivée *a priori* en certains points. Le modèle final est ensuite déterminé sur les données expérimentales en contraignant sa dérivée à correspondre à la dérivée *a priori*.

Publications liées au chapitre 4

> *La méthode générale pour l'incorporation de connaissances a priori dans l'apprentissage SVM pour la régression est présentée dans*
>
> [J4] F. Lauer and G. Bloch. Incorporating prior knowledge in support vector regression. *Machine Learning*, 70(1):89–118, 2008.
>
> *L'application de cette méthode à l'estimation de la fraction de gaz résiduels dans un cylindre de moteur est décrite dans*
>
> [C3] G. Bloch, F. Lauer, G. Colin, and Y. Chamaillard. Combining experimental data and physical simulation models in support vector learning. Dans : *Proc. of the 10th Int. Conf. on Engineering Applications of Neural Networks (EANN), Thessaloniki, Grèce*, vol. 284 of *CEUS Workshop Proceedings*, pages 284–295, 2007.
>
> [J5] G. Bloch, F. Lauer, G. Colin, and Y. Chamaillard. Support vector regression from simulation data and few experimental samples. *Information Sciences*, 178(20):3813–3827, 2008.
>
> *Par ailleurs, l'idée principale de la méthode et de son application apparaît aussi dans un chapitre d'ouvrage promouvant plus généralement l'approche boîte grise :*
>
> [L1] G. Bloch, F. Lauer, and G. Colin. On learning machines for engine control. Dans *Computational Intelligence in Automotive Applications*, D. Prokhorov (Ed.), vol. 132 de *Studies in Computational Intelligence*, Springer, pages 125–142, 2008.

Chapitre 5 – Identification de systèmes hybrides

Un système hybride est un système dynamique commutant entre différents modes de fonctionnement, habituellement de types connus et le plus souvent linéaires. L'identification de tels systèmes est un problème actuel et peu étudié en comparaison à l'identification de systèmes linéaires ou même non-linéaires. En effet, en règle générale, faire des hypothèses fortes sur la structure du modèle permet de réduire la complexité du problème. Cependant, bien qu'utilisant une structure simplifiée et prédéfinie, construire un modèle hybride se révèle plus difficile que d'identifier un système non-linéaire sans *a priori* de structure. En fait, modéliser un système hybride à partir de données expérimentales comporte deux sous-problèmes intrinsèquement liés :

- *Classification.* Pour chaque point de donnée, le mode de fonctionnement doit être identifié.
- *Régression.* Pour chaque mode, le modèle estimant la dynamique du système dans ce mode doit être déterminé.

Le cas trivial dans lequel le mode est connu pour chaque point revient à résoudre plusieurs problèmes classiques d'identification à partir des données associées à chaque mode uniquement. La thèse se concentre donc sur le cas où le mode est inconnu et doit être estimé.

Il existe deux grandes classes de systèmes hybrides : les systèmes à commutations arbitraires et les sytèmes par morceaux. Dans le premier cas, il n'existe pas de relation entre l'état courant du système et son mode de fonctionnement, qui dépend d'une variable extérieure au système. Dans le deuxième cas, l'état du système détermine son mode de fonctionnement au travers d'une partition de l'espace d'état. Chaque région (ou morceau) de cette partition correspond à un mode particulier. Dans les deux cas, les problèmes de classification et de régression sont indissociables. De plus, lors de l'identification d'un modèle de système par morceaux, il est nécessaire d'estimer la partition de l'espace, ce qui revient à un problème de reconnaissance de formes (ou de classification supervisée) comme étudié dans le chapitre 2.

L'identification de systèmes hybrides est donc un champ de recherche interdisciplinaire par définition et un domaine intéressant pour les méthodes décrites dans cette thèse, traitant à la fois des problèmes de classification et de régression. Dans ce cadre, la thèse propose une approche basée sur la régression par SVMs. Le problème d'identification est reformulé en un programme d'optimisation non-linéaire sous contraintes linéaires, et donc non-convexe. Les non-linéarités impliquées dans le critère de ce programme d'optimisation sont des produits de variables continues et positives. Le critère est donc donné sous la forme d'une fonction « douce » (*smooth*), contrairement à d'autres approches basées sur l'optimisation mixte impliquant des variables entières. La méthode ainsi proposée est ensuite étendue pour traiter les cas suivants.

- *Systèmes hybrides non-linéaires.* A notre connaissance, l'identification de systèmes commutants entre des dynamiques non-linéaires, arbitraires et inconnues n'a pas été traitée dans la littérature. En s'inspirant des SVMs, cette thèse propose l'inclusion de modèles à noyaux pour traiter ce problème.
- *Bruit commutant.* La méthode permet de considérer des niveaux de bruits différents pour chaque mode. De plus, l'algorithme proposé est capable de s'adapter automatiquement à ces différents niveaux.
- *Estimation du nombre de modes.* Dans sa forme originelle, la méthode nécessite la connaissance du nombre de modes. Une procédure est donc proposée pour estimer ce nombre *a posteriori* en fournissant initialement à l'algorithme un nombre de modes maximal.
- *Estimation de l'ordre du système.* Dans le cas où l'ordre du système est inconnu, l'apprentissage est modifié pour permettre la sélection automatique des régresseurs. Cette méthode permet également de traiter le cas où l'ordre du système varie d'un mode à l'autre.

Le chapitre 5 se conclut par une série d'exemples numériques illustrant la méthode sur divers cas possibles (systèmes à commutations ou par morceaux, linéaires ou non-linéaires. . .).

Publications liées au chapitre 5

> *La méthode proposée pour l'identification de systèmes hybrides est décrite dans*
>
> [C4] F. Lauer and G. Bloch. A new hybrid system identification algorithm with automatic tuning. Dans : *Proc. of the 17th IFAC World Congress, Séoul, Corée du Sud*, pages 10207–10212, 2008.
>
> [C5] F. Lauer and G. Bloch. Switched and piecewise nonlinear hybrid system identification. Dans : *Proc. of the 11th Int. Conf. on Hybrid Systems: Computation and Control (HSCC), St. Louis, MO, USA*, vol. 4981 of *LNCS*, pages 330–343, 2008.

Chapitre 6 – Conclusions et perspectives

Le chapitre 6 donne les conclusions générales et met en valeur les problèmes restant ouverts.

- *Classification.* Pour l'incorporation d'invariances de classe, les méthodes passées en revue dans le chapitre 2 permettent de ne traiter que peu d'invariances simumltanément et de manière efficace. Le problème de l'inclusion de multiples invariances dans l'apprentissage de classifieurs SVM reste donc ouvert. Par ailleurs, la combinaison de différentes méthodes complémentaires semble être une piste pour l'augmentation des performances des classifieurs.

- *Régression par SVM pour l'identification de systèmes.* Un des verrous majeurs pour l'application de la régression par SVM à l'identification de systèmes dynamiques reste l'apprentissage de modèles récurrents (ou à erreur de sortie), comme démontré par d'autres travaux [174]. Ce problème a cependant été traité dans le cadre de l'apprentissage LS-SVM [256] mais au détriment de la plupart des propriétés intéressantes des SVMs.

- *Incorporation de connaissances a priori dans l'apprentissage SVM pour la régression.* La méthode proposée dans le chapitre 4 considère l'ajout de contraintes au problème d'apprentissage SVM par programmation linéaire. L'application d'une méthodologie similaire à la régression par LS-SVM ou par programmation quadratique est possible. Cependant, pour cette dernière, les contraintes peuvent être ajoutées au problème dans sa forme primale ou duale. Dans un cas, l'application est directe. Dans l'autre, les contraintes peuvent conduire à une modification de la structure du modèle résultant. Une étude de cet effet semble donc nécessaire.
  Par ailleurs, la méthode proposée implique le réglage d'un certain nombre d'hyperparamètres. Les valeurs de ces paramètres de haut niveau peuvent être fixées selon le degré de confiance sur les connaissances *a priori*. De plus, il a été montré, sur une application particulière, que la méthode n'est pas très sensible à des variations de ces paramètres autour de leur valeur optimale. Cependant, ces expériences considèrent un nombre limité de paramètres et une étude de la sensibilité de la méthode pour un nombre plus important peut être envisagée. De plus, quand aucune information *a priori* ne permet de fixer ces paramètres, leur réglage reste un problème ouvert.

- *Identification de systèmes hybrides.* La méthode proposée pour l'identification de systèmes hybrides repose sur l'optimisation non-linéaire sous contraintes et implique un nombre de variables croissant avec le nombre de données. Ceci limite l'applicabilité de la méthode à de petits jeux de données pour lesquels un optimum peut être trouvé dans un laps de temps raisonnable. La question ouverte est donc de pouvoir traiter des problèmes à grande dimension. Le nombre de variables étant associé au nombre de contraintes, une approche prometteuse pour y répondre consisterait à reformuler le problème en un programme d'optimisation non-linéaire et non-contraint.

Contents

Preface

Context

In automatic control, obtaining an accurate model of the process is the key issue that influences the overall design of the control scheme. When physical knowledge is not sufficient or when first-principle laws are too complex to be implemented in real-time controllers, the models are often built from experimental data. The thesis focuses on this latter problem, i.e. the determination of a model of a system based on input-output data. Thus stated, this problem is very general and has been the concern of various research communities depending on the type of system to be modeled, the nature of its inputs and outputs and finally the model itself. Looking at the nature of the outputs, two major types of problems may be defined: classification, in which the outputs take only discrete values, and modeling, in which the outputs are usually real numbers. Then modeling can be further divided into regression and dynamical system identification.

Classification aims at finding a model assigning an input pattern to a class, i.e. recognizing the object, sound, magnetic signature, and so on, represented by the pattern. In this setting, the outputs take only discrete values. A typical example is handwriting recognition, in which the model must output the character represented by the input image.

Regression aims at finding a model mapping the inputs to a real-valued output. Typical applications include modeling of static processes, interpolation and function approximation. Besides, in engineering, the aim of modeling is to obtain a low complexity model of a system for which a model may already be available but is not embeddable.

System identification is concerned with dynamical processes and thus differs from regression in that the real-valued output of the system depends not only on the inputs but also on the state of the system, i.e. on previous events. This problem can be tackled (for a particular class of systems) by using past measured outputs as additional inputs to the model.

> *"Constructing models from observed data is a fundamental element in science. [. . .] The art and technique of building mathematical models of (dynamic) systems is crucial to many application areas. Hence, many scientific communities are active in developing theory and algorithms. With a few exceptions, this has taken place in surprisingly separated and isolated environments, with journals and conferences of their own. So, we see separate subcultures in the general problem area, and it would be highly desirable to encourage more active exchanges of ideas. In particular, I am sure that the System Identification community would benefit from an influx of new ideas from other cultures."*
>
> — Lennart Ljung
> *Perspectives on system identification*, invited paper at the 17th IFAC World Congress, Seoul, South Korea, July 2008.

Approach

Machine learning. Among all the available approaches, machine learning offers interesting features such as a sound theory, a unified framework for classification and regression, and computationally efficient algorithms. In particular, the Support Vector Machines (SVMs) crystallize all these advantages and have been a source of intensive research over the last fifteen years. Acknowledging the state-of-the-art performances obtained in numerous applications with SVMs, the thesis places these learning machines at the center of the study. The goal is to promote an interdisciplinary approach, in which tools and methods developed by the machine learning community can help to solve typical problems studied by the automatic control community. More precisely, SVMs, originally developed for classification purposes, are also studied in the context nonlinear modeling and dynamical system identification.

Optimization. The thesis also strongly relies on optimization. As a consequence, the results and contributions are often presented as optimization problems (or programs), for which solutions can be obtained thanks to suitable optimizers, rather than theorems. Again, the idea is to benefit from advances in other fields of research. A similar tendency can be observed in other subfields of automatic control, where control or system analysis problems often lead to Linear Matrix Inequalities (LMI), efficiently solved thanks to interior-point algorithms. The main difference is that LMI constitute feasibility problems, whereas in the present work, minimization problems are considered.

The computational advantages of convex optimization, that has now become a "push-button" technology, can explain the growing literature involving optimization problems as solutions of field-specific problems. However, non-convex problems should not be considered as intractable, but rather as more difficult. The bulk of the thesis concentrates on convex methods, before turning to a non-convex problem: hybrid system identification.

Grey box approach. Focusing on nonlinear systems, the thesis will mainly discuss the use of nonparametric models of black box nature. These black box models are determined by learning from data. This approach, also known as agnostic learning, is at the opposite of white box modeling based on first-principle laws and physical meaning of parameters. Instead of arguing in favor of one or the other of these two approaches, the thesis promotes a third approach, known as grey box modeling. In this approach, black box training algorithms are modified to take into account prior knowledge on the problem. This allows to benefit from experimental data and the capabilities of learning machines as well as the information usually available in many tasks.

The main claim of the thesis beside the merits of the interdisciplinary approach can be summarized as follows. When trying to solve a modeling problem, two complementary, though contradictory, tracks should be followed.

- *Black box learning* techniques are very efficient and can be applied very quickly to a particular problem. These techniques should be investigated to yield the best performance with the least amount of knowledge and the least number of data.

- *Prior knowledge* is essential, constitutes the best help to solve a problem and should always be incorporated when available. Therefore, black box algorithms should be turned into grey box algorithms and include the largest amount of knowledge in the most computationally and practically efficient manner.

In pattern recognition, prior knowledge is usually considered in the form of class-invariance, i.e. the output of the classifier must remain unchanged if a particular transformation is applied to the input pattern. For instance, invariance to translations or rotations of the image are often included in handwriting recognition systems.

In regression, prior knowledge may take various forms such as particular known points or information on the derivatives of the function to approximate. For numerous situations met in engineering, experimental data are difficult or expensive to obtain, but complex models, more or less accurate, exist and can provide prior knowledge in the form of simulation data. In this case an efficient modeling procedure should be able to take into account both the experimental and simulation data.

In system identification, prior knowledge on the system can be of various nature, including bounds on physical variables or dynamical properties such as stability. In some cases, a predefined structure may also be imposed on the model. This structure may either correspond to the true structure of the system or simply be convenient for control purposes. In particular, this thesis will focus on hybrid system identification, where the system is assumed to switch between different modes and dynamics. In some cases, this problem relates to non-smooth function approximation or piecewise regression.

Thesis outline

The structure of the thesis follows the chronological development of the SVMs, i.e. classification is presented before regression, which is then extended to system identification. However, more emphasis is placed on regression and system identification rather than classification. Besides, the thesis presents three of the most common formulations of SVMs, namely, the original quadratic programming SVM (QP-SVM), the linear programming SVM (LP-SVM) and the least squares SVM (LS-SVM). However, the bulk of the thesis focuses on QP-SVM and LP-SVM.

Chapter 1 introduces the general problem of learning from data and gives a formal presentation of the three particular problems studied in the thesis: classification, regression and system identification. The aim here is to provide the necessary background for the next chapters.

Chapter 2 introduces the SVMs for classification. Both the quadratic and linear programming formulations are presented before the LS-SVM method. Then the main forms of prior knowledge that are considered in classification or pattern recognition are defined. Based on a hierarchical categorization of these types of knowledge, a review of the methods for the incorporation of prior knowledge in the SVMs for classification is provided. A second categorization, based on the means used to include the knowledge, is proposed and related to the previous one. The chapter ends with an example in handwritten digit recognition, where class-invariance with respect to a transformation of the input pattern is incorporated by the use of virtual samples.

Chapter 3 presents the Support Vector Regression (SVR) problem and algorithms. The primary aim is to provide an introduction for readers unfamiliar with SVR and to give the necessary background for the next chapters. Theoretical properties described throughout the chapter are then illustrated in some experiments. In particular, SVR is applied to nonlinear dynamical system identification with few data and its robustness to outliers is studied on a function approximation example.

Chapter 4 develops a general method for the incorporation of prior knowledge in the SVR learning. Prior knowledge to be included can take many forms ranging from particular values at particular points or conditions on the derivatives of the function to be approximated. This method also allows one to consider relationships between multiple functions to be estimated and thus extends SVR to deal with multiple outputs. The method is applied to the estimation of the in-cylinder residual gas fraction of a Spark-Ignition engine. In this application, very few experimental samples are available, but the model can be enhanced thanks to a simulator built from physical knowledge.

Chapter 5 focuses on the problem of hybrid dynamical system identification. A hybrid system is a particular nonlinear dynamical system that switches between different dynamics (or modes), usually of known types, i.e. most of the time linear. Identification of such systems is a typical application of mixed approaches and interdisciplinary problems. The three major problems studied in the thesis are indeed intrinsically mixed together in one single problem: estimate both the modes to which the data points belong (classification) *and* the submodels (regression) governing the dynamics (system identification) inside each mode. A nonlinear optimization approach, based on support vector regression, is proposed to tackle this problem and then further extended to: (i) consider different noise levels for the different modes; (ii) automatically tune itself to the noise levels; (iii) estimate the number of modes and the order of the system; (iv) and finally, consider nonlinear submodels in kernel form to deal with unknown and arbitrary nonlinearities.

Chapter 6 gives concluding remarks, highlights open issues and discusses the possible perspectives.

Each chapter is self-contained and can be read independently of the others. However, some basic notions on kernel functions, introduced in Sect. 2.1.3 for classification, will not be recalled in chapter 3 for regression. Similarly, the discussion on chapter 4 about including prior knowledge in SVR, assumes that the reader is familiar with the linear programming form of SVR introduced in Sect. 3.1.2. Finally, chapter 5 requires knowledge of SVR with the ε-insensitive loss function defined in Sect. 3.1 and of the automatic tuning techniques discussed in Sect. 3.1.3.

At the end of each chapter, a section entitled "Further reading" provides the reader with information about uncovered material and related references.

Contributions

The main results of the thesis can be summarized and related to the author's publications (listed on the next page) as follows.

Classification and prior knowledge

Previous works on classification and pattern recognition [C1, J1, J2] led the author to write a synthesis of methods for the incorporation of prior knowledge in Support Vector Machines for classification [J3]. Two categorizations are proposed in this thesis: one for the methods and one for the types of knowledge included in the learning. This review shows that most of the works in the field focus on transformation-invariance and that the combination of complementary methods should be further investigated. In addition, a unifying framework is proposed in the form of an optimization program with a composite criterion and additional constraints given for each method.

Regression and prior knowledge

Despite the large number of the methods described in [J3], few of them can be directly transposed to Support Vector Regression (SVR). Therefore a general method allowing to include prior knowledge in the SVR learning is proposed in this thesis [J4]. It is shown that including information on the function, its derivatives or between multiple outputs can be formulated as a linear program. This method is successfully applied to a difficult problem where very few data are available, namely, the estimation of the in-cylinder residual gas fraction in spark ignition engine [C3, J5]. These results also appear in a book chapter [B1] promoting the use of machine learning in a grey box approach for automotive applications.

Hybrid system identification

One of the main contribution of the thesis is a new method for hybrid system identification, originally proposed in [C4]. The algorithm can deal with cases where the noise level also switches with the system and automatically tune itself in accordance with the noise levels. The method is then extended by introducing nonlinear submodels able to estimate arbitrary and unknown nonlinearities [C5]. In addition, the thesis provides extensions to systems with varying orders, multiple output systems and two estimation frameworks for the interpretation of the algorithms.

Author's bibliography[1]

Journal papers

[J1] F. Lauer and G. Bloch. Ho–Kashyap classifier with early stopping for regularization. *Pattern Recognition Letters*, 27(9):1037–1044, 2006.

[J2] F. Lauer, C. Y. Suen, and G. Bloch. A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40(6):1816–1824, 2007.

[J3] F. Lauer and G. Bloch. Incorporating prior knowledge in support vector machines for classification: a review. *Neurocomputing*, 71(7-9):1578–1594, 2008.

[J4] F. Lauer and G. Bloch. Incorporating prior knowledge in support vector regression. *Machine Learning*, 70(1):89–118, 2008.

[J5] G. Bloch, F. Lauer, G. Colin, and Y. Chamaillard. Support vector regression from simulation data and few experimental samples. *Information Sciences*, 178(20):3813–3827, 2008.

Book chapter

[B1] G. Bloch, F. Lauer, and G. Colin. On learning machines for engine control. In *Computational Intelligence in Automotive Applications*, D. Prokhorov (Ed.), vol. 132 of *Studies in Computational Intelligence*, Springer, pages 125–142, 2008.

Conference papers

[C1] F. Lauer, M. Bentoumi, G. Bloch, G. Millerioux, and P. Aknin. Ho–Kashyap with early stopping versus soft margin SVM for linear classifiers – an application. In: *Advances in Neural Networks, Proc. of the Int. Symp. on Neural Networks (ISNN), Dalian, China*, vol. 3173 of *LNCS*, pages 524–530, 2004.

[C2] F. Lauer and G. Bloch. Méthodes SVM pour l'identification. In: *Journées Identification et Modélisation Expérimentale (JIME), Poitiers, France*, 2006.

[C3] G. Bloch, F. Lauer, G. Colin, and Y. Chamaillard. Combining experimental data and physical simulation models in support vector learning. In: *Proc. of the 10th Int. Conf. on Engineering Applications of Neural Networks (EANN), Thessaloniki, Greece*, vol. 284 of *CEUS Workshop Proceedings*, pages 284–295, 2007.

[C4] F. Lauer and G. Bloch. A new hybrid system identification algorithm with automatic tuning. In: *Proc. of the 17th IFAC World Congress, Seoul, Korea*, pages 10207–10212, 2008.

[C5] F. Lauer and G. Bloch. Switched and piecewise nonlinear hybrid system identification. In: *Proc. of the 11th Int. Conf. on Hybrid Systems: Computation and Control (HSCC), St. Louis, MO, USA*, vol. 4981 of *LNCS*, pages 330–343, 2008.

[1] The papers are available for download through the author's homepage at http://fabien.lauer.googlepages.com/.

Notations

a	scalar
$\boldsymbol{a}$	column vector
$\boldsymbol{a}^T$	transpose of $\boldsymbol{a}$
a_i	ith component of $\boldsymbol{a}$
$\mathbf{0}$	vector of appropriate dimension with all components equal to 0
$\mathbf{1}$	vector of appropriate dimension with all components equal to 1
$\boldsymbol{A}$	matrix
$\boldsymbol{A}_i$	ith column of $\boldsymbol{A}$
A_{ij}	element at the ith row and jth column of $\boldsymbol{A}$
$\boldsymbol{I}$	identity matrix with $I_{ii} = 1$ and all other components equal to 0
$diag(\boldsymbol{a})$	diagonal matrix with the components of $\boldsymbol{a}$ on the diagonal
$\|a\|$	absolute value of a
$\\|\boldsymbol{a}\\|_2$	Euclidean or ℓ_2-norm of $\boldsymbol{a}$
$\\|\boldsymbol{a}\\|_1$	ℓ_1-norm of $\boldsymbol{a}$ computed as the sum over all its components of $\|a_i\|$
$\langle \boldsymbol{x}, \boldsymbol{z} \rangle$	inner product[1] between $\boldsymbol{x}$ and $\boldsymbol{z}$
$P\{a \mid b\}$	conditional probability of a given b
$P(x)$	cumulative distribution function of x
$p(x)$	probability density function of x
$x \sim \mathcal{D}$	x is randomly drawn from the probability distribution $\mathcal{D}$
$\mathbb{R}$	the set of real numbers

$\boldsymbol{x}_i$	ith sample vector
x_i^j	jth component of the ith sample vector
$\boldsymbol{X}$	observation matrix composed of sample vectors $\boldsymbol{x}_i^T$ as rows
$\boldsymbol{y}$	target vector defined as $\boldsymbol{y} = [y_1\ y_2\ \dots\ y_N]^T$
Z	set of samples $\boldsymbol{z}$
$\|Z\|$	cardinality of the set Z
$\boldsymbol{Z}$	matrix containing the samples $\boldsymbol{z} \in Z$ as rows: $\boldsymbol{Z} = [\boldsymbol{z}_1\ \dots\ \boldsymbol{z}_{\|Z\|}]^T$
$k(\cdot, \cdot)$	kernel function
$\boldsymbol{K}(\boldsymbol{A}, \boldsymbol{B})$	matrix defined by $K_{ij} = k(\boldsymbol{A}_i, \boldsymbol{B}_j)$
$\boldsymbol{K}$	kernel matrix $\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{X}^T, \boldsymbol{X}^T)$
Φ	nonlinear mapping
$\boldsymbol{\Phi}(\boldsymbol{x})$	image of $\boldsymbol{x}$ via the mapping $\Phi : \boldsymbol{x} \mapsto \boldsymbol{\Phi}(\boldsymbol{x})$
$Z \setminus S$	set difference $Z \setminus S = \{\boldsymbol{x} \mid \boldsymbol{x} \in Z, \text{and } \boldsymbol{x} \notin S\}$
$\text{sinc}(x)$	normalized sinc function $\text{sinc}(x) = \sin(\pi x)/\pi x$
$<, \leq, >, \geq$	component-wise inequality signs

s.t.	subject to
w.r.t.	with respect to
i.i.d.	independent and identically distributed

[1] For the sake of clarity, and when appropriate, this product will also be written as the dot product $\boldsymbol{x}^T\boldsymbol{z}$. In particular, this will be the case for finite dimensional vectors or matrices involved in linear programs based on Mangasarian's work.

Chapter 1

Learning from data

ABSTRACT. *This chapter formalizes the problem of learning from data, while defining some notations and key terms such as the generalization error. Then the problem is further detailed for classification, regression and system identification. Throughout the chapter, emphasis is put on nonlinear models and the issue of overfitting.*

THE following gives a formal presentation of the three main problems studied in the thesis: classification, regression and system identification. An exhaustive presentation would be out of the scope of the thesis and the reader is referred to textbooks such as [25, 113, 198] for more details. The choice here is to focus on material that will be used in the remaining of the thesis.

1.1 General problem

The general problem studied in this thesis can be stated as follows.

On the basis of N input–output samples (the training set*)*

$$(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_i, y_i), \dots, (\boldsymbol{x}_N, y_N), \tag{1.1}$$

where $\boldsymbol{x}_i \in \mathcal{X}$ is the ith input vector (or pattern or sample) and $y_i \in \mathcal{Y}$ is the corresponding output (or target), find the model f that best approximates the true underlying function mapping $\boldsymbol{x}$ to y.

In machine learning, finding f is referred to as supervised *learning* or *training.*

In learning theory, the training (available) data as well as the test (unknown) data are considered i.i.d. from an unknown but fixed probability distribution $\mathcal{D}$ defined over the pairs $(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$ with a corresponding probability density function $p(\boldsymbol{x}, y)$ [62]. Thus the output y is given by a fixed function t of the input $\boldsymbol{x}$ by $y = t(\boldsymbol{x})$. The goal of a learning machine is to find a function f from the class $\mathcal{F}$ of admissible functions (i.e. functions with a particular structure or satisfying basic assumptions) that approximates t so as to minimize the risk. Also known as the *generalization error*, the risk $R[f]$ of a model f is defined as the average loss implied by the choice of f and measured pointwise by a loss function $l(f(\boldsymbol{x}), y)$. This risk will be referred to as the *expected risk* (or the generalization error) and can be written for a particular loss function l as

$$R[f] = \mathbb{E}\left[l(f(\boldsymbol{x}), y)\right] = \int_{\mathcal{X}\times\mathcal{Y}} l(f(\boldsymbol{x}), y) p(\boldsymbol{x}, y) \ d\boldsymbol{x}dy, \tag{1.2}$$

where $\mathbb{E}\left[l(f(\boldsymbol{x}), y)\right]$ denotes the expectation of the loss $l(f(\boldsymbol{x}), y)$ over the distribution $\mathcal{D}$. However, the expected risk cannot be explicitly computed without knowing $p(\boldsymbol{x}, y)$. Thus learning algorithms have to rely on estimates of $R[f]$ based on the available data. When learning on a training set S, a straightforward estimate, known as the empirical risk (also know as the *training error*), is defined as[1]

$$E_S[f] = \frac{1}{N} \sum_{i=1}^{N} l(f(\boldsymbol{x}_i), y_i), \tag{1.3}$$

where $N = |S|$ is the cardinality of S (the number of samples in the training set).

Similarly, estimates of the generalization error have to be used to assess the algorithms and measure the performance of the models. Such common estimates are given below, after the definition of hyperparameters. The following sections will discuss different problem settings with the corresponding choices for the loss function l.

Hyperparameters

The parameters involved in the process of learning a model from data can be divided in two categories.

- *Parameters* $\boldsymbol{\theta}$. These are the trainable parameters of the model or the function implemented by the learning machine. Their values are determined by the learning algorithm.
- *Hyperparameters* $\boldsymbol{\eta}$. These are high-level (or design) parameters that may influence the training procedure or the general form of the model. They are the constants defining a particular instance of a learning algorithm or involved in the model. They are not usually determined by the learning algorithm, but are instead fixed at the design stage.

In short, $\boldsymbol{\theta}$ are the parameters of the learning machine, whereas $\boldsymbol{\eta}$ are the parameters of the learning algorithm.

Tuning the hyperparameters refers to the procedure employed to find the optimal values of the hyperparameters $\boldsymbol{\eta}$ minimizing the risk (1.2). Such a tuning is a complex task and usually requires to estimate the generalization performance of the model (see below). A common method is to perform a grid search, i.e. to test many values of $\boldsymbol{\eta}$ and to estimate the generalization error (and thus train the model) for each value. This procedure is very time consuming but can be sped up by starting with a coarse grid and then refining the search around the best values. However, for more than two hyperparameters, a full grid search usually becomes intractable.

Estimates of the generalization error

Out-of-sample error. The most straightforward procedure to estimate the risk (1.2) of a learning machine is to compute the average loss

$$E_{\widehat{S}}[f] = \frac{1}{|\widehat{S}|} \sum_{i=1}^{|\widehat{S}|} l(f(\boldsymbol{x}_i), y_i), \tag{1.4}$$

on an independent data set $\widehat{S}$. When the data in $\widehat{S}$ are used neither for training nor for tuning the hyperparameters, $\widehat{S}$ is referred to as the *test set* and the estimate of the generalization error $E_{\widehat{S}}[f]$ is known as the *test error*. On the other hand, when the data in $\widehat{S}$ are used for tuning purposes, $\widehat{S}$ is the *validation set* and $E_{\widehat{S}}[f]$ the *validation error*.

Cross-validation procedures. When the available data are too few, one cannot afford to retain a share of the training set for validation or test. In this case, k-fold cross-validation estimates of $R[f]$ can be used. They are computed as follows.

1. Divide the training set S in $k \in \mathbb{N}$ subsets S_j, $j = 1, \dots, k$, of N/k samples
2. For $j = 1$ to k,
   - train a model f on $S \setminus S_j$

[1] This is obtained by taking $p(\boldsymbol{x}, y) = \frac{1}{N} \sum_{i=1}^{N} \delta(\boldsymbol{x} - \boldsymbol{x}_i)\delta(y - y_i)$.

- evaluate the error $E_{S_j}[f]$ of f on the remaining subset S_j by (1.4)

3. Compute the k-fold estimate $R_{k\text{-fold}}[f]$ as the average error: $R_{k\text{-fold}}[f] = \frac{1}{k}\sum_{j=1}^{k} E_{S_j}[f]$

The larger k is, the more accurate the estimate is. But the computational load also grows with k. An almost unbiased cross-validation estimate, known as the *Leave-One-Out* (LOO) estimate, is obtained for the limit case $k = N$. For the validation of hyperparameters $\boldsymbol{\eta}$ on large training sets, the LOO procedure may become quite time consuming as it requires training the classifier N times for each value of $\boldsymbol{\eta}$. In these cases k-fold cross-validation with $k = 5$ or 10 offers a convenient alternative.

Other estimates of the generalization error. To avoid the computational cost of the LOO procedure during the tuning process, another approach consists in searching for the optimal hyperparameters minimizing an upper bound on the LOO estimate. A review and experimental comparison of such bounds in the context of classification with support vector machines can be found in [73].

1.2 Classification

Classification aims at finding a model assigning an input pattern (usually described by a set of features) to a class, i.e. recognizing the element represented by the pattern. Typical applications of classification include pattern recognition tasks such as image, character or speech recognition. In the field of automatic control, classification is also used for diagnosis applications, e.g. [147]. The following only describes the approach taken in the thesis, in which the classifier corresponds to a set of separating surfaces, or boundaries, in the input space. Other approaches to classification can be found in standard textbooks such as [75] or [25].

The supervised classification problem with n classes can be stated as follows.

Find a classifier f that determines correctly the class-membership (or label) of unknown inputs $\boldsymbol{x}$ (patterns) based on N labeled samples $(\boldsymbol{x}_i, y_i)$, $i = 1\dots, N$, where $y_i \in \{1, \dots, n\}$ is the label of the sample $\boldsymbol{x}_i \in \mathbb{R}^p$.

In this context, the output space $\mathcal{Y}$ contains only discrete values and the Hinge loss defined as

$$l(f(\boldsymbol{x}), y) = [f(\boldsymbol{x}) \neq y], \tag{1.5}$$

where [.] is 1 if the bracketed expression is true and 0 otherwise, is considered in (1.2). Here, the risk $R[f]$ of a classifier f can be seen as the probability that f misclassifies a sample randomly drawn from the distribution $\mathcal{D}$.

However, the Hinge loss being a non-smooth and non-differentiable function, it cannot be minimized conveniently. Therefore, learning algorithms usually implement smooth approximations of (1.5).

1.2.1 Binary classification

In this thesis, all the methods are presented in the context of binary classification (two-class problems), in which the labels $y \in \mathcal{Y} = \{-1, +1\}$ can only take two distinct values. In this case, the classification problem amounts to finding a single *separating surface* $\mathcal{S}$ dividing the input space $\mathcal{X}$ in two half-spaces, each one assigned to a class.

The separating surface can be described by a real-valued function h as $\mathcal{S} = \{\boldsymbol{x} : h(\boldsymbol{x}) = 0\}$. The corresponding classifier f then classifies a pattern $\boldsymbol{x}$ with respect to the side of $\mathcal{S}$ on which $\boldsymbol{x}$ lies. Its output is thus given by

$$f(\boldsymbol{x}) = \text{sign}(h(\boldsymbol{x})), \tag{1.6}$$

as shown in Fig. 1.1.

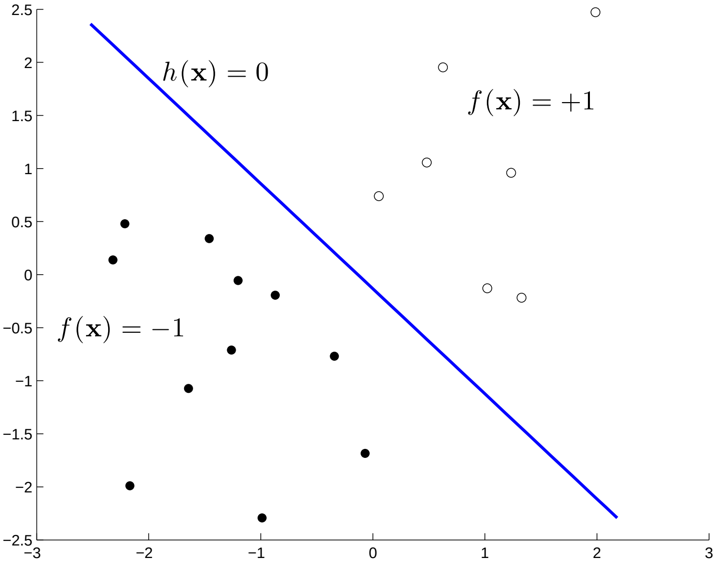


Figure 1.1: Linear classifier separating the white points from the black points by a hyperplane.

1.2.2 Linear classifiers

For linear classification, the separating surface $\mathcal{S}$ is a hyperplane defined by

$$\mathcal{S} = \{\boldsymbol{x} \ : \ h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = 0\}, \tag{1.7}$$

where the vector $\boldsymbol{w}$ and b are the parameters. Finding a hyperplane, or training a classifier, consistent with a training set $(\boldsymbol{x}_i, y_i),\ i = 1, \dots, N$, may be written as the feasibility problem

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x}_i + b) = y_i,\ i = 1, \dots, N, \tag{1.8}$$

or equivalently[2]

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) > 0,\ i = 1, \dots, N. \tag{1.9}$$

If there exists such a hyperplane, the training set is said to be *linearly separable.*

Defining the observation matrix $\boldsymbol{X} = [\boldsymbol{x}_1\ \boldsymbol{x}_2\ \dots\ \boldsymbol{x}_N]^T$ and the target vector $\boldsymbol{y} = [y_1\ y_2\ \dots\ y_N]^T$, problem (1.9) can be rewritten in matrix form as

$$\boldsymbol{D}(\boldsymbol{X}\boldsymbol{w} + b\mathbf{1}) > 0, \tag{1.10}$$

where $\boldsymbol{D} = diag(\boldsymbol{y})$.

1.2.3 Nonlinear classifiers

Most pattern recognition tasks are complex in nature, hence the need for nonlinear classifiers able to implement more complex separating surfaces.

Separating hyperplane in feature space

This thesis focuses on kernel methods for nonlinear classification. These methods allow one to use linear classifiers and their algorithms to deal with nonlinear problems. The overall approach is as follows.

[2] The strict inequality ensures that the output of the classifier is defined for all the training samples.

Nonlinear classification can be handled by mapping the data to a higher dimensional feature space F by

$$\mathbb{R}^p \ni \boldsymbol{x} \overset{\Phi}{\mapsto} \boldsymbol{\Phi}(\boldsymbol{x}) \in F, \tag{1.11}$$

and then performing linear classification (constructing a hyperplane) in this feature space. Indeed, it has been proved that a training set of N points represented in a feature space F of dimension $d_F > N$ can always be separated by a hyperplane [59]. The resulting classifier is given by

$$f(\boldsymbol{x}) = \text{sign}\ \left(\langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x})\rangle + b\right), \tag{1.12}$$

where $\boldsymbol{w} \in F \subset \mathbb{R}^{d_F}$ and $\langle ., . \rangle$ now stands for the inner product in the feature space F.

Example 1 (Nonlinear classification, taken from [192])**.** *Consider the data plotted in input space on the left of Figure 1.2. In this example, a nonlinear separating surface is required to correctly classify all the data. However, by using the nonlinear mapping*

$$\Phi : (x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2), \tag{1.13}$$

the data becomes linearly separable in feature space.

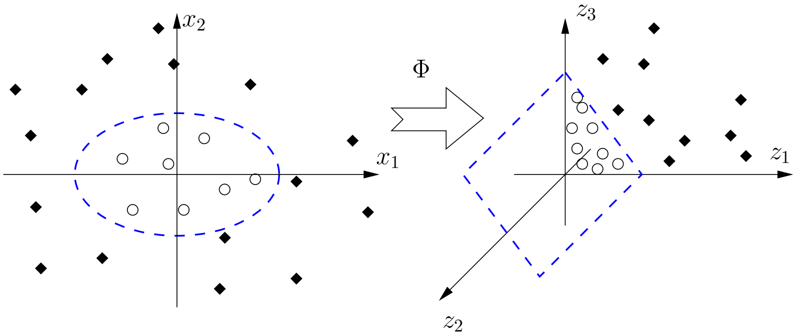


Figure 1.2: Nonlinear mapping $\Phi : (x_1, x_2) \mapsto (z_1, z_2, z_3)$ from the input space (*left*) to the feature space (*right*) allowing one to linearly separate the data.

Nonetheless, both computational and generalization performance can degrade as the dimension of F grows, a phenomenon often referred to as the *curse of dimensionality* [62]. In chapter 2, this problem will be conveniently overcome thanks to the *kernel trick* (see Sect. 2.1.3).

Overfitting

For small training sets, minimizing the training error may lead to overfitting. For instance, a classifier learning "by heart" all the training samples, could yield zero training error but would not be able to generalize to other samples. This is due to the unconstrained complexity of the model. To see this, consider classifying the data plotted in Fig. 1.3 with an unconstrained nonlinear classifier and a linear classifier, both minimizing their training error (left). Depending on the true and unknown distribution of the data, either the linear classifier leads to underfitting (middle) or the nonlinear classifier leads to overfitting (right). A more formal discussion on this topic is provided in section 2.1.5.

1.2.4 Multiclass problems

A very common approach to muticlass problems (where the number of classes $n > 2$) consists in building a set of binary classifiers, each one assigned to a particular task. Two methods can be used.

- *One-against-all* method. Each binary classifier is trained to separate one class from the others. This method requires to train n classifiers.

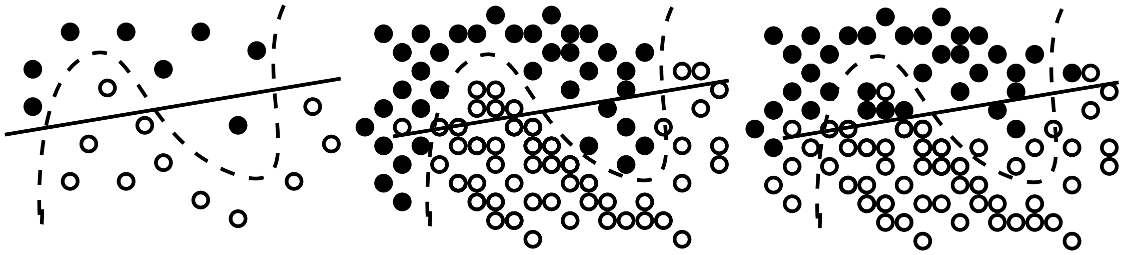


Figure 1.3: Linear (–) and nonlinear (- -) classifiers trained on the sparse data (*left*). Depending on the true data distribution, the linear classifier underfits (*middle*) or the nonlinear classifier overfits (*right*). This example is taken from [192].

- *One-against-one* method. Each binary classifier is trained to distinguish between two classes. This method requires training $\sum_{i=1}^{n-1} i = n(n-1)/2$ binary classifiers. However, as the training sets of each classifier contains the data of two classes only, the computation burden of training these classifiers is not necessarily greater than with the one-vs-one method.

There also exists some learning machines able to tackle directly multiclass applications such as neural networks [25, 160] or multiclass support vector machines (see Sect. 2.8).

1.3 Regression

Regression amounts to finding a model that is able to explain the real-valued dependent variable on the basis of explanatory variables. Applications of regression include statistics, modeling of static processes, knowledge discovery, and so on. The regression problem stated as a function approximation problem is the following.

From N samples $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}$, find the function f that can predict the output value $y \in \mathbb{R}$ on the basis of the regression vector $\boldsymbol{x} \in \mathbb{R}^p$ by $y = f(\boldsymbol{x})$.

In this context, the squared loss function

$$l(f(\boldsymbol{x}), y) = (y - f(\boldsymbol{x}))^2, \tag{1.14}$$

is usually considered in the risk (1.2), leading to the so-called Mean Squared Error (MSE). In the thesis, the notation MSE will refer to the test error computed on an independent test set $\widehat{S}$ as in (1.4) by

$$\text{MSE} = \frac{1}{|\widehat{S}|} \sum_{i=1}^{|\widehat{S}|} (y_i - f(\boldsymbol{x}_i))^2. \tag{1.15}$$

Remark 1. *It must be emphasized that classification, using the Hinge loss, is generally more difficult than regression, using a real-valued loss. There is no such thing as "small mistakes" in classification, whereas regression considers the magnitude of the errors. For instance, a small test error e of magnitude $|e| = 0.1$ on the real-valued output $h(\boldsymbol{x})$ of a classifier may account for an additional mistake with an effect of magnitude $1/N$ on the performance measure. In regression, the same error would merely have an effect of magnitude $e^2/N = 0.01/N$.*

Remark 2 (Parametric models and estimation theory)**.** *For parametric models $f_{\boldsymbol{\theta}}$, estimation theory is concerned with the estimation of the true parameters $\boldsymbol{\theta}_0$ rather than the output y. In this framework, the risk is defined as the average loss over the parameter space, computed by loss functions defined over the parameters as, e.g. for squared loss, $l(\boldsymbol{\theta}, \boldsymbol{\theta}_0) = \|\boldsymbol{\theta}_0 - \boldsymbol{\theta}\|_2^2$. In this context, the methods are often compared on the basis of asymptotic properties of the parameter estimates*

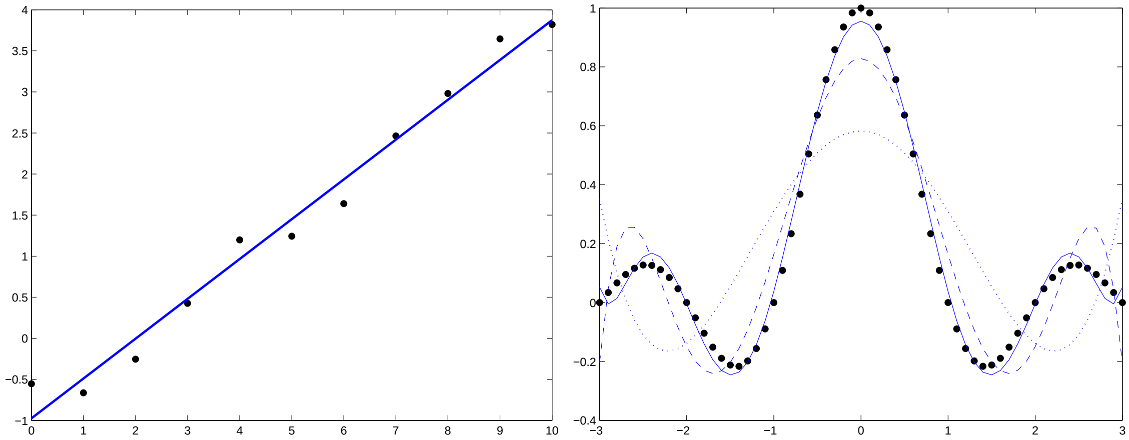


Figure 1.4: *Left:* linear regression from noisy data (black dots). *Right:* nonlinear regression by polynomial models (using the matlab function *polyfit*) of increasing degree, 5 $(\cdots)$, 7 (- -) and 8 (—), and thus increasing flexibility.

such as the bias or variance. However, as most of the thesis focuses on nonlinear modeling with nonparametric models, this framework is not considered and the models will be evaluated on the basis of their ability to predict the true output of the function, measured by their generalization error.

1.3.1 Linear regression

Linear regression assumes that the model is linear w.r.t. to the inputs $\boldsymbol{x}$. In the noiseless case, this problem can be stated as the feasibility problem

$$f(\boldsymbol{x}_i) = \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle = y_i, \ i = 1, \dots, N, \tag{1.16}$$

where the vector $\boldsymbol{w}$ contains the parameters of the model, or in matrix form

$$\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}, \tag{1.17}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1 \ \boldsymbol{x}_2 \ \dots \ \boldsymbol{x}_N]^T$ and $\boldsymbol{y} = [y_1 \ y_2 \ \dots \ y_N]^T$. However, in most cases, the data are corrupted by noise and the problem becomes the minimization the empirical risk (1.3), or equivalently the minimization of the norm of the residuals $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|$ as represented in Figure 1.4. If the norm of the residuals is the Euclidean norm, corresponding to a squared loss function for (1.3), then the so-called *least squares* solution is given by

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{1.18}$$

Different estimators can be defined by choosing different loss functions for (1.3) or norms of the residuals. Another famous estimator is obtained for the absolute loss function $l(f(\boldsymbol{x}), y) = |y - f(\boldsymbol{x})|$, corresponding to the ℓ_1-norm of the residuals.

1.3.2 Nonlinear regression

The global approach usually taken in nonlinear function approximation is to build a linear model on the basis of mapped inputs $\boldsymbol{\Phi}(\boldsymbol{x})$. Depending on the nature of the mapping $\Phi : \boldsymbol{x} \mapsto \boldsymbol{\Phi}(\boldsymbol{x})$, different classes of models are defined.

Parametric models

Parametric models have a fixed structure and number of parameters determined on the basis of prior knowledge. If the nonlinearities of the target function are known under a linearly parametrized form, the problem can be recast as a linear regression problem. For instance, if the model is known to contain the square root of the kth regressor, then an extended regression vector $\tilde{\boldsymbol{x}} = [\boldsymbol{x}^T \sqrt{x^k}]^T$ may be used to take the nonlinearity into account.

Nonparametric models

Nonparametric models refer to models where the structure is not defined *a priori* but is instead estimated from the data. In this context, the number of parameters is usually not fixed beforehand.

Polynomial models. Polynomial models implement a function f defined by a p-dimensional polynomial of degree d as

$$f(\boldsymbol{x}) = w_0 + \sum_{k=1}^{p} w^k x^k + \sum_{k_1=1}^{p}\sum_{k_2=1}^{p} w^{k_1 k_2} x^1 x^2 + \dots + \sum_{k_1=1}^{p} \dots \sum_{k_d=1}^{p} w^{k_1 \dots k_d} x^1 \dots x^d, \tag{1.19}$$

where x^k stands for the kth component of the regression vector $\boldsymbol{x}$. Depending on the degree d, these models can be used to approximate a large class of nonlinear functions (see Fig. 1.4). However, the number M of parameters w increases dramatically with the size of $\boldsymbol{x}$ and the degree d of the polynomial as

$$M = \frac{(d+p)!}{d!p!}. \tag{1.20}$$

This number clearly reflects that polynomial models suffer from the *curse of dimensionality.* The estimation of d can be performed by stepwise regression.

The model (1.19) can then be reformulated as a linear model

$$f(\boldsymbol{x}) = \tilde{f}(\tilde{\boldsymbol{x}}) = \theta_0 + \sum_{k=1}^{M} \theta^k \tilde{x}^k, \tag{1.21}$$

where $\theta^k \tilde{x}^k$ corresponds to the kth term in (1.19), with a change of variables from $\boldsymbol{x}$ to $\tilde{\boldsymbol{x}}$. Finally, this parametrized form can be estimated by standard linear regression methods.

Neural networks and basis functions. Another famous class of nonlinear models is based on the expansion

$$f(\boldsymbol{x}) = \sum_k \alpha_k f_k(\boldsymbol{x}), \tag{1.22}$$

where the so called *basis functions* f_k can be different for each k [241]. Typically, these basis functions are constructed by varying the parameters (β, γ) of a mother basis function $\kappa(\boldsymbol{x}, \beta, \gamma)$. For instance, the radial basis function (RBF), $f_k(\boldsymbol{x}) = \kappa(\|\boldsymbol{x} - \boldsymbol{\gamma}_k\|_{\boldsymbol{\beta}_k})$, is perhaps the most common of these. It is usually based on a quadratic norm defined by the matrix $\boldsymbol{\beta}_k$ as $\|\boldsymbol{x}\|^2_{\boldsymbol{\beta}_k} = \boldsymbol{x}^T \boldsymbol{\beta}_k \boldsymbol{x}$. For simple cases, the matrix $\boldsymbol{\beta}_k$ can be either the identity matrix or a scaled version of the identity matrix.

The training of a neural network based on basis functions, such as the RBF neural network (RBFNN), is divided in three steps [241]:

1. choose a mother basis function,
2. determine the number and the parameters of the basis functions (the centers $\boldsymbol{\gamma}_k$ and the matrix $\boldsymbol{\beta}_k$ for the RBFNN),
3. compute the optimal weights α_k for the expansion (1.22).

Once the basis functions have been chosen with their corresponding parameters, the model (1.22) is linear w.r.t. the remaining parameters α_k. Then these can be determined by linear regression methods such as the least squares method or by minimizing some suitable loss function.

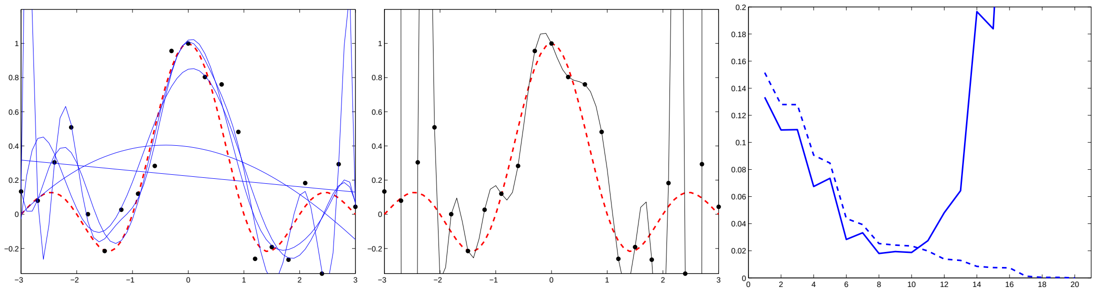


Figure 1.5: *Left:* Approximation of sinc(x) (- -) by polynomial models (—) of increasing degree d from noisy data (black points). *Middle:* Polynomial model with $d = 20$ (black line), leading to the minimum training error. *Right:* training (- -) and test (—) mean squared error versus the degree of the polynomial. The test error is minimum for $d = 8$.

Overfitting in regression

For nonlinear regression, minimizing the mean squared error on the training set S may lead to overfitting as shown on Fig. 1.5. In this example, the training error decreases as the flexibility of the model (the degree d of the polynomial) increases until a zero training error is achieved for $d = 20$. On the other hand, the test error, computed on 61 samples in the range $-3 \leq x \leq 3$, decreases for small d but increases when the flexibility of the model is large enough to model the noise. For $d = 20$, the test error is 3.9×10^3. This is also due to the divergent nature of the polynomial model.

1.4 System identification

In the field of automatic control, obtaining an accurate model of the dynamical system being controlled is an important task that can influence the overall performance of the control loop. System identification aims at finding such an appropriate model of the system based on first-principles equations and input-output data. The model can be of three types as defined in [241].

- White Box models: this is the case when a model is perfectly known; it has been possible to construct it entirely from prior knowledge and physical insight.
- Grey Box models: this is the case when some physical insight is available, but several parameters remain to be determined from observation data. Two sub-cases can be considered:
    - Physical modeling: a model structure can be built on physical grounds, which has a certain number of parameters to be estimated from data, e.g. a state space model of given order and structure.
    - Semi-physical modeling: physical insight is used to suggest certain nonlinear combinations of measured data signals. These new signals are then subjected to model structures of black box character.
- Black Box models: no physical insight is available nor used, but the chosen model structure belongs to families that are known to have good flexibility and have been "successful in the past."

In this section, black box modeling is considered. Grey box modeling including a particular form of the model will be discussed in section 1.4.3.

Remark 3 (Parametric identification)**.** *As for regression (see remark 2), linear system identification is usually tackled by using parametric models and studied in the framework of estimation theory (see e.g. [169]), which is not considered here.*

1.4.1 Black box modeling

For the sake of clarity, the presentation is limited to discrete-time and Single Input Single Output (SISO) systems. For a system of input $u_k \in \mathcal{U}$ and output $y_k \in \mathcal{Y}$ at time step k, black box modeling aims at finding a model between past observations

$$\boldsymbol{u}_{k-1} = [u_0 \; u_1 \; \dots \; u_{k-1}] \tag{1.23}$$

$$\boldsymbol{y}_{k-1} = [y_0 \; y_1 \; \dots \; y_{k-1}], \tag{1.24}$$

and future outputs y_k based only on observations of input-outputs pairs. The problem is thus to find a function g so that the model in predictor form

$$\hat{y}_{k|k-1} = g(\boldsymbol{u}_{k-1}, \boldsymbol{y}_{k-1}), \tag{1.25}$$

approximates correctly the behavior of the system as $y_k = \hat{y}_{k|k-1} + e_k$. The additive term e_k accounts for the fact that the next output y_k will not be an exact function of past data [241].

In black box modeling, the function g is usually considered as the concatenation of two mappings: one that maps the increasing sequence of inputs and outputs $[\boldsymbol{u}_{k-1}, \boldsymbol{y}_{k-1}]$ to a finite-dimensional vector $\boldsymbol{x}_k$ of fixed dimension, and one that maps this vector to the output space $\mathcal{Y}$. The function g is thus written as

$$g(\boldsymbol{u}_{k-1}, \boldsymbol{y}_{k-1}) = f(\boldsymbol{x}_k), \tag{1.26}$$

where $\boldsymbol{x}_k = \boldsymbol{\varphi}(\boldsymbol{u}_{k-1}, \boldsymbol{y}_{k-1})$ is called the *regression vector*[3] and $\boldsymbol{\varphi}$ is the mapping of all the past observations into the regression vector.

The problem of black box identification is thus decomposed in two steps.

1. *Define a mapping $\boldsymbol{\varphi}$ that builds the regression vector $\boldsymbol{x}_k$ from past inputs and outputs,*

2. *find the function f that maps this vector to the output space.*

The first step typically amounts to choose the lagged inputs and outputs kept in the regression vector depending on the class of the model one wants to construct. These model classes and their corresponding regression vectors are reviewed in the next section. Once the regression vector (or the mapping $\boldsymbol{\varphi}$) has been chosen, a training set $(\boldsymbol{x}_i, y_i)$, $i = 1, \dots, N$, containing the regression vector and the output for N different time steps can be generated. In most cases, the problem can thus be recast into a regression problem.

1.4.2 Building the regression vector

In parametric identification of linear systems, the choice of the regressors among $\boldsymbol{u}_{k-1}$ and $\boldsymbol{y}_{k-1}$, constituting $\boldsymbol{x}_k$, defines the class of the model. The standard nomenclature for these model classes is presented in Table 1.1. They can be constructed in the following order. The FIR model predicts the output based only on past inputs. The ARX model adds past outputs to its regressors. Adding prediction errors to the regressors of the ARX model yields the ARMAX model. The ARMA

Table 1.1: Classes of linear models with the corresponding choice of regressors.

Model		past inputs	past outputs	prediction errors
		$u_{k-1}, \dots, u_{k-n_u}$	$y_{k-1}, \dots, y_{k-n_y}$	$e_{k-1}, \dots, e_{k-n_e}$
Finite Impulse Response	(FIR)	$\surd$		
AutoRegressive with eXogenous inputs	(ARX)	$\surd$	$\surd$	
AutoRegressive Moving Average with eXogenous inputs	(ARMAX)	$\surd$	$\surd$	$\surd$
Output Error	(OE)	$\surd$	$\hat{y}_{k-1}, \dots, \hat{y}_{k-n_y}$	

The prediction errors are defined by $e_{k-i} = y_{k-i} - \hat{y}_{k-i}$, where $\hat{y}_{k-i}$ is the model output at iteration $k - i$.

[3] Here, $\boldsymbol{x}_k$ should not be mistaken for the continuous state vector usually written $\boldsymbol{x}$ in system and control theory.

model is similar to the ARMAX model except that it does not consider past inputs. The OE model uses past predictions as regressors instead of the measured outputs. This makes it suitable for simulators since in this case output measurements are not available.

For nonlinear system identification, the common approach is to build the regression vector $\boldsymbol{x}_k$ from chosen regressors among the past inputs $\boldsymbol{u}_{k-1}$ and outputs $\boldsymbol{y}_{k-1}$ as for linear identification [241]. Following the standard nomenclature for linear models, nonlinear models are categorized by their regressors. The letter 'N' for "nonlinear" is simply added at the beginning of the name of the model. For example, a NARX model is a nonlinear autoregressive model with exogenous inputs which uses past inputs and past outputs as regressors as in the following example.

Example 2 (Autonomous nonlinear dynamical system)**.** *Figure 1.6 shows a sample trajectory for an autonomous (without external input) nonlinear system of equation*

$$y_k = \left(0.8 - 0.5\exp(-y_{k-1}^2)\right) y_{k-1} - \left(0.3 + 0.9\exp(-y_{k-1}^2)\right) y_{k-2} + 0.1\sin(\pi y_{k-1}). \tag{1.27}$$

This system has an unstable equilibrium point at the origin from which it spirals out toward an invariant closed curve (equivalent to a limit cycle in continuous time) as shown on the right of Fig.1.6. The regression surface defined by (1.27), plotted at the right of Fig. 1.6, is close to a hyperplane. However, the system exhibits a complex nonlinear and non-periodical behavior close to chaos[4]*. This system will be used in the experiments of section 3.3.1.*

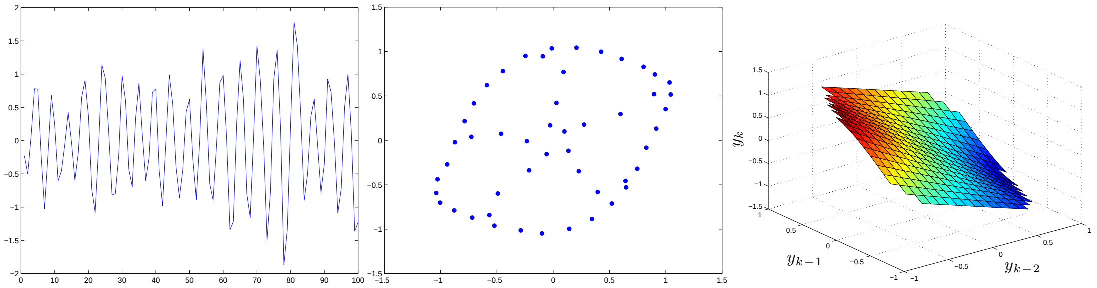


Figure 1.6: Nonlinear dynamical system. *Left:* Output trajectory y_k vs. time steps k for the initial conditions $y_0 = y_{-1} = 0.1$. *Middle:* Same trajectory in the regression space $\boldsymbol{x}_k = [y_{k-1},\ y_{k-2}]^T$. *Right:* regression surface of the system.

Performance measures

How to estimate the performance of a model depends on the form of the model. For equivalently parametrized systems and models, parametric identification tries to fit the model parameters to the data. Measuring the performance of a model thus amounts to measuring the error on the parameters (the true parameters must be known). On the other hand, unknown nonlinear systems are estimated by nonparametric models, in which case parameter estimates are either not available or irrelevant. In this context, two main performance measures can be defined.

One-step-ahead prediction error. As for nonlinear regression, the common approach for evaluating the learned model consists in measuring the output error on an independent data set $\widehat{S}$ (test set), thus obtaining an estimate of the generalization error, i.e. the ability of the model to make good predictions on unknown data. In this setting, the model predicts the output $\hat{y}_k = f(\boldsymbol{x}_k)$ given test regression vectors $\boldsymbol{x}_k$ and is evaluated on the basis of the one-step-ahead prediction error computed by (1.4). If the prediction error is measured by the squared loss function (1.14), then the MSE as defined in (1.15) is used.

[4] The Lyapunov exponents of the system are $\lambda_1 = -8.71 \times 10^{-5}$ and $\lambda_2 = -0.15$. A discrete-time system is said to be chaotic as soon as one of its Lyapunov exponents is positive.

n-step-ahead predicition error. Another quality measure, representative of the ability of the model to make long-term predictions, is the n-step-ahead predicition error. In this setting, the predicted outputs $\hat{y}_k$ are obtained from an estimated regression vector $\hat{\boldsymbol{x}}_k$ as $\hat{y}_k = f(\hat{\boldsymbol{x}}_k)$. The estimated regression vector is built from past predictions $\hat{y}_{k-j}$, $j = 1, \dots, n_y$, instead of the measurements y_{k-j}. Only the first predicted output $\hat{y}_1$ is computed from true measurements corresponding to the initial condition $\hat{\boldsymbol{x}}_1 = \boldsymbol{x}_1$. For a NARX model f, the n-step-ahead predicition error is computed by using f as a NOE model, leading to

$$\text{MSE}_n = \frac{1}{n}\sum_{k=1}^{n}(y_k - f(\hat{\boldsymbol{x}}_k))^2, \tag{1.28}$$

where n is the horizon of the test and $\hat{\boldsymbol{x}}_k = [\hat{y}_{k-1}, \dots, \hat{y}_{k-n_y},\ u_{k-1}, \dots, u_{k-n_u}]^T$.

1.4.3 Hybrid system identification

Classification as defined in section 1.2 amounts to estimating a discrete function taking values in a finite set of labels. On the other hand, system identification based on regression considers continuous and smooth function approximation. A particular problem in nonlinear system identification consists in non-smooth function approximation, involving both discrete and continuous variables. This problem is studied in the framework of *hybrid system identification*.

Hybrid dynamical systems are systems that switch between different modes with different dynamics, thus involving both discrete and continuous variables. A typical example taken from [16] is a gasoline engine: the power train, gas flow, and thermal dynamics are continuous processes, while the pistons have four modes of operation which can be described as a discrete event process or a finite state machine. These two heterogeneous processes interact tightly, as the timing of the transitions between two phases of the pistons is determined by the continuous dynamics of the power train, which, in turn, depends on the torque produced by each piston.

In the context of hybrid systems, two types of identification problems may arise depending on whether the switching sequence that generated the data is known or not. If it is, then the problem can be simply recast as multiple standard identification problems, each one using only the data for a given mode. However, in most cases this sequence is unknown and the problem becomes nontrivial. Indeed, the three major problems studied in the thesis are intrinsically mixed together in one single problem: estimate the modes to which the data points belong (classification) *and* the models (regression) governing the dynamics (system identification) for each mode. More details on the mathematical formulation of this problem will be given at the beginning of chapter 5.

1.5 Further reading

Feature selection

It must be noticed that the chapter does not present the problems of feature extraction (for pattern recognition) and of choosing the regressors (for regression and system identification) in the machine learning framework. In most of the thesis, it is assumed that the input vector $\boldsymbol{x}$ is available to the learning machine. Regarding this issue, the reader can refer to [105] and the book [106] for overviews of feature selection algorithms.

Non-convex learning algorithms

Neural networks. As universal approximators [118], neural networks have a long history in pattern recognition [24], regression [302] and system identification [197, 241]. Beside the RBFNN, the MultiLayer Perceptron (MLP) and the back-propagation algorithm [223] are perhaps the most common architecture and learning algorithm, respectively. The reader is referred to [115, 70] for a global review of neural networks.

Deep learning. The thesis focuses on kernel methods, which allow one to obtain a model by solving a convex optimization program. However, the reader must be aware that these methods do have limitations [18]. In particular, they are specifically developed for the approximation of smooth functions and are thus not sufficient for very complex tasks such as perception (vision, audition). Some authors recently started to promote the use of non-convex algorithms as a way to tackle more complex problems [20, 159]. This point of view is studied in the framework of *deep learning* and implemented by networks with deep architectures, including convolutional networks [160] and deep belief networks [117, 19].

System identification

Continuous-time identification. In this thesis, only discrete-time (DT) system identification is considered. The discussed methods can however be applied to continuous-time (CT) systems by considering that the data is provided by a sampling of the system. Recently, direct methods for the identification of CT models have regained some interest (see [89, 218, 90] and references therein). The main advantage of such direct methods is that they allow the interpretation of the parameter values, which often have a physical meaning. However, as most of this thesis is concerned with black box based modeling of nonlinear systems, in which the parameters are meaningless, CT model identification will not be studied.

Nonlinear system identification. Other models have been proposed for nonlinear system identification, see e.g. [198] for more details. Among these, Hammerstein [196, 76] and Wiener [303, 301] models consider the combination in series of a linear dynamical system and a static nonlinearity, respectively on the input and on the output. In addition, Hammerstein–Wiener models [10] combine both nonlinearities. For an overview of some other advanced black techniques, the reader can refer to [252].

Chapter 2

Support Vector Machines for classification

Abstract. *This chapter describes the Support Vector Machines (SVMs) for classification and provides an introduction to kernel methods. In particular, quadratic and linear programming formulations of the SVM training will be given before presenting the least squares SVM (LS-SVM). The chapter also provides a short discussion of the main features of statistical learning theory behind SVMs. As the incorporation of prior knowledge in SVMs is the key element that allows to increase the performance in many applications, the second part of the chapter reviews existing methods to tackle this problem and highlights the links and differences between them. The different types of prior knowledge are divided into two groups: class-invariance, that includes transformation and permutation invariances, but also invariance in an input domain, and knowledge on the data, that takes into account unlabeled data, unbalanced data or data with varying quality. The methods are then described and classified into three categories: sample methods based on the modification of the training data, kernel methods based on the modification of the kernel and optimization methods based on the modification of the problem formulation. A recent method, developed for support vector regression, considers prior knowledge about arbitrary regions of the input space. It is presented here when applied to the classification case. A discussion is then conducted to regroup sample and optimization methods under a regularization framework. Finally, the use of virtual samples to incorporate transformation-invariance is shown on the handwritten digit recognition problem.*

As an introduction to support vector machines (SVMs) and kernel methods, this chapter describes SVMs for classification, which is their original application. The quadratic and linear programming formulations are presented before the least squares SVM. All these algorithms aim at building a classifier on the basis of data only. Nonetheless, in real world applications, a certain amount of information on the problem is usually known beforehand. For instance, in character recognition, if an image is slightly translated or rotated it still represents the same character. This prior knowledge indicates that one should incorporate invariance to translations and rotations in the classifier. In many applications, domain knowledge is crucial to obtain state-of-the-art results and becomes necessary to further improve the performance of classifiers. Thus, following the grey box approach of the thesis, this chapter proposes a review of the state of the art regarding the incorporation of prior knowledge in SVMs for classification. The different forms of prior knowledge considered here are presented hierarchically and divided into two main groups: class-invariance and knowledge on the data. The first one includes invariances to transformations, to permutations and in domains of input space, whereas the second one involves knowledge about unlabeled data, the imbalance of the training set or the quality of the data. This review chooses to present general methods that can be used for different applications rather than to attempt to provide an exhaustive list of application specific prior knowledge with its practical implementation into SVMs. However, some interesting methods derived from an application specific point of view can still be used in other fields and thus deserve to be presented. The review focuses on the methods and reuses a categorization from the literature based on the component of the problem (the samples, the kernel or the optimization program), which is modified to include the prior knowledge, rather than the prior knowledge itself. A regularization framework is then used in section 2.5.2 to regroup the sample and optimization based methods. Prior knowledge for SVM in the context of regression will be discussed in chapter 4.

The first part of the chapter provides some fundamental insights into SVMs for the reader mostly interested in regression or system identification but unfamiliar with SVMs. The review presented in the second part (Sect. 2.3, 2.4 and 2.5 until page 41) is provided here as published in [149], thus including works that will not be used in the next chapters.

2.1 Support vector machines (SVM)

This section focuses on the classification problem where the aim is to determine the class of a pattern or sample, as defined in section 1.2.

A SVM classifier is built from a training set of N labeled samples $(\boldsymbol{x}_i, y_i)$, where $\boldsymbol{x}_i \in \mathbb{R}^p$ is the input vector corresponding to the ith sample labeled by $y_i \in \{-1, +1\}$ depending on its class (only binary problems are considered here). For the linear case, the machine implements the decision function

$$f(\boldsymbol{x}) = \text{sign }\left(\langle \boldsymbol{w}, \boldsymbol{x}\rangle + b\right), \tag{2.1}$$

with parameters $\boldsymbol{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$. This function determines on which side of the separating hyperplane ($\langle \boldsymbol{w}, \boldsymbol{x}\rangle + b = 0$) the sample $\boldsymbol{x}$ lies.

Support vector machines (SVMs) were first introduced as large margin classifiers [29, 275]. For a linearly separable training set (as defined in Sect. 1.2.2), the margin Δ of a linear classifier is defined as the minimum distance between the points of the two classes, measured perpendicularly to the separating hyperplane (see Figure 2.1). For the SVMs, this hyperplane is considered in its canonical form, meaning that its parameters $\boldsymbol{w}$ and b are normalized such that the training points closest to the hyperplane satisfy $|\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b| = 1$. In this case, the margin equals $2/\|\boldsymbol{w}\|$. Maximizing this margin is a way for a learning algorithm to control the complexity of the model, and select the *optimal separating hyperplane* amongst all the hyperplanes that separate the two classes of the training set. As will be seen in section 2.1.5, the control of the complexity (or capacity) is a key feature allowing the model to generalize on unknown samples.

The following presents the original quadratic programming formulation of SVMs, inspired by [275] and [62]. The linear programming version of SVMs will be introduced following [179], using matrix notations, which are better suited in this case.

2.1.1 Quadratic programming (QP-SVM)

In its original form, the SVM learning leads to a quadratic program which is a convex constrained optimization problem and thus has a unique solution. This is a large advantage in comparison to other learning algorithms such as the back-propagation for neural networks [223, 24]. The SVM problem can be stated as follows: find the parameters $\boldsymbol{w}$ (also called the weights) and b that maximize the margin while ensuring that the training samples are well classified. This can be written as the quadratic programming (QP) optimization problem

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 \tag{2.2}$$

$$\text{s.t. } y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \geq 1, \quad i = 1, \dots, N, \tag{2.3}$$

whose solution corresponds to the saddle point of the primal Lagrangian

$$L_P = \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_{i=1}^{N} \alpha_i[(y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) - 1], \tag{2.4}$$

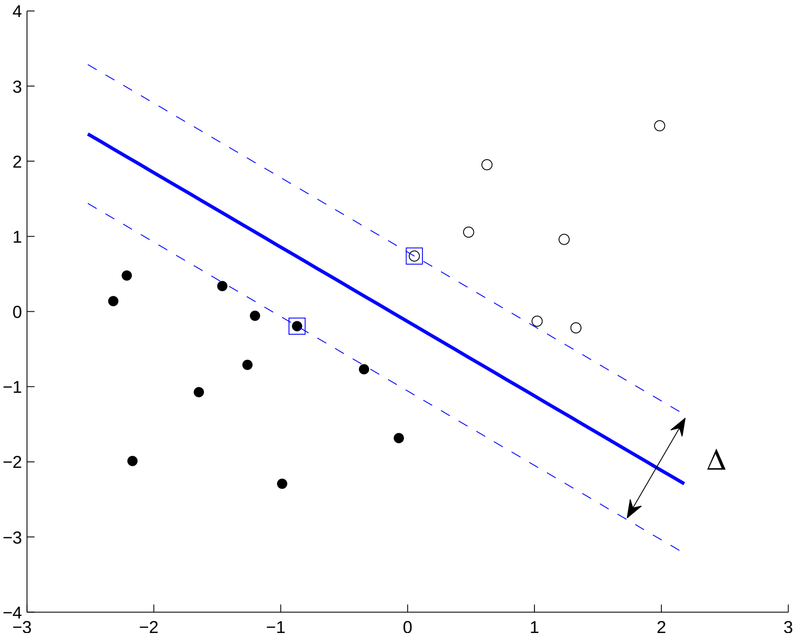


Figure 2.1: Large margin classifier separating the white points from the black points with an optimal canonical hyperplane (—). Squares are plotted around the support vectors (SVs).

where the $\alpha_i \geq 0$ are the Lagrange multipliers. The minimum of the Lagrangian L_P is obtained by making its derivatives w.r.t. all the variables vanish to zero

$$\begin{aligned} \frac{\partial L_P}{\partial \boldsymbol{w}} &= \boldsymbol{w} - \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i = 0 && (2.5)\\ \frac{\partial L_P}{\partial b} &= \sum_{i=1}^{N} \alpha_i y_i = 0. && (2.6) \end{aligned}$$

This allows to remove the primal variables from the formulation (2.4) to yield the dual Lagrangian

$$L_D = \inf_{\boldsymbol{w},b} L_P = \frac{1}{2} \sum_{i=1,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i=1,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i=1}^{N} \alpha_i y_i b + \sum_{i=1}^{N} \alpha_i. \quad (2.7)$$

As the original problem is convex, strong duality implies that the problem can be solved by maximizing the dual Lagrangian w.r.t. the dual variables α_i as

$$\begin{aligned} \max_{\alpha_i \geq 0} \quad & L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0, \end{aligned} \qquad (2.8)$$

where (2.6) is introduced as a constraint and used to remove the term with b in (2.7). Solving this quadratic program yields the parameters α_i. The weight vector $\boldsymbol{w}$ is computed by (2.5) while b can be recovered from the original constraints (2.3) that are equalities for the support vectors $\boldsymbol{x}_i$. The resulting decision function is

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{\alpha_i > 0} y_i \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + b \right), \qquad (2.9)$$

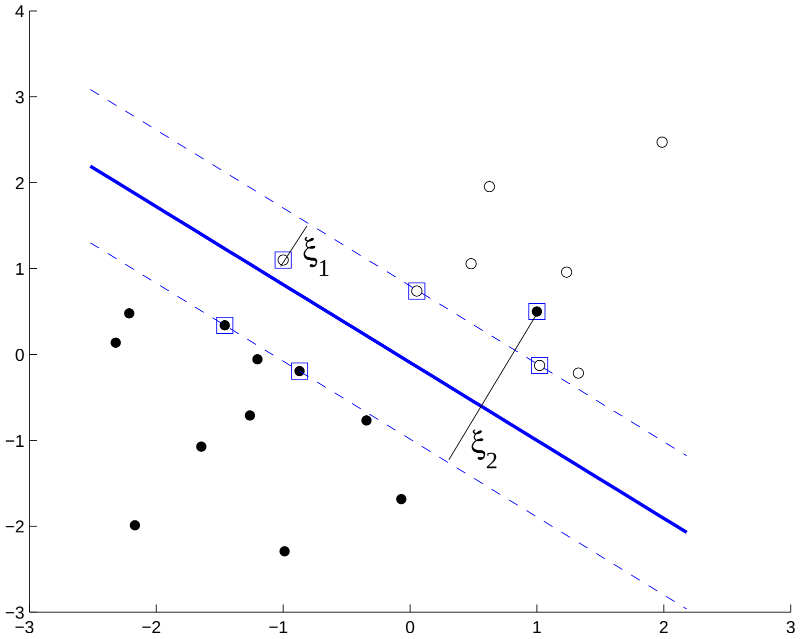


Figure 2.2: Soft margin SVM separating the white points from the black points. The support vectors (SVs) are represented by squares. The slack variables ξ_1 and ξ_2 are assigned respectively to a point inside the margin and an error sample.

where the $\boldsymbol{x}_i$ are the support vectors (SVs), i.e. with non-zero corresponding Lagrange multipliers α_i. The SVs are the training patterns that lie on the margin boundaries. An advantage of this algorithm is its sparsity since only a small subset of the training samples are finally retained for the classifier.

Relaxing the constraints: soft margin

The soft margin hyperplane is used to deal with nonlinearly separable data. A set of slack variables ξ_i is introduced to allow errors and points inside the margin during the training (see Figure 2.2). A hyperparameter C is used to tune the trade-off between the amount of accepted errors and the maximization of the margin

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi_i\geq 0} \ & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \ & y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle + b) \geq 1-\xi_i, \quad i=1,\ldots,N. \end{aligned} \tag{2.10}$$

This new formulation leads to the same dual problem (2.8) but with the addition of an upper bound on the Lagrange multipliers

$$\begin{aligned} \max_{\alpha_i\geq 0} \quad & L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{N}\alpha_i\alpha_j y_i y_j\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle \\ \text{s.t.} \quad & \sum_{i=1}^{N}\alpha_i y_i = 0 \\ & 0\leq \alpha_i \leq C, \quad i = 1\ldots N. \end{aligned} \tag{2.11}$$

The tuning of the hyperparameter C is a delicate task. A larger C implies a smaller training error, but also a smaller margin and possibly a lower generalization performance. On the other hand, a small C gives more weight to the regularization term $\|\boldsymbol{w}\|_2^2$ and may lead to a better classifier when the training data are sparse or mislabeled, but may also diminish the influence of relevant data. Beside standard techniques described in section 1.1, other methods for the tuning of the SVM hyperparameters have been proposed, including evolutionary tuning [84] and gradient-based approaches [44]. These are usually based on efficiently computable approximations of the LOO estimate [73]. A more recent approach, developed in [112] and [172], consists in computing the entire regularization paths of the optimization problem, yielding the parameters α_i and b for all values of C, with the same computational cost as solving one single SVM problem. Testing the resulting models on validation data can then be very efficiently obtained.

2.1.2 Linear programming (LP-SVM)

The training of SVMs can also result in a linear program. Following the approach of [179], the weight vector $\boldsymbol{w}$ of the classifier (2.1) is assumed to be $\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{D}\boldsymbol{\alpha}$, where $\boldsymbol{X} \in \mathbb{R}^{N\times p}$ is the observation matrix defined as $\boldsymbol{X} = [\boldsymbol{x}_1\ \boldsymbol{x}_2\ \dots\ \boldsymbol{x}_N]^T$, $\boldsymbol{\alpha} \in \mathbb{R}^N$ is a new set of parameters, and $\boldsymbol{D}$ is a diagonal matrix containing the training labels and defined by $\boldsymbol{D} = diag(\boldsymbol{y})$. The training is then performed by minimizing the ℓ_1-norm of these parameters instead of the ℓ_2-norm of the weights $\boldsymbol{w}$ as in (2.10). This leads to the soft margin SVM problem

$$\begin{aligned} &\min_{\boldsymbol{\alpha},b,\xi_i\geq 0} \|\boldsymbol{\alpha}\|_1 + C\sum_{i=1}^N \xi_i \\ &\text{s.t.}\ y_i(\boldsymbol{x}_i^T\boldsymbol{X}^T\boldsymbol{D}\boldsymbol{\alpha} + b) \geq 1 - \xi_i, \quad i = 1,\dots,N. \end{aligned} \tag{2.12}$$

In order to yield a linear program, a vector $\boldsymbol{a}$ of N positive variables bounding the ℓ_1-norm of the parameters is used. Then, training a linear programming SVM (LP-SVM) can be written in matrix form as

$$\begin{array}{rcl} \min\limits_{\boldsymbol{\alpha},b,\boldsymbol{\xi}\geq\boldsymbol{0},\boldsymbol{a}\geq\boldsymbol{0}} & \boldsymbol{1}^T\boldsymbol{a} + C\boldsymbol{1}^T\boldsymbol{\xi} & \\ \text{s.t.} & \boldsymbol{D}(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{D}\boldsymbol{\alpha} + b\boldsymbol{1}) & \geq \boldsymbol{1} - \boldsymbol{\xi} \\ -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a}, \end{array} \tag{2.13}$$

where $\boldsymbol{\xi} = [\xi_1\ \xi_2\ \dots\ \xi_N]^T$ and $\boldsymbol{y} = [y_1\ y_2\ \dots\ y_N]^T$. The linear program (2.13) can be solved directly. Here, the dual formulation does not increase the efficiency of the optimizer.

The resulting decision function is given by

$$f(\boldsymbol{x}) = \operatorname{sign}\left(\boldsymbol{x}^T\boldsymbol{X}^T\boldsymbol{\alpha} + b\right) = \operatorname{sign}\left(\boldsymbol{\alpha}^T\boldsymbol{X}\boldsymbol{x} + b\right). \tag{2.14}$$

Here, the sparsity is enforced by the minimization of the ℓ_1-norm of the parameters $\boldsymbol{\alpha}$ which makes some α_i vanish to zero. This can be related to the Least Absolute Shrinkage and Selection Operator (LASSO) developed in [259] and also used in [310]. Variable selection by shrinkage will be studied for hybrid system identification in chapter 5.

2.1.3 Nonlinear Support Vector Machines

As seen in section 1.2.3, a common approach to nonlinear classification is to map the data into a higher dimensional feature space F, where the problem becomes linear. To overcome the curse of dimensionality due to the dimension of F, the kernel trick, depicted below, is used in SVMs.

Nonlinear mapping and the kernel trick

Consider classifying the data according to a separating hyperplane in F by

$$f(\boldsymbol{x}) = \operatorname{sign}\ \left(\langle\boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x})\rangle_F + b\right), \tag{2.15}$$

where $\boldsymbol{w} \in F$ and $\boldsymbol{\Phi}(\boldsymbol{x})$ is the image of the input vector $\boldsymbol{x}$ in feature space given by the nonlinear mapping $\Phi : \boldsymbol{x} \mapsto \boldsymbol{\Phi}(\boldsymbol{x})$.

As it can be seen in (2.9) and (2.11), the input vectors are only involved through their inner product. Thus, to map the data into a high dimensional feature space, one does not need to

Table 2.1: Most common kernel functions.

Kernel	$k(\boldsymbol{x}, \boldsymbol{z})$	Hyperparameters
linear	$\langle \boldsymbol{x}, \boldsymbol{z} \rangle$	
polynomial	$(\gamma\langle \boldsymbol{x}, \boldsymbol{z} \rangle + c)^d$	$d \in \mathbb{N},\ c \geq 0$
Gaussian RBF	$\exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{z}\|^2}{2\sigma^2}\right)$	$\sigma > 0$
sigmoidal	$\tanh(\gamma\langle \boldsymbol{x}, \boldsymbol{z} \rangle + c)$	$\gamma, c \geq 0$

Note that the sigmoidal kernel requires $1 - 2\gamma\langle \boldsymbol{x}, \boldsymbol{z} \rangle \tanh(\gamma\langle \boldsymbol{x}, \boldsymbol{z} \rangle + c) \geq 0$ to satisfy Mercer's condition.

consider the mapping Φ in explicit form. One only has to calculate the inner products of the images of the points $\boldsymbol{\Phi}(\boldsymbol{x})$ in the feature space. This is the *kernel trick* which replaces the inner products between images of points by a *kernel function*

$$k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{\Phi}(\boldsymbol{x}), \boldsymbol{\Phi}(\boldsymbol{z}) \rangle, \tag{2.16}$$

which is symmetric by definition.

Example 3 (Nonlinear classification (continued from example 1 on page 5))**.** *Consider the example of Sect. 1.2.3, where the data are mapped by* $\Phi : (x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$*. For two points* $\boldsymbol{a} = (a_1, a_2)$ *and* $\boldsymbol{b} = (b_1, b_2)$*, we have*

$$\langle \boldsymbol{\Phi}(\boldsymbol{a}), \boldsymbol{\Phi}(\boldsymbol{b}) \rangle = \langle (a_1^2, \sqrt{2}a_1a_2, a_2^2), (b_1^2, \sqrt{2}b_1b_2, b_2^2) \rangle = (a_1b_1)^2 + 2a_1b_1a_2b_2 + (a_2b_2)^2 = \langle \boldsymbol{a}, \boldsymbol{b} \rangle^2.$$

Thus the inner product between images of points mapped by Φ *is equivalently obtained by the polynomial kernel function* $k(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle^2$*, computed in the original input space.*

Mercer's condition. To be an admissible kernel function, k must be equivalent to an inner product in a feature space. This is the case for any k satisfying Mercer's condition, which can be summarized by the inequality [244]

$$\int_{\mathcal{X}\times\mathcal{X}} k(\boldsymbol{x}, \boldsymbol{z}) f(\boldsymbol{x}) f(\boldsymbol{z})\, d\boldsymbol{x} d\boldsymbol{z} \geq 0,\ \forall f \in L_2(\mathcal{X}), \tag{2.17}$$

where $L_2(\mathcal{X})$ is the space of square-integrable functions over $\mathcal{X}$ (Lebesgue space).

As shown in [62], for a particular training set, the Mercer's condition can also be verified by studying the positive definiteness (non-negativity of the eigenvalues) of the kernel matrix $\boldsymbol{K}$ defined from the samples $\boldsymbol{x}_i$ by

$$\boldsymbol{K} = \{k(\boldsymbol{x}_i, \boldsymbol{x}_j)\}_{i,j=1}^N . \tag{2.18}$$

Moreover, since $\boldsymbol{K}$ is symmetric, it is sufficient to study its positive semi-definiteness.

Most common kernels. In practice, the most common kernels are the polynomial, Gaussian Radial Basis Function (RBF) and sigmoidal kernels. A summary is given in Table 2.1. It can be noted that the sigmoidal kernel does not satisfy Mercer's conditions in the general case. It has however been successfully used in practice [227]. This can be explained by considering that a function k can violate Mercer's condition in general but still lead to a positive-definite kernel matrix for the given training data. Moreover, if the kernel matrix has only a few negative eigenvalues which are small, it is always possible to add a small multiple of a positive-definite matrix (given for instance by another kernel) to yield a positive-definite matrix [227].

More details on kernel functions such as how to build admissible kernels from other kernels can be found e.g. in [62].

Nonlinear QP-SVM

The kernel trick allows the construction of a separating surface given by a nonlinear function $h(\boldsymbol{x})$ in the input space, which is linear in the feature space via an implicit nonlinear mapping. Training a nonlinear QP-SVM amounts to training a linear QP-SVM in feature space, i.e. to replacing the samples $\boldsymbol{x}_i$ by their images $\boldsymbol{\Phi}(\boldsymbol{x}_i)$ in (2.11). After changing the inner products $\langle \boldsymbol{\Phi}(\boldsymbol{x}_i), \boldsymbol{\Phi}(\boldsymbol{x}_j)\rangle$ by the kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, the dual problem becomes

$$\begin{aligned} \max_{\alpha_i \geq 0} \quad & L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{N} \alpha_i \alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1 \dots N, \end{aligned} \tag{2.19}$$

and the final classifier is now given by

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{\alpha_i > 0} y_i \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b \right). \tag{2.20}$$

Nonlinear LP-SVM

To train a nonlinear LP-SVM, one only has to replace the product $\boldsymbol{X}\boldsymbol{X}^T$, containing all the inner products between sample vectors $\boldsymbol{x}_i$, by the kernel matrix $\boldsymbol{K}$ in (2.13). This leads to the linear program

$$\begin{array}{rccl} & \displaystyle\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi} \geq \boldsymbol{0}, \boldsymbol{a} \geq \boldsymbol{0}} & \boldsymbol{1}^T \boldsymbol{a} + C \boldsymbol{1}^T \boldsymbol{\xi} & \\ \text{s.t.} & & \boldsymbol{D}(\boldsymbol{K}\boldsymbol{D}\boldsymbol{\alpha} + b\boldsymbol{1}) & \geq \boldsymbol{1} - \boldsymbol{\xi} \\ & -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a}, \end{array} \tag{2.21}$$

and the final classifier

$$f(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T) \boldsymbol{D}\boldsymbol{\alpha} + b \right), \tag{2.22}$$

where $\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T) = [k(\boldsymbol{x}, \boldsymbol{x}_1) \ k(\boldsymbol{x}, \boldsymbol{x}_2) \ \dots \ k(\boldsymbol{x}, \boldsymbol{x}_N)]$, as defined in the notations, page XXI.

In this formulation, no assumption on the symmetry or positive definiteness of the kernel matrix $\boldsymbol{K}$ is needed [179]. The LP-SVM can thus work with a larger class of kernel functions than the QP-SVM. This comes from the fact that the problem does not involve the ℓ_2-norm $\|\boldsymbol{w}\|_2$ corresponding to an inner product in the feature space. Thus, it is not required for the kernel function to implicitly define an inner product space.

2.1.4 Algorithmic implementation and optimization

Many general optimization softwares such as CPLEX, LOQO, Matlab linprog and quadprog are capable of solving the linear and quadratic programs derived from SVMs. Nonetheless, the scale of the problems led people to develop specific methods such as chunking [140], decomposition [203] or its extreme case known as the Sequential Minimal Optimization (SMO) algorithm [207], of which modifications have been proposed [141]. Besides, a new trend towards learning in the primal space has also recently emerged [41]. Many algorithms and softwares, usually available on the Internet, have been developed in the last years to speed up the training time or to deal with large data sets such as SVM^{light} [127], libSVM [40, 80] and libsvmTL [221], HeroSVM [68, 69], Core Vector Machine (CVM) [269], SimpleSVM [289, 170, 171]. An overview of most of these algorithms is given in [30]. Other Matlab toolboxes can be found such as the SVM and Kernel Methods Toolbox [35], S. Gunn's toolbox [104] or the Spider [298]. Note, however, that this is a non-exhaustive list.

2.1.5 Elements of statistical learning theory

The following gives a quick overview of the main ideas and concepts from statistical learning theory. For a complete description of the theory, the reader is referred to the books [275, 276] or [62]. The aim here is rather to provide some arguments to justify the use of SVMs with few key features.

Empirical vs. expected risk

The global aim of a learning task is to minimize the expected risk $R[f]$ (1.2) as defined in section 1.2. However, as $p(\boldsymbol{x}, y)$ is unknown, this quantity cannot be computed. In practice an inductive principle, defining how an algorithm should build a model from data, must be chosen. The most straightforward approach is to choose the Empirical Risk Minimization (ERM) principle, which proposes to retain the best function on the training data as the final model. This amounts to approximating the expected risk by the empirical risk

$$E_S[f] = \frac{1}{N}\sum_{i=1}^{N}[f(\boldsymbol{x}_i) \neq y_i], \tag{2.23}$$

as in (1.3) using (1.5). But minimizing the empirical risk does not lead to a minimum of the expected risk. This phenomenon is known as overfitting (see section 1.2.3).

Without constraints on the class $\mathcal{F}$ of admissible functions f, the empirical risk $E[f]$ can always be made equal to zero, without implying that f can generalize, i.e. lead to a low risk $R[f]$ (think a machine that learns "by heart"). Statistical learning theory studies in which cases the ERM principle is consistent, i.e. when it is sufficient to minimize the training error (2.23).

Nontrivial consistency and uniform convergence

The classical definition of consistency for the Empirical Risk Minimization (ERM) principle is

$$\forall \varepsilon, \quad \lim_{N \to +\infty} P\{|R[f_S] - E_S[f_S]| \geq \varepsilon\} = 0, \tag{2.24}$$

where for a training set S of size N, $f_S \in \mathcal{F}$ minimizes the empirical risk $E_S[f_S]$. This condition of consistency ensures that the empirical risk converges in probability to the true risk.

One of the goals of statistical learning theory is to obtain conditions of consistency in terms of general characteristics of the class of admissible functions $\mathcal{F}$. Unfortunately, trivial cases, where consistency (2.24) holds but the ERM does not yield a function able to generalize, can occur[1] in the presence of particular functions in the set $\mathcal{F}$ [275].

In order to study consistency on the basis of general properties of $\mathcal{F}$ only, Vapnik defines the notion of *nontrivial consistency*, based on consistency for subsets of $\mathcal{F}$ after removing functions with the smallest values of the risks. This nontrivial consistency can be shown to be equivalent to the uniform convergence

$$\forall \varepsilon, \quad \lim_{N \to +\infty} P\{\sup_{f \in \mathcal{F}}(R[f] - E_S[f_S]) \geq \varepsilon\} = 0. \tag{2.26}$$

This means that the expected risk $R[f]$ of the worst function of $\mathcal{F}$ (the one leading to the maximal $R[f]$) must converge to the minimal empirical risk for $N \to +\infty$. Thus statistical learning theory is a *worst-case* theory. However, Vapnik showed that this condition is necessary and sufficient: any theory of consistency of the ERM principle *has to be* a worst-case theory.

Generalization error bounds

Statistical learning theory allows one to derive bounds on the risk $R[f]$ in order to guarantee a certain performance for a given learning machine.

The study of the asymptotic rate of uniform convergence (2.26) leads to finite-sample bounds on the generalization error: a fast convergence rate implies a tighter bound for a particular N. In

[1] *Example of a trivial case of consistency.* Consider a set $\mathcal{F}$ for which consistency (2.24) does not hold. Define the set $\tilde{\mathcal{F}} = \mathcal{F} \bigcup \{\tilde{f}\}$, containing an additional function $\tilde{f}$ satisfying

$$\inf_{f \in \mathcal{F}} E_S[f] > E_S[\tilde{f}], \ \forall S. \tag{2.25}$$

Applying the ERM principle to $\tilde{\mathcal{F}}$, the minimum of the empirical risk $E_S[f]$ will always be obtained for $\tilde{f}$, which also minimizes the expected risk $R[f]$, for $f \in \tilde{\mathcal{F}}$ (since (2.25) is verified for all S, including when $N \to +\infty$). Thus using the classical definition, the ERM principle is consistent for the class of functions $\tilde{\mathcal{F}}$, but always yields the arbitrary function $\tilde{f}$ unable to generalize. This trivial case of consistency depends on whether the set of admissible functions contains a function leading to a lower bound on $E_S[f]$ for all S.

this setting, it is required to bound $R[f]$ for the worst f and thus for all f in $\mathcal{F}$. It is clear that such a bound cannot be obtained for an unconstrained function class $\mathcal{F}$. The capacity of $\mathcal{F}$ must be finite for the ERM to be nontrivially consistent.

VC dimension. Measuring the capacity of a class of functions $\mathcal{F}$ thus becomes crucial. One of such measures is the Vapnik Chervonenkis (VC) dimension h. For binary classification, h is the maximal number of points which can be separated into two classes in all possible 2^h ways by using functions $f \in \mathcal{F}$ [275]. Using the VC dimension, the expected risk $R[f]$ can be bounded with probability at least $1 - \delta$ by

$$R[f] \leq E_S[f] + \sqrt{\frac{h(\ln(2N/h) + 1) - \ln(\delta/4)}{N}}. \tag{2.27}$$

This distribution-independent bound shows that both the capacity of $\mathcal{F}$ and the empirical risk should be minimized in order to guarantee a low risk $R[f]$. Moreover, the VC dimension of a set of functions can be both larger or smaller than the number of parameters of the functions [275]. It is thus possible for a learning machine to generalize well on the basis of a set of functions containing a huge number of parameters (think of a linear function in a very high dimensional space) but possessing a low VC dimension.

Margin-based bounds. The VC dimension of the class of separating hyperplanes can be bounded by a function of the margin Δ [275]. Generalization error bounds can thus be expressed in terms of the margin. For instance, the following bound holds with probability $1 - \delta$ [62]:

$$R[f] \leq \frac{c}{N}\left(\frac{R^2 + \|\boldsymbol{\xi}\|_1^2 \log_2(1/\Delta)}{\Delta^2} \log_2^2 N + \log_2 \frac{1}{\delta}\right). \tag{2.28}$$

where c is a constant, $\boldsymbol{\xi}$ is the slack vector w.r.t. the margin Δ and R is the radius of the minimum enclosing ball of the data in feature space, i.e. for all $\boldsymbol{x}$, we have $\|\boldsymbol{\Phi}(\boldsymbol{x})\| = \sqrt{\langle \boldsymbol{\Phi}(\boldsymbol{x}), \boldsymbol{\Phi}(\boldsymbol{x})\rangle} \leq R$. This bound clearly shows that maximizing the margin leads to better generalization.

Empirical vs. structural risk minimization

When the number of data available for training is large, the ERM principle is well suited and yields a low bound on the generalization error (2.27). However, when the number of data is low (typically $N/h < 20$), the term

$$c(N, h, \delta) = \sqrt{\frac{h(\ln(2N/h) + 1) - \ln(\delta/4)}{N}}, \tag{2.29}$$

becomes large and minimizing $E_S[f]$ no longer ensures a low generalization error $R[f]$. On the contrary, the Structural Risk Minimization (SRM) principle proposes a strategy to minimize both terms in the bound by controlling the function class capacity.

Consider a nested sequence of function classes

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \cdots \subset \mathcal{F}_i \subset \cdots \subset \mathcal{F}_M, \tag{2.30}$$

and the functions $f_i \in \mathcal{F}_i$ with minimum number of errors $E_S[f_i]$ on the training set S. Their number of errors satisfy

$$E_S[f_1] \geq E_S[f_2] \geq \cdots \geq E_S[f_i] \geq \cdots \geq E_S[f_M], \tag{2.31}$$

while their VC dimension h_i form a non-decreasing sequence

$$h_1 \leq h_2 \leq \cdots \leq h_i \leq \cdots \leq h_M. \tag{2.32}$$

Since the term $c(N, h, \delta)$ increases with h, this measure of the capacity of the function class must be controlled. The SRM strategy considers finding the functions f_i for each $\mathcal{F}_i$ and then choosing the one that minimizes the bound (2.27). It can be seen as multiple-objective minimization of both the empirical risk and the capacity.

Learning algorithms. As seen above, the SRM principle shows that both minimization of the empirical risk and of the capacity of a learning machine are required to minimize the generalization error. Based on this consideration two types of approaches can be devised [275]:

- Keep the capacity of the machine fixed and minimize the empirical risk *(implemented by neural networks).*
- Keep the value of the empirical risk fixed and minimize the capacity *(implemented by support vector machines).*

Conclusion

As stated by Vapnik [277]:

> *Two theoretical results made the construction of SVMs possible:*
>
> - *The generalization ability of a learning machine depends on the capacity of a set of functions (in particular on the VC dimension of a set of functions) rather than the dimensionality of the space. Therefore, [...] a function that describes the data well and belongs to a set of functions with low capacity will generalize well regardless of the dimensionality of the space.*
> - *To construct a hyperplane one only needs to evaluate the inner product between vectors of the training data. Since in Hilbert space the general form of the inner product has a kernel representation, the evaluation of the inner product also does not depend on the dimensionality of the space.*

These theoretical results allow one to build linear classifiers with low capacity in very high dimensional feature space while avoiding the curse of dimensionality. The true strength of this is that training a linear classifier with capacity control in feature space can be formulated as a convex optimization problem, which has a unique solution and for which efficient solvers exist even for very large data sets. Thus SVMs benefit from a *sound theoretical background that can be efficiently implemented with a low computational complexity.*

2.1.6 Applications

The success of SVMs is largely due to their generality of application and their ability to yield state-of-the-art results with little effort and without domain knowledge in many applications[2] including handwritten digit recognition (without prior knowledge on the feature extraction) [162], difficult problems with high dimensional feature representations such as 3D object recognition [209], text categorization [126] or color-based image recognition [42], and bioinformatics applications such as protein classification [122] or analysis of microarray gene expression data [33]. The applicability of SVMs and kernel methods is mainly limited to vectorial inputs for which kernel functions can be easily defined based on similarity measures. However, kernels between structured inputs such as graphs [145, 139], automata [57] or between dynamical systems [287, 37, 290] have already been proposed.

SVMs are well suited when little knowledge on the task at hand is available. However, it is clear that classifiers specifically developed with domain knowledge are more likely to outperform the others. In section 2.4, a review of the methods that include such knowledge in SVMs will be given.

2.2 Least Squares Support Vector Machines (LS-SVM)

The Least Squares SVM (LS-SVM) [251][3] are a widely known simplification of the SVM, allowing to obtain the model by solving a system of linear equations instead of a linear or quadratic minimization program.

[2] For a continuously growing list of SVM applications, the reader may refer to Isabelle Guyon's webpage on the subject at http://www.clopinet.com/isabelle/Projects/SVM/applist.html.

[3] LS-SVM classifiers were originally presented in [253].

2.2.1 Algorithm

The following describes the algorithm for nonlinear classifiers $f(\boldsymbol{x}) = \text{sign}(h(\boldsymbol{x}))$, where $h(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x})\rangle + b$. The linear case is recovered by replacing the images $\boldsymbol{\Phi}(\boldsymbol{x})$ by the points $\boldsymbol{x}$.

The main idea is to consider the labels y_i as real-valued targets instead of considering only their sign. Training a LS-SVM classifier amounts to force the real-valued output of the classifier $h(\boldsymbol{x}_i)$ to equal $y_i \in \{-1, +1\}$ for all training samples instead of imposing that the product $y_i h(\boldsymbol{x}_i)$ is greater than 1 as for SVMs in (2.3). Thus the problem is reformulated with equality constraints instead of inequalities. Relaxing these constraints with slack variables e_i (not necessarily positive) and minimizing a squared loss function on these e_i leads to

$$\min_{\boldsymbol{w},b,e_i} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\frac{1}{2}\sum_{i=1}^{N} e_i^2 \tag{2.33}$$

$$\text{s.t. } y_i(\langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b) = 1 - e_i,\ i = 1, \dots, N. \tag{2.34}$$

The solution of this problem corresponds to the saddle point of the primal Lagrangian

$$L_P = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\frac{1}{2}\sum_{i=1}^{N} e_i^2 - \sum_{i=1}^{N}\alpha_i[(y_i(\langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b) - 1 + e_i], \tag{2.35}$$

where the $\alpha_i \in \mathbb{R}$ are the Lagrange multipliers. The optimality conditions obtained by making the derivatives w.r.t. all the variables vanish to zero are thus written as

$$\frac{\partial L_P}{\partial \boldsymbol{w}} = \boldsymbol{0} \Rightarrow\ \boldsymbol{w} = \sum_{i=1}^{N}\alpha_i y_i \boldsymbol{\Phi}(\boldsymbol{x}_i) \tag{2.36}$$

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow\ \sum_{i=1}^{N}\alpha_i y_i = 0 \tag{2.37}$$

$$\frac{\partial L_P}{\partial e_i} = 0 \Rightarrow\ e_i = \frac{1}{C}\alpha_i,\ i = 1, \dots, N. \tag{2.38}$$

Replacing $\boldsymbol{w}$ by (2.36) and e_i by (2.38) in (2.34) leads to

$$y_i(\sum_{j=1}^{N}\alpha_j y_j \langle \boldsymbol{\Phi}(\boldsymbol{x}_j), \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b) + \frac{1}{C}\alpha_i = 1,\ i = 1, \dots, N. \tag{2.39}$$

Then defining $\boldsymbol{Z} = [\boldsymbol{\Phi}(\boldsymbol{x}_1)y_1\ \boldsymbol{\Phi}(\boldsymbol{x}_2)y_2\ \dots\ \boldsymbol{\Phi}(\boldsymbol{x}_N)y_N]^T$ and $\boldsymbol{\Omega} = \boldsymbol{Z}^T\boldsymbol{Z}$ allows to write (2.37) and (2.39) as the linear system of equations

$$\begin{bmatrix} 0 & \boldsymbol{y}^T \\ \boldsymbol{y} & \boldsymbol{\Omega} + \frac{1}{C}\boldsymbol{I} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{1} \end{bmatrix}, \tag{2.40}$$

to be solved in $\boldsymbol{\alpha}$ and b. Solving this problem is less demanding than solving a linear or quadratic program as it basically amounts to inverting a matrix.

As with standard SVM, the final classifier $f(\boldsymbol{x}) = \text{sign}\left(\langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x})\rangle + b\right)$ is recovered from (2.36) as

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{N}\alpha_i y_i \langle \boldsymbol{\Phi}(\boldsymbol{x}_i), \boldsymbol{\Phi}(\boldsymbol{x})\rangle + b\right) = \text{sign}\left(\sum_{i=1}^{N}\alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b\right). \tag{2.41}$$

Remark 4. *By using the kernel trick and the matrix notations of the LP-SVM (Sect. 2.1.2), the matrix* $\boldsymbol{\Omega}$*, of components* $\Omega_{ij} = y_i y_j \langle \boldsymbol{\Phi}(\boldsymbol{x})_i, \boldsymbol{\Phi}(\boldsymbol{x})_j\rangle$ *can also be obtained by* $\boldsymbol{\Omega} = \boldsymbol{DKD}$.

2.2.2 Discussion

Pros and cons. The LS-SVM is a computationally advantageous simplification of the original SVM, leading to a linear system of equations.

In addition, the LS-SVM retains some of properties of the original SVM:

- *Uniqueness of the solution.* Since the system (2.40) is square, there is a unique LS-SVM solution as long as the matrix in the left-hand side has full rank.

- *Kernel method.* The use of kernel functions allows to deal easily with the nonlinear case as for SVM.

However, the computational benefits are payed for by other shortcomings:

- *Sparsity.* The sparsity of the original SVM comes from the margin acting as a threshold for the selection of support vectors among the training samples. In LS-SVM, sparsity is lost and the classifier (2.41) is computed on the basis of all the training samples as support vectors. However, support vectors can be pruned in an additional step to yield sparse classifiers. This will be presented in the context of regression in section 3.2.2.

- *Robustness.* As the standard least squares method for regression, the LS-SVM is not robust to outliers (e.g. mislabeled samples), which have a large influence on the objective function through the squared error terms e_i^2.

Related approaches. Note that the LS-SVM is strongly related to Fisher discriminant analysis in feature space [251] and can also be interpreted in a Bayesian framework [273].

Benchmarks. Results of an extensive experimental study of the LS-SVM on 20 benchmark data sets is reported in [272]. These show that LS-SVM can lead to performances comparable to standard SVM approaches.

2.3 Prior knowledge for classification

We now turn to the case where additional knowledge about the problem is available before training. This section starts by giving a definition of prior knowledge as considered in the chapter. The different types of prior knowledge encountered in pattern recognition are then regrouped under two main categories: class-invariance and knowledge on the data. Only the types of prior knowledge for which methods allow their incorporation in SVMs are discussed in this review. The next section provides the description and categorization of these methods together with a tree-like diagram summarizing the types of prior knowledge that can be included.

2.3.1 Definition

In this review, *prior knowledge* is defined as in [232] and refers to all information about the problem available in addition to the training data. Determining a model from a finite set of samples without prior knowledge is an ill-posed problem, in the sense that, for instance, a unique model may not exist. Many classifiers incorporate the general smoothness assumption that a test pattern similar to one of the training samples tends to be assigned to the same class. Also, choosing the soft margin version of SVMs can be seen as a use of prior knowledge on the non-separability of the data or the presence of outliers and noise in the training set. However, in both cases, these assumptions are intrinsically made by the support vector learning and are thus excluded from the definition of prior knowledge in the remainder of the thesis. In machine learning, the importance of prior knowledge can be seen from the No Free Lunch theorem [305]. This theorem states that all the algorithms perform the same when averaged over the different problems and thus implies that to gain in performance one must use a specialized algorithm that includes some prior knowledge about the problem at hand.

2.3.2 Class-invariance

A very common type of prior knowledge in pattern recognition is the invariance of the class (or the output of the classifier) to a transformation of the input pattern. Throughout this thesis, this type of knowledge will be referred to as *transformation-invariance.* Incorporating the invariance to a transformation $T_\theta : \boldsymbol{x} \mapsto T_\theta \boldsymbol{x}$, parametrized in θ, into a classifier of output $f(\boldsymbol{x})$ for an input pattern $\boldsymbol{x}$ corresponds to enforcing the equality

$$f(\boldsymbol{x}) = f(T_\theta \boldsymbol{x}), \quad \forall \boldsymbol{x}, \theta. \tag{2.42}$$

However, local invariance is sometimes considered instead. In this case, the invariance is only imposed around a fixed value of θ. For a transformation centered at $\theta = 0$, so that $T_0 \boldsymbol{x} = \boldsymbol{x}$, local invariance can be enforced by the constraint

$$\left.\frac{\partial}{\partial \theta}\right|_{\theta=0} h(T_\theta \boldsymbol{x}) = 0, \tag{2.43}$$

thus limiting the variation of h (here the real-valued output of the classifier) for a variation of θ.

Some methods are based on another approach, which is to consider the class-invariance with respect to a *domain of the input space* instead of a transformation. In this case, the problem becomes finding f so that

$$f(\boldsymbol{x}) = y_{\mathcal{P}}, \ \forall \boldsymbol{x} \in \mathcal{P}, \tag{2.44}$$

where $y_{\mathcal{P}}$ is the class label of the region $\mathcal{P}$ of the input space. In practice, this approach is particularly useful to provide prior knowledge in regions of input space that lack training samples.

Another type of class-invariance found in pattern recognition is the *permutation-invariance*, i.e. invariance of the class to a permutation of the elements in a structured input. A typical application is a classifier invariant to permutations of rows in matrix inputs. Since permutations are a particular transformation, permutation-invariance can be considered as a special case of transformation-invariance.

2.3.3 Knowledge on the data

Other forms of prior knowledge than class-invariance concern the data more specifically and are thus of particular interest for real-world applications. In this review, the three particular cases that most often occur when gathering data are studied:

- *Unlabeled samples* are available with presumed class-memberships.
- *Imbalance* of the training set is due to a high proportion of samples from one class.
- *Quality of the data* may vary from a sample to another.

Prior knowledge on these can enhance the quality of the recognition if included in the learning. Moreover, not taking into account the poor quality of some data or a large imbalance between the classes can mislead the decision of a classifier.

2.4 Incorporating prior knowledge in SVM: a survey

A review of the methods for incorporating prior knowledge in SVMs is now given. In the last decade, authors considered the introduction of prior knowledge in SVMs and some reviews can be found in the literature. A chapter of the well-known book [232] is dedicated to the incorporation of invariances, but deals only with transformation-invariances. Thus, the authors do not present methods to include knowledge on the data or class-invariance in a domain (which were not available at the time). Nonetheless, they present the three different ways of exploiting prior knowledge, on which relies our categorization of the methods. This categorization considers three groups: sample methods, kernel methods and optimization methods. In [107] or [108], an overview of the works in the field is also given. However, this overview focuses on invariant kernel methods for pattern recognition in general. Here, we are interested in different types of prior knowledge (not only invariance) that can be included in the particular learning machine known as SVM, either in the kernel or not.

In the following, the methods are classified in three categories depending on the means used to include the prior knowledge and the component of the problem that is modified. These three groups of methods are:

- *sample methods* that incorporate the prior knowledge either by generating new data or by modifying the way the data are taken into account;
- *kernel methods* that incorporate the prior knowledge in the kernel function either by selecting the most appropriate kernel or by creating a new kernel;
- *optimization methods* that incorporate the prior knowledge in the problem formulation either by adding constraints to the original problem or by defining a new formulation which intrinsically includes the prior knowledge.

The tree-like diagram of Figure 2.3 presents simultaneously the hierarchy of the methods and of the types of prior knowledge with arrows relating the types of prior knowledge to suited methods. The description of the methods is provided in the following. Further discussions and links between the methods can be found in the next section.

2.4.1 Sample methods

Two different approaches will be described in this section. The first one is based on the generation of virtual samples, while the second one aims at weighting the influence of different samples. Whereas the virtual samples methods focus on the incorporation of transformation-invariance, the weighting of samples allows to include knowledge on the data.

Virtual samples

In machine learning, the generalization ability of the obtained model depends on the number of data at hand. The more representative samples we have, the better we learn. Based on this simple fact, the idea of creating new samples to enlarge the training set was first introduced in [208] as virtual samples and in [2, 3] as "hints". In [201], learning on an extended training set by virtual samples was linked to regularization and it thus showed a justification for the method.

The basic idea of the virtual samples approach [201] is to incorporate a known transformation-invariance as defined by (2.42). New samples are generated from the training data to extend the training set as follows

$$(\boldsymbol{x}_i, y_i) \mapsto (T\boldsymbol{x}_i, y_i), \quad i = 1, \dots, N. \tag{2.45}$$

This method can be easily implemented in the context of pattern recognition. For instance, in image recognition, invariances to translations or rotations are often considered.

To incorporate transformation-invariance into SVMs, the authors of [228] introduced the Virtual SVM (VSVM). The idea is to generate the virtual samples only from the support vectors (SVs) since they contain all the information about the problem. The virtual samples are thus called "virtual SVs" (VSVs). The so-called Virtual SVM requires two SVM trainings in the procedure. The first one extracts the SVs from the training set while the second one is performed on a dataset composed of the SVs and the VSVs.

More details on the application of the virtual sample approach in the context of handwriting recognition are given in section 2.6.

Weighting of samples

The weighting of samples allows to include other forms of prior knowledge than transformation-invariance. It is typically used to express knowledge on the data such as an imbalance between classes, the relative quality of the samples or prior knowledge on unlabeled samples. In practice this amounts to weighting the errors or to choosing a different trade-off parameter C for different samples.

Originally, different misclassification costs C_i were used to deal with unbalanced data sets, i.e. data sets providing much more samples from one class than from the other [33, 129]. C_i is set to a higher value for the less represented class, thus penalizing more the errors on this class. Besides in [62], an equivalence is shown between the soft margin SVM using the ℓ_2-norm of the errors and a hard margin SVM trained with a modified kernel matrix

$$\boldsymbol{K}' = \boldsymbol{K} + \frac{1}{C}\boldsymbol{I}. \tag{2.46}$$

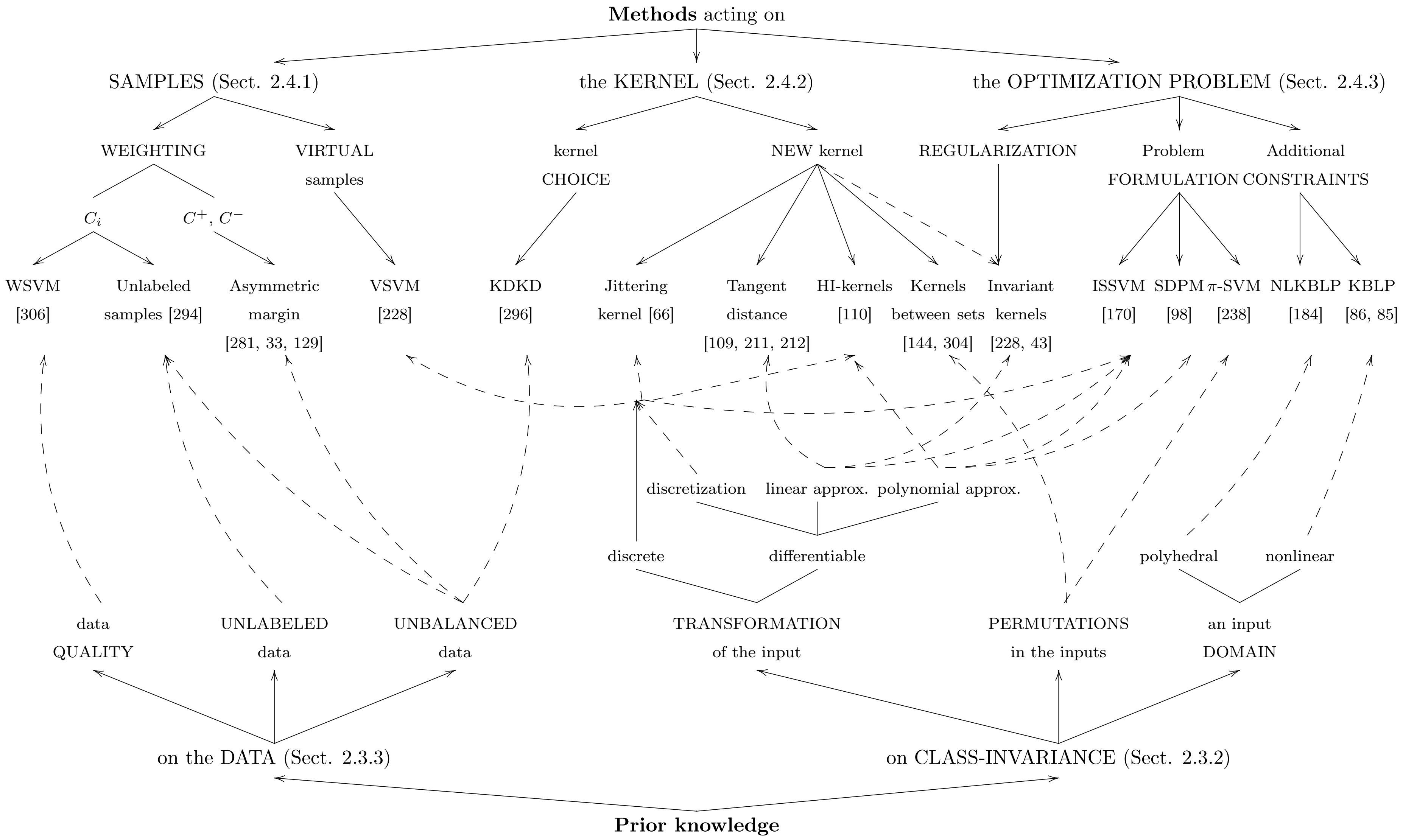


Figure 2.3: *From top to bottom:* Hierarchy of the methods for the incorporation of prior knowledge in SVM from the component of the problem which is modified to the methods. *From bottom to top:* Hierarchy of the types of knowledge that can be incorporated. Each particular type points to the methods available for its inclusion.

This idea is extended in [281] to deal with unbalanced data when different misclassification costs C^+ and C^- are assigned for the positive and negative classes. The kernel matrix is then given by

$$\boldsymbol{K}' = \boldsymbol{K} + \boldsymbol{D}, \tag{2.47}$$

where $\boldsymbol{D}$ is a diagonal matrix with components $D_{ii} = 1/C^+$ for $y_i = +1$ and $D_{ii} = 1/C^-$ for $y_i = -1$. This method amounts to defining an asymmetric margin keeping further away from the decision boundary the class with a higher C. Heuristics are proposed in [33] to tune D_{ii} based on the knowledge of unbalanced data: set $D_{ii} = \lambda n^+/N$ for $y_i = +1$ and $D_{ii} = \lambda n^-/N$ for $y_i = -1$, where n^+ and n^- are, respectively, the number of positive and negative samples in the training set of size N, and λ is a scaling factor.

Another approach is developed in [306] in order to incorporate prior knowledge on the quality of the training data, which may vary from a sample to another. The method, known as Weighted-SVM (WSVM), proposes to set a different cost C_i for each sample with respect to a confidence value based on some knowledge of the data acquisition or labeling procedure.

In [294], prior knowledge on unlabeled samples is considered. Based on presumed class-memberships, pseudo-labels are assigned to these samples, which are then added to the training set with a different weight C_i in the objective function. The incorporation of test samples as unlabeled data in the training is introduced in [276] as "transductive learning". A transductive learning for SVMs, in which the prior knowledge takes the form of the number num_+ of positive samples in the test set, has also been proposed for text classification [128]. In this scheme, the test samples are assigned a misclassification cost C^*, different from the one used for the training samples. In order to deal with unbalanced data, C^* can then be further refined for each class as C^*_+ and C^*_- in accordance with the number num_+.

2.4.2 Kernel methods

This section presents five methods based on the direct modification of the kernel function: the jittering kernels, the sample-to-object distance, the Haar-integration kernels, the kernels between sets and the knowledge-driven kernel design. The first three methods aim at building invariant kernels k that can provide the same value for a sample $\boldsymbol{x}$ and its transformed $T\boldsymbol{x}$:

$$k(\boldsymbol{x}, \boldsymbol{z}) = k(T\boldsymbol{x}, \boldsymbol{z}), \tag{2.48}$$

thus leading to the transformation-invariance (2.42). Besides, the kernels between sets introduce permutation-invariance into the learning, which is another form of class-invariance. The last method considers the problem of selecting the kernel amongst admissible kernels with respect to prior knowledge on the imbalance of the training set.

Jittering kernels

Jittering kernels were first developed for kernel k-Nearest-Neighbors [65] and then presented for incorporating transformation-invariance into SVMs [66]. This approach is related to the virtual SV (VSV) method (see Sect. 2.4.1). Instead of considering an extended training set with all the jittered forms (translated, rotated...) of the training samples, these forms are considered in the kernel itself. Using the notation $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, the jittered form k^J_{ij} of the kernel k_{ij} is computed in two steps [66].

1. Consider the sample $\boldsymbol{x}_i$ and all its jittered forms, and select the one, $\boldsymbol{x}_q$, closest to $\boldsymbol{x}_j$ by minimizing the distance between $\boldsymbol{x}_q$ and $\boldsymbol{x}_j$ in the space induced by the kernel and given by

$$\sqrt{k_{qq} - 2k_{qj} + k_{jj}}. \tag{2.49}$$

2. Let $k^J_{ij} = k_{qj}$.

Using such a jittering kernel may provide an output invariant to transformations. For instance, for a sample $\boldsymbol{x}$ and its translated $T\boldsymbol{x}$, the jittering kernel function can yield $k(T\boldsymbol{x}, \boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x})$.

The computation of such a kernel can be time consuming. However, for a RBF kernel, only k_{qj} needs to be considered for the minimization since k_{qq} and k_{jj} are constants and equal 1. Moreover it is argued that compared to the VSV method it can still be faster for the training by making use of kernel caching. Nonetheless, in testing phase, this kernel might be slower since it requires to repeat the steps (1) and (2) for each new test sample.

Tangent distance

Another approach to incorporate knowledge of transformation-invariance is via the distance measurement, not in the space induced by the kernel but in the input space. In order to do so, one can implement a different distance $\rho(\boldsymbol{x}, \boldsymbol{z})$ instead of the Euclidean distance commonly used in radial basis kernels. For the Gaussian RBF kernel, this yields

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(\frac{-\rho(\boldsymbol{x}, \boldsymbol{z})^2}{2\sigma^2}\right). \tag{2.50}$$

The use of another distance for transformation-invariance has been extensively studied in [239] for neural networks under the name of tangent distance (TD) and originally incorporated in SVMs as TD kernels by [109]. The main idea is to measure the distance not between the samples $\boldsymbol{x}$ and $\boldsymbol{z}$ but between the sets $P_{\boldsymbol{x}}$ and $P_{\boldsymbol{z}}$. These sets contain all the patterns generated by the transformation T of the samples $\boldsymbol{x}$ and $\boldsymbol{z}$ that leave the label unchanged. Thus, the distance between a sample and its translation can be made so that it equals zero.

As simple image transformations can correspond to highly nonlinear transformations in the input space, the tangent distance uses linear approximations. Considering the transformation $T\boldsymbol{x}$ as a combination of n_L local and simple transformations L_{α_k} of parameter α_k in $[\alpha_k^{min}, \alpha_k^{max}]$, it can be linearly approximated by using the tangent vectors $\boldsymbol{\ell}_{\alpha_k}(\boldsymbol{x})$ as

$$T\boldsymbol{x} \approx \boldsymbol{x} + \sum_{k=1}^{n_L} \alpha_k \boldsymbol{\ell}_{\alpha_k}(\boldsymbol{x}). \tag{2.51}$$

The tangent distance is thus defined as the minimal Euclidean distance between the linearly approximated sets of all transformed samples

$$\begin{aligned}\rho(P_{\boldsymbol{x}}, P_{\boldsymbol{z}})^2 = &\min_{\alpha_1,\dots,\alpha_{n_L},\beta_1,\dots,\beta_{n_L}} \left(\boldsymbol{x} - \boldsymbol{z} + \sum_{k=1}^{n_L} (\alpha_k \boldsymbol{\ell}_{\alpha_k}(\boldsymbol{x}) - \beta_k \boldsymbol{\ell}_{\beta_k}(\boldsymbol{z}))\right)^2 \\ &\text{s.t. } \alpha_k \in [\alpha_k^{min}, \alpha_k^{max}],\ \beta_k \in [\beta_k^{min}, \beta_k^{max}],\ k = 1, \dots, n_L,\end{aligned} \tag{2.52}$$

where β_k and $\boldsymbol{\ell}_{\beta_k}(\boldsymbol{z})$ correspond to the parameter and the tangent vector for the kth local transformation of $\boldsymbol{z}$.

A similar approach was originally taken in [82] where a joint manifold distance was defined by minimizing a distance between sets of transformed samples. When the transformation is approximated by a Taylor expansion, this method is analogous to the tangent distance method of [239]. In [212], these concepts are considered under the name of object-to-object distance, where an object corresponds to the set of all transformed samples $P_{\boldsymbol{x}}$ or $P_{\boldsymbol{z}}$. Sample-to-object distance is also considered, in which case the transformation of only one of the two samples is allowed. This corresponds to a one-sided tangent distance which is computed for a sample $\boldsymbol{x}$ and an object $P_{\boldsymbol{z}}$ by

$$\begin{aligned}\rho(\boldsymbol{x}, P_{\boldsymbol{z}})^2 \approx &\min_{\beta_1,\dots,\beta_{n_L}} \left(\boldsymbol{x} - \boldsymbol{z} - \sum_{k=1}^{n_L} \beta_k \boldsymbol{\ell}_{\beta_k}(\boldsymbol{z})\right)^2 \\ &\text{s.t. } \beta_k \in [\beta_k^{min}, \beta_k^{max}],\ k = 1, \dots, n_L,\end{aligned} \tag{2.53}$$

The sample-to-object (or one-sided tangent distance) method can be related to the jittering kernel method. They both amount to compute the kernel $k(\boldsymbol{x}, \boldsymbol{z})$ between the sample $\boldsymbol{x}$ and the closest pattern generated around the center $\boldsymbol{z}$ by a transformation that does not change the class label. The main difference lies in the implementation: the sample-to-object distance can be considered as an analytical form of the distance used in jittering kernels, whereas these latter require to test every jittered form of the center. Therefore, the sample-to-object method can be faster but introduces restrictions on the class of admissible transformations [212].

Objects can also be coded by local distributions centered at the samples. In this case they are called *soft-objects*. One has then to define a similarity measure between a sample and such an object. Here, tangent vectors can be used to locally approximate the transformations [212] and lead to the Tangent Vector Kernels (TVK) introduced in [211].

Haar-integration kernels

Haar-integration has been introduced for the construction of invariant features in [235]. In a similar approach, Haar-integration has been used to generate invariant kernels known as Haar-integration kernels (HI-kernels) [110]. Consider a standard kernel k_0 and a transformation group $\mathcal{T}$ which contains the admissible transformations (see [235] for a complete definition). The idea is to compute the average of the kernel output $k_0\left(T\boldsymbol{x}, T'\boldsymbol{z}\right)$ over all pairwise combinations of the transformed samples $\left(T\boldsymbol{x}, T'\boldsymbol{z}\right)$, $\forall T, T' \in \mathcal{T}$. The HI-kernel k of k_0 with respect to $\mathcal{T}$ is thus

$$k(\boldsymbol{x}, \boldsymbol{z}) = \int_{\mathcal{T}} \int_{\mathcal{T}} k_0\left(T\boldsymbol{x}, T'\boldsymbol{z}\right) dT dT', \tag{2.54}$$

under the condition of existence and finiteness of the integral (which can be satisfied for instance by discretization of $\mathcal{T}$). An interpretation of this kernel in the feature space F can be given thanks to the following equality [110]

$$\langle \int_{\mathcal{T}} \boldsymbol{\Phi}\left(T\boldsymbol{x}\right) dT, \int_{\mathcal{T}} \boldsymbol{\Phi}\left(T'\boldsymbol{z}\right) dT' \rangle = \int_{\mathcal{T}} \int_{\mathcal{T}} \langle \boldsymbol{\Phi}\left(T\boldsymbol{x}\right), \boldsymbol{\Phi}\left(T'\boldsymbol{z}\right) \rangle \, dT dT' = k(\boldsymbol{x}, \boldsymbol{z}). \tag{2.55}$$

In other words, averaging over $k_0\left(T\boldsymbol{x}, T'\boldsymbol{z}\right)$ is equivalent to computing the inner product between the averages $\overline{\boldsymbol{\Phi}(\boldsymbol{x})}$ and $\overline{\boldsymbol{\Phi}(\boldsymbol{z})}$ of the sets of transformed samples $\{\boldsymbol{\Phi}(T\boldsymbol{x}) | \; T \in \mathcal{T}\}$ and $\{\boldsymbol{\Phi}(T'\boldsymbol{z}) | \; T' \in \mathcal{T}\}$.

Kernels between sets

In [144] a kernel between sets of vectors is proposed. The idea is to classify the samples defined as sets of p-dimensional vectors $\boldsymbol{x}_i$, $\chi = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_i, \dots, \boldsymbol{x}_s\}$, where s is the size of the set. This representation allows to intrinsically incorporate invariance to permutations of vectors $\boldsymbol{x}_i$ in the set. For instance, in image recognition, a vector $\boldsymbol{x}_i$ can represent a point in the image identified by its coordinates and a gray-level $\boldsymbol{x}_i = [x, y, \gamma]^T$. A sample χ is then composed of all the points corresponding to an image. It is clear that the ordering of the vectors inside this sample is irrelevant for the image classification. Thus, the recognition algorithm must include an invariance to permutations of vectors inside a sample, which is included here in the kernel.

The kernel between two sets χ and χ' is defined as the Bhattacharyya's affinity between the distributions P and P' fitted to the sets χ and χ', i.e.

$$k(\chi, \chi') = k(P, P') = \int \sqrt{P(\boldsymbol{x})}\sqrt{P'(\boldsymbol{x})} \; d\boldsymbol{x}. \tag{2.56}$$

The approach here is to consider the elements of χ and χ' as i.i.d. samples from unknown distributions P and P' from a parametric family $\mathcal{P}$. The kernel requires to fit the distributions P and P' to the sets as an intermediate step, which ensures explicit invariance to permutations [144]. When $\mathcal{P}$ is chosen as the family of multivariate normal distributions $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, P and P' are fitted by setting $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ to their maximum likelihood estimates given by the sample mean and the empirical covariance matrix. The kernel is then computed by replacing $P(\boldsymbol{x})$ and $P'(\boldsymbol{x})$ in (2.56) by the probability density functions of a multivariate normal distribution with the estimated parameters.

So far, the distributions are fitted in the vector space $\mathbb{R}^p$, which might be limited considering that $p = 3$ for the image recognition example. However, the method is extended in [144] with an additional kernel $\kappa : \; \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ defined between the vectors $\boldsymbol{x}$ in order to consider P and

P' as distributions over the feature space induced by κ. In this case, a regularized estimate of the covariance matrix, involving the computation of eigenvectors in the feature space by Kernel Principal Component Analysis (KPCA) [231], is used.

Another similar approach, independently developed at the same time, can be found in [304], where a positive definite kernel is defined over sets of vectors represented as matrices. The proposed kernel uses the concept of principal angles, which is extended in the paper to be able to compute principal angles in feature space using only inner products between columns of the input matrices. This extension allows to introduce an additional kernel as in the method of [144] in order to deal with nonlinear cases without requiring to explicitly compute the feature map.

Considering two matrices $\boldsymbol{A} = [\boldsymbol{\Phi}(\boldsymbol{a}_1), \dots, \boldsymbol{\Phi}(\boldsymbol{a}_s)]$ and $\boldsymbol{B} = [\boldsymbol{\Phi}(\boldsymbol{b}_1), \dots, \boldsymbol{\Phi}(\boldsymbol{b}_s)]$, where $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ are vectors of $\mathbb{R}^p$, the proposed kernel between these matrices with column order invariance is given by

$$k(\boldsymbol{A}, \boldsymbol{B}) = \prod_{i=1}^{s} \cos(\theta_i), \tag{2.57}$$

where θ_i stands for the ith principal angle in feature space [304].

Clearly, the permutation-invariance of these methods can also be used as transformation-invariance by building the sets of vectors from jittered forms of the input patterns.

Knowledge-driven kernel selection

All previously described methods involving a modification of the kernel aim at building invariant kernels. The method described here applies to unbalanced training sets and the cases where prior knowledge indicates that the negative class includes a wide variety of samples with only a few available for training. For instance, in face recognition, when classifying between the images representing a particular man (positive class) and the images of other people (negative class), the training set cannot include all the possible faces for the negative class.

In image retrieval, this problem has been tackled in [293] by a knowledge-driven kernel design (KDKD) procedure. The authors highlight the fact that in image retrieval, when the training set is small, the data cannot effectively represent the true distributions of the positive and negative classes, especially the negative one. Based on this prior knowledge, the kernel is designed so that, in the feature space, the positive samples are tightly clustered while the negative samples are pushed away from the positive ones. The main point here is that the negative samples can lie anywhere in the feature space, not necessarily in a cluster, but scattered.

In practice, the kernel k, with parameters $\boldsymbol{\theta}$, is designed by maximizing, w.r.t. $\boldsymbol{\theta}$, the ratio

$$\mathcal{J}(k, \boldsymbol{\theta}) = \frac{\operatorname{tr}(\boldsymbol{S}_{np}^{\Phi})}{\operatorname{tr}(\boldsymbol{S}_{p}^{\Phi})}, \tag{2.58}$$

between the scatter $\operatorname{tr}(\boldsymbol{S}_{np}^{\Phi})$ of the negative samples ($\boldsymbol{x}_i \in \mathcal{D}_n$) and the scatter $\operatorname{tr}(\boldsymbol{S}_{p}^{\Phi})$ of the positive ones ($\boldsymbol{x}_i \in \mathcal{D}_p$) w.r.t. the mean $\boldsymbol{m}_p^{\Phi}$ of the positive ones. In (2.58), the matrices $\boldsymbol{S}_{np}^{\Phi}$ and $\boldsymbol{S}_{p}^{\Phi}$ are defined by

$$\boldsymbol{S}_{np}^{\Phi} = \sum_{\boldsymbol{x}_i \in \mathcal{D}_n} [\boldsymbol{\Phi}(\boldsymbol{x}_i) - \boldsymbol{m}_p^{\Phi}][\boldsymbol{\Phi}(\boldsymbol{x}_i) - \boldsymbol{m}_p^{\Phi}]^T \tag{2.59}$$

$$\boldsymbol{S}_{p}^{\Phi} = \sum_{\boldsymbol{x}_i \in \mathcal{D}_p} [\boldsymbol{\Phi}(\boldsymbol{x}_i) - \boldsymbol{m}_p^{\Phi}][\boldsymbol{\Phi}(\boldsymbol{x}_i) - \boldsymbol{m}_p^{\Phi}]^T. \tag{2.60}$$

The traces of the matrices can be computed from the kernel function as described in [293] and the criterion has continuous first and second order derivatives w.r.t. the kernel parameters as long as the kernel function has. However, nonlinear optimization techniques are required to solve the problem.

The maximization of the ratio (2.58) can be used to find the optimal kernel function amongst a set of admissible kernels, but also to tune the parameters of a previously chosen kernel.

2.4.3 Optimization methods

This section presents the methods that incorporate prior knowledge directly in the problem formulation: invariant kernels, Semidefinite Programming Machines (SDPM), Invariant SimpleSVM (ISSVM), Knowledge-Based Linear Programming (KBLP), nonlinear Knowledge-Based Linear Programming (NLKBLP) and π-SVM. Though it may be argued that the first one belongs to the category of kernel methods, it is derived from a regularization approach minimizing a composite criterion. It is thus categorized as an optimization based method. The first three methods of the list aim at incorporating transformation-invariance, whereas the KBLP method considers class-invariance in polyhedral regions of the input space. From a method originally formulated for support vector regression, an extension of KBLP to arbitrary nonlinear domains, NLKBLP, is proposed in Sect. 2.4.3 for classification. The last method exposed in this review, π-SVM, concerns permutation-invariance for SVMs that classify sets of elements instead of vectors.

Invariant kernels

Regularization of the cost function has been extensively used for neural networks [95, 24], allowing to incorporate prior knowledge on a property of the function to estimate (usually the smoothness). But the implicit inclusion of regularization in SVMs, equivalent to regularization networks [245, 79], might explain why few articles studied the application of regularization techniques for incorporating other forms of prior knowledge into SVMs. However, in the case of classification, the addition of a term to be minimized in the cost function for this purpose has been considered. Nonetheless, it results in a modification of the kernel rather than a modification of the optimization problem as explained below.

The authors of [230] incorporated local invariance in the sense of (2.43) and proposed invariant kernels. Defining the tangent vectors by

$$d\boldsymbol{x}_i = \left.\frac{\partial}{\partial\theta}\right|_{\theta=0} T_\theta \boldsymbol{x}_i, \tag{2.61}$$

allows to include the local invariance (2.43) in the learning of a linear SVM by minimizing

$$\frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{w}^T(d\boldsymbol{x}_i)\right)^2, \tag{2.62}$$

and thus making the weight vector $\boldsymbol{w}$ as orthogonal as possible to the tangent vectors. For the original QP formulation and a linear kernel, the regularized cost becomes

$$J(\boldsymbol{w}) = (1-\gamma)\|\boldsymbol{w}\|_2^2 + \gamma\sum_{i=1}^{N}\left(\boldsymbol{w}^T(d\boldsymbol{x}_i)\right), \tag{2.63}$$

where $\gamma \in [0,1]$ controls the trade-off between the standard SVM ($\gamma = 0$) and a full enforcement of the orthogonality between the hyperplane and the invariance directions ($\gamma \to 1$). Let define $\boldsymbol{C}_\gamma$ as the square root of the regularized covariance matrix of the tangent vectors

$$\boldsymbol{C}_\gamma = \left((1-\gamma)\boldsymbol{I} + \gamma\sum_{i=1}^{N} d\boldsymbol{x}_i d\boldsymbol{x}_i^T\right)^{\frac{1}{2}}. \tag{2.64}$$

Then, minimizing the regularized cost (2.63) under the original constraints (2.3) leads to a standard SVM problem [230], yielding the real-valued output function

$$h(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i \langle \boldsymbol{C}_\gamma^{-1}\boldsymbol{x}_i, \boldsymbol{C}_\gamma^{-1}\boldsymbol{x}\rangle + b. \tag{2.65}$$

Thus a linear invariant SVM is equivalent to a standard SVM where the input is first transformed via the linear mapping $\boldsymbol{x} \mapsto \boldsymbol{C}_\gamma^{-1}\boldsymbol{x}$.

In order to extend directly the approach to nonlinear kernels [43], one would have to compute the matrix $\boldsymbol{C}_\gamma$ in feature space by

$$\boldsymbol{C}_\gamma = \left((1-\gamma)\boldsymbol{I} + \gamma \sum_{i=1}^{N} d\boldsymbol{\Phi}(\boldsymbol{x}_i) d\boldsymbol{\Phi}(\boldsymbol{x}_i)^T \right)^{\frac{1}{2}}, \tag{2.66}$$

where $\boldsymbol{\Phi}(\boldsymbol{x}_i)$ is defined by (1.11), and use the new kernel

$$k(\boldsymbol{x}_i, \boldsymbol{x}) = \langle \boldsymbol{C}_\gamma^{-1} \boldsymbol{\Phi}(\boldsymbol{x}_i), \boldsymbol{C}_\gamma^{-1} \boldsymbol{\Phi}(\boldsymbol{x}) \rangle = \boldsymbol{\Phi}(\boldsymbol{x}_i)^T \boldsymbol{C}_\gamma^{-2} \boldsymbol{\Phi}(\boldsymbol{x}), \tag{2.67}$$

which cannot be directly computed because of the high dimensionality of F and the implicit nature of Φ. Two methods were proposed in [43] to circumvent this issue, but they still suffer from computational problems when applied to large datasets or when more than one invariance is considered.

This approach to incorporating invariance can be related to the virtual samples method that simply adds the transformed samples to the training set (see Sect. 2.4.1). Indeed, some equivalence between the two methods has been shown in [165]. However, the invariant SVM makes the *class* invariant to not only the transformation, but also the *real value* of the output (which can be considered as a class-conditional probability) [43].

Semidefinite Programming Machines (SDPM)

In [98], another formulation of the large margin classifier is developed for incorporating transformation-invariance. The aim is to find an optimal separating hyperplane between trajectories defined as sets of the type $\{T_\theta \boldsymbol{x}_i : \theta \in \mathbb{R}\}$ rather than between points. In practice, the trajectories are based on training samples $\boldsymbol{x}_i$ and a differentiable transformation T, with parameter θ, w.r.t. which the class is known to be invariant. The problem can be solved by approximating T by a transformation $\tilde{T}$ polynomial in θ that can be a Taylor expansion of the form

$$T_\theta \boldsymbol{x}_i \approx \tilde{T}_\theta \boldsymbol{x}_i = \sum_{j=0}^{r} \theta^j \left(\frac{1}{j!} \left. \frac{d^j T_\theta \boldsymbol{x}_i}{d\theta^j} \right|_{\theta=0} \right) = \tilde{\boldsymbol{X}}_i^T \boldsymbol{\theta}, \tag{2.68}$$

where the $(r+1) \times p$-dimensional matrix $\tilde{\boldsymbol{X}}_i$ contains the derivative components and $\boldsymbol{\theta} = [1\ \theta\ \theta^2\ \dots\ \theta^r]^T$. For this type of transformations, the problem of finding the optimal separating hyperplane between trajectories can be formulated as

$$\min_{\boldsymbol{w}} \frac{1}{2} \|\boldsymbol{w}\|_2^2, \quad \text{s.t.} \quad y_i \boldsymbol{w}^T \tilde{\boldsymbol{X}}_i^T \boldsymbol{\theta} \leq 0,\ \forall \theta \in \mathbb{R},\ i = 1, \dots, N. \tag{2.69}$$

The authors of [98] propose an equivalence to this problem in the form of a semidefinite program (SDP) [274], for which efficient algorithms exist. They also show that the resulting expansion of the optimal weight vector $\boldsymbol{w}^*$ in terms of $\tilde{\boldsymbol{X}}_i$ is sparse. Moreover, not only the examples $\tilde{\boldsymbol{X}}_i$ are determined but also their corresponding optimal transformation parameter θ_i^*. Thus, the so-called Semidefinite Programming Machines (SDPM) extend the idea of virtual support vectors (VSVs) [228], since truly virtual samples that are not in the training set are used as SVs.

However, practical issues regarding the application of SDPM to nonlinear classification with kernels remain open.

Invariant SimpleSVM

The authors of [170] propose a general framework for the incorporation of transformation-invariance into the learning based on a modification of the problem formulation. For the hard-margin SVM, the problem reads

$$\begin{aligned} &\min \frac{1}{2} \|f\|_H^2 \\ &\text{s.t. } y_i \left(f(T_\theta \boldsymbol{x}_i) + b \right) \geq 1,\ i = 1, \dots, N,\ \theta \in \Theta, \end{aligned} \tag{2.70}$$

where the function f lies in the Hilbert space H and does not correspond to the decision function, but to the inner product in the feature space. This setting allows one to ensure that not only the training sample $\boldsymbol{x}_i = T_0\boldsymbol{x}_i$, but also the admissible transformations of this sample $T_\theta\boldsymbol{x}_i$, are classified as belonging to the class y_i. To solve the problem, the method requires a discretization of the parameter space Θ based on the assumption that only a finite number of values for θ will yield support vectors. This is reasonable since for the hard-margin SVM, only samples that lie exactly on the margin borders are SVs. In this case, a dual form of the problem with a finite number of Lagrange multipliers $\alpha_i(\theta)$ can lead to the solution. However, for the soft-margin case, any sample lying inside the margin corresponds to a SV. Thus the trajectory of a transformation that goes through the margin would yield an infinite number of SVs.

As highlighted in [170], this method can include other methods that incorporate invariance as follows:

- the Virtual SV approach (see Sect. 2.4.1) is recovered after a discretization of the parameter space Θ;
- the tangent distance based methods (Sect. 2.4.3 and 2.4.2) are recovered by approximating the transformation $T_\theta\boldsymbol{x}_i$ by a first order polynomial;
- the semidefinite programming machine (SDPM, Sect. 2.4.3) is recovered if the transformation $T_\theta\boldsymbol{x}_i$ is approximated by a second order polynomial.

In this framework, the authors proposed an efficient algorithm called Invariant SimpleSVM (ISSVM) based on the SimpleSVM algorithm developed in [289]. However, this algorithm requires the discretization of the parameter space Θ for the non-separable case. This is not necessary for SDPM, which considers all values of θ in $\mathbb{R}$.

Knowledge-Based Linear Programming (KBLP)

The following methods consider the incorporation of prior knowledge into support vector learning by the addition of constraints to the optimization problem. In this framework, class-invariance inside polyhedral regions was introduced for linear classification in [86] and then extended to the nonlinear case via a reformulation of the kernel in [85]. These two methods are regrouped under the name Knowledge-Based Linear Programming (KBLP).

The learning machine considered here uses the linear programming formulation (2.13). Assume as prior knowledge that all the points $\boldsymbol{x}$ on a polyhedral domain $\mathcal{P} = \{\boldsymbol{x}|\ \boldsymbol{B}_+\boldsymbol{x} \leq \boldsymbol{d}_+\}$ are positive samples ($y = +1$). This can be written as the implication

$$\boldsymbol{B}_+\boldsymbol{x} \leq \boldsymbol{d}_+ \ \Rightarrow\ \boldsymbol{w}^T\boldsymbol{x} + b \geq 1, \tag{2.71}$$

where, for a domain of dimension n, $\boldsymbol{B}_+ \in \mathbb{R}^{n\times p}$, $\boldsymbol{x} \in \mathbb{R}^p$ and $\boldsymbol{d}_+ \in \mathbb{R}^n$. The implication (2.71) can be transformed for a linear SVM to an equivalent system of linear inequalities having the solution $\boldsymbol{u} \in \mathbb{R}^n$ [86]

$$\boldsymbol{B}_+^T\boldsymbol{u} + \boldsymbol{w} = \boldsymbol{0}, \quad \boldsymbol{d}_+^T\boldsymbol{u} - b + 1 \leq 0,\ \boldsymbol{u} \geq \boldsymbol{0}. \tag{2.72}$$

For prior knowledge on negative samples ($y = -1$), we have

$$\boldsymbol{B}_-\boldsymbol{x} \leq \boldsymbol{d}_- \ \Rightarrow\ \boldsymbol{w}^T\boldsymbol{x} + b \leq -1, \tag{2.73}$$

which is equivalent to

$$\boldsymbol{B}_-^T\boldsymbol{u} - \boldsymbol{w} = \boldsymbol{0}, \quad \boldsymbol{d}_-^T\boldsymbol{u} + b + 1 \leq 0,\ \boldsymbol{u} \geq \boldsymbol{0}. \tag{2.74}$$

This result is then extended to SVMs with nonlinear kernels by assuming that $\boldsymbol{x} = \boldsymbol{X}^T\boldsymbol{t}$ is a linear combination of the training samples. For the positive class, the "kernelized" prior knowledge becomes

$$\boldsymbol{K}(\boldsymbol{B}_+^T, \boldsymbol{X}^T)\boldsymbol{t} \leq \boldsymbol{d}_+ \ \Rightarrow\ \boldsymbol{\alpha}^T\boldsymbol{D}\boldsymbol{K}\boldsymbol{t} + b \geq 1, \tag{2.75}$$

with the diagonal matrix $\boldsymbol{D} = \text{diag}(y_1, \dots, y_i, \dots, y_N)$ and $\boldsymbol{\alpha}$ defined as in (2.22). The following system of linear inequalities is equivalent

$$\boldsymbol{K}(\boldsymbol{X}^T, \boldsymbol{B}_+^T)\boldsymbol{u} + \boldsymbol{K}\boldsymbol{D}\boldsymbol{\alpha} = \boldsymbol{0}, \quad \boldsymbol{d}_+^T\boldsymbol{u} - b + 1 \leq 0,\ \boldsymbol{u} \geq \boldsymbol{0}. \tag{2.76}$$

These inequalities can then be easily incorporated to the linear program (2.21). With the introduction of $N+1$ slack variables $\boldsymbol{z} = [z_1, \dots, z_i, \dots, z_N]^T$ and ζ, this leads to the addition of $N+1$ linear constraints (not counting $\boldsymbol{u} \geq \boldsymbol{0}$ and $\zeta \geq 0$) as

$$\begin{array}{lrcl}
\min\limits_{\boldsymbol{\alpha}, b, \boldsymbol{\xi} \geq \boldsymbol{0}, \boldsymbol{a}, \boldsymbol{u} \geq \boldsymbol{0}, \boldsymbol{z}, \zeta \geq 0} & & \boldsymbol{1}^T\boldsymbol{a} + C\boldsymbol{1}^T\boldsymbol{\xi} + \mu_1\boldsymbol{1}^T\boldsymbol{z} + \mu_2\zeta & \\
\text{s.t.} & & \boldsymbol{D}(\boldsymbol{K}\boldsymbol{D}\boldsymbol{\alpha} + b\boldsymbol{1}) & \geq \boldsymbol{1} - \boldsymbol{\xi} \\
& -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a} \\
& -\boldsymbol{z} \leq & \boldsymbol{K}(\boldsymbol{X}^T, \boldsymbol{B}_+^T)\boldsymbol{u} + \boldsymbol{K}\boldsymbol{D}\boldsymbol{\alpha} & \leq \boldsymbol{z} \\
& & \boldsymbol{d}_+^T\boldsymbol{u} - b + 1 & \leq \zeta,
\end{array} \tag{2.77}$$

where μ_1 and μ_2 are two trade-off parameters between the training on the data and the learning of the prior knowledge.

Similar constraints can be derived for prior knowledge on negative samples and added to the problem [85]. As prior knowledge on a polyhedral set only requires the addition of a set of linear constraints, knowledge on many polyhedral regions for the two classes can be easily combined and included to the problem.

It must be noticed that the three kernels appearing in (2.77) could be distinct kernels and do not need to be positive semidefinite.

Nonlinear Knowledge Based Linear Programming

In the framework of regression and kernel approximation of functions, Mangasarian and his coworkers proposed a new approach based on a nonlinear formulation of the knowledge [184][4] to overcome the drawbacks of the previously described KBLP method. It is presented here and extended to classification, and thus to class-invariance inside an input domain. The prior knowledge can now be considered on any nonlinear region of the input space and takes the general form

$$\forall \boldsymbol{x},\ \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \ \Rightarrow\ h(\boldsymbol{x}) \geq \overline{h}(\boldsymbol{x}), \tag{2.78}$$

where h is the real-valued output of the model and $\overline{h}$ a known function representing the prior knoweldge. This implication can be shown [183] to be equivalent to

$$\forall \boldsymbol{x},\ \exists \boldsymbol{v} \geq \boldsymbol{0},\ h(\boldsymbol{x}) - \overline{h}(\boldsymbol{x}) + \boldsymbol{v}^T\boldsymbol{g}(\boldsymbol{x}) \geq 0. \tag{2.79}$$

In this formulation, $h, \boldsymbol{g}, \overline{h}$ are arbitrary nonlinear functions. Indeed the demonstration of the equivalence requires $h, \boldsymbol{g}, -\overline{h}$ to be convex but only for the implication (2.78) ⇒ (2.79). The implication (2.79) ⇒ (2.78), which is the one we are interested in, holds for any $h, \boldsymbol{g}, \overline{h}$. Nonetheless, the inequality (2.79) must be verified for all $\boldsymbol{x}$ and thus it cannot be directly included in the linear program as a finite set of constraints. To overcome this, a discretization of the knowledge is performed over a set of points $\{\boldsymbol{x}_p\} \subset \{\boldsymbol{x} |\ \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\}$, yielding a finite set of constraints.

For the classification case, we can use the real-valued output of the classifier, $h(\boldsymbol{x}) = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\boldsymbol{D}\boldsymbol{\alpha} + b$, and $\overline{h}(\boldsymbol{x}) = +1$. Of course, if the region considered is known to belong to the negative class instead of the positive class, the setting becomes $h(\boldsymbol{x}) = -\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\boldsymbol{D}\boldsymbol{\alpha} - b$ and $\overline{h}(\boldsymbol{x}) = -1$. Associating slack variables z_p to the N_p points of discretization $\{\boldsymbol{x}_p\}$ gives the Nonlinear Knowledge-Based Linear Program (NLKBLP)

$$\begin{array}{lrcl}
\min\limits_{\alpha, b, \boldsymbol{\xi} \geq \boldsymbol{0}, \boldsymbol{a}, \boldsymbol{v} \geq \boldsymbol{0}, z_p \geq 0} & & \boldsymbol{1}^T\boldsymbol{a} + C\boldsymbol{1}^T\boldsymbol{\xi} + \mu \sum_{p=1}^{N_p} z_p & \\
\text{s.t.} & & \boldsymbol{D}(\boldsymbol{K}\boldsymbol{D}\boldsymbol{\alpha} + \boldsymbol{1}b) & \geq \boldsymbol{1} - \boldsymbol{\xi} \\
& -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a} \\
& & h(\boldsymbol{x}_p) - \overline{h}(\boldsymbol{x}_p) + \boldsymbol{v}^T\boldsymbol{g}(\boldsymbol{x}_p) + z_p & \geq 0,\ p = 1, \dots, N_p.
\end{array} \tag{2.80}$$

A difference between this method and the polyhedral method is that the constraints are applied on an arbitrarily discretized domain, while with a polyhedral domain, the constraints hold on the whole domain (without discretization).

It is thus possible to include prior knowledge such as the class-membership of an arbitrary region into support vector learning. For each region a set of constraints is added to the linear program. The method is thus only limited by computational power.

[4] This work was originally published as a technical report [183].

Permutation-invariant SVM (π-SVM)

The paper [238] focuses on the issue of section 2.4.2, i.e. building a classifier between sets of vectors (here in matrix form) that incorporates permutation-invariance (here between rows of matrices). But the approach is rather different from the ones of [144] or [304]. Here, the permutation invariance is not incorporated in the kernel but in the learning machine itself. To do so, an SVM with matrix inputs $\boldsymbol{x} \in \mathbb{R}^{m \times p}$ instead of vectors is considered by defining a function $\pi : \mathbb{R}^{m \times p} \times \mathbb{R}^{m \times p} \to \mathbb{R}$ as

$$\pi(\boldsymbol{A}, \boldsymbol{B}) = \sum_{i=1}^{p} \langle \boldsymbol{A}_i, \boldsymbol{B}_i \rangle = \sum_{i=1}^{p} \sum_{j=1}^{m} A_{ij} B_{ij}, \tag{2.81}$$

and the norm of a matrix as $\|\boldsymbol{A}\| = \sqrt{\sum_{i=1}^{p} \|\boldsymbol{A}_i\|^2}$. The training of such an SVM can still be written as in (2.10) by considering $\boldsymbol{w}$ as a matrix in $\mathbb{R}^{m \times p}$ and replacing the inner product $\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle$ in the constraints by $\pi(\boldsymbol{x}_i, \boldsymbol{w})$. Similarly, the class of a sample $\boldsymbol{x}$ is given by $f(\boldsymbol{x}) = \text{sign}(\pi(\boldsymbol{x}, \boldsymbol{w}) + b)$. The margin of this classifier can be defined as the minimal value for $y_i(\pi(\boldsymbol{x}_i, \boldsymbol{w}) + b)$ over the training set, since maximizing this margin would lead to a better separation of the training data.

The main steps of the proposed procedure to train a permutation-invariant SVM (π-SVM) [238] are as follows:

1. compute the radius R and centroid of the smallest hypersphere enclosing all training data;
2. solve the SVM on the training data composed of matrices;
3. find the permutation of rows for each training sample that both minimizes the radius R and maximizes the margin (with a chosen trade-off parameter);
4. permute the rows of the training matrices accordingly;
5. repeat from step 1 until some criterion is met.

For all these steps, efficient algorithms, of which the description can be found in [238], exist. The idea here is to permute the rows of training matrices so as to minimize the bound on the generalization error based on the ratio between the radius of the data and the margin. The class of a test sample $\boldsymbol{x}$ is then determined by $f(\boldsymbol{x})$ computed for the permutation of rows of $\boldsymbol{x}$ that yields the larger margin.

2.5 Discussions

This section starts with some general remarks on the incorporation of prior knowledge in SVMs that highlight the activity in this field of research. Then, a technical discussion on particular aspects of the methods is presented to highlight the links among them. Notably, a unifying framework from an implementation point of view is proposed in section 2.5.2.

2.5.1 General issues

Our overview classified the approaches for incorporating prior knowledge into SVMs into three categories: sample methods, kernel methods and optimization methods.

Sample methods are often chosen in practice for their simplicity of use. At the contrary, kernel methods may suffer from computational issues. However, most of the current research aims at building invariant kernels. One reason is that these kernels may also be directly applied to other kernel-based classifiers such as Kernel Principal Component Analysis (KPCA) [231] or Kernel Fisher Discriminant (KFD) [191].

Also, most of the works focus on one particular type of prior knowledge: class-invariance to a transformation of the input. This can be explained by the availability of this type of prior knowledge in many of the pattern recognition problems such as image recognition and all its various applications. However, on a smaller scale, other forms of invariance are also studied, for instance when considering structured inputs such as matrices. In this setting, invariance to permutations of rows as proposed by [144], [304] or [238] can be crucial for the problem. The methods proposed by

[86], [85] and their extension based on [183] allow to include some class-invariance knowledge on regions of the input space, which might be interesting if, for instance, these regions lack training samples.

Methods that incorporate knowledge on the data, such as the Weighted-SVM [306], are also interesting since they include knowledge on the experimental setup used to provide the training data. From a practical point of view, it is clear that in real-world applications, the process of gathering data may suffer from a lack of accuracy or a variability of the accuracy along the process. Being able to track these and learn accordingly is an important issue. As an example, the labeling of images in handwriting recognition is not always exact and can sometimes differ with respect to the person who is asked to label a pattern. As highlighted in [248], a certain amount of errors made by the classifiers are due to images that humans have difficulty in identifying because of cursive writing, degradation and distortion due to the quality of the scanner or the width of the tip of the writing instrument. These problems are present and identifiable in most of the common databases (see [248] for an analysis on the MNIST, CENPARMI, USPS and NIST SD 19 databases) and should be taken into account to be able to improve the performance of recognition systems.

2.5.2 Technical discussions

In the following, a regularization framework is proposed to regroup both the sample and optimization methods. A comparison of the invariant kernel methods is then proposed to show the different spaces in which the distance measures are considered by the different methods. The end of this section briefly discusses the perspective of combining the methods.

Sample and constrained methods in a regularization framework

Though stemming from different approaches, many of the presented methods can be interpreted in a regularization framework from the way they are implemented. Actually both sample and constrained methods amount to minimizing a composite criterion J_C with additional constraints

$$\begin{aligned} \min \ & J_C = J_{SVM} + J_{PK} \\ \text{s.t.} \ & \boldsymbol{c}_{SVM} \leq \mathbf{0} \\ & \boldsymbol{c}_{PK} \leq \mathbf{0}, \end{aligned} \tag{2.82}$$

where J_{SVM} and $\boldsymbol{c}_{SVM}$ correspond to the usual SVM criterion and constraints in (2.10), whereas J_{PK} and $\boldsymbol{c}_{PK}$ are added to the problem to include the prior knowledge. The setting for the different methods is derived in the following and summarized in Table 2.2.

All virtual sample methods (Sect. 2.4.1) basically add new samples to the training set, which simply corresponds to augmenting the size of the training set N by the number of virtual samples N_V in the optimization problem (2.10). Separating the training samples $\boldsymbol{x}_i$ from the virtual samples $\tilde{\boldsymbol{x}}_i$ in the writing yields the setting of Table 2.2 for (2.82).

Though originally developed for different purposes and from different points of view, all the methods based on weighting (Sect. 2.4.1) can be written as a standard SVM problem with the parameter C set to a different value C_i for each sample. For asymmetric margins, C_i can only take two values C^+ and C^- depending on the class of the sample $\boldsymbol{x}_i$: positive (for i in $\mathcal{P} = \{i : \ y_i = +1\}$) or negative (for i in $\mathcal{N} = \{i : \ y_i = -1\}$). Considering C as an average weight allows to write the problem as in (2.82) with the settings of Table 2.2 for asymmetric margin methods and the Weighted-SVM. Besides, the method of [294] for incorporating unlabeled samples can be seen as a mixture of virtual samples and weighting by considering the N_U unlabeled samples as extra samples $\tilde{\boldsymbol{x}}_i$ weighted by C_i.

The KBLP and NLKBLP problems (Sect. 2.4.3 and 2.4.3) are directly formulated as in (2.82), except for J_{SVM} and $\boldsymbol{c}_{SVM}$, which correspond now to the criterion and constraints used by the linear programming form of SVM (2.13). Because of the required discretization of the domain of knowledge, the NLKBLP method can be seen as adding virtual samples to the training set. In this case, the samples are not generated by transformations of other samples as usual, but chosen in the input space by discretization of a region that might actually contain no sample. An interesting point is that, although basically corresponding to the same problem from an optimization viewpoint, the

Table 2.2: Setting for the regularization framework with respect to the different methods.

Method	J_{PK}	$\boldsymbol{c}_{PK} \leq \boldsymbol{0}$
Virtual samples	$C\sum_{i=1}^{N_V}\tilde{\xi}_i$	$\tilde{y}_i(\langle\tilde{\boldsymbol{x}}_i,\boldsymbol{w}\rangle + b) \geq 1 - \tilde{\xi}_i,\ i = 1,\dots,N_V$
Asymmetric margin	$(C^+ - C)\sum_{i\in\mathcal{P}}\xi_i + (C^- - C)\sum_{i\in\mathcal{N}}\xi_i$	
WSVM	$\sum_{i=1}^{N}(C_i - C)\xi_i$	
Unlabeled samples	$\sum_{i=1}^{N_U}C_i\tilde{\xi}_i$	$\tilde{y}_i(\langle\tilde{\boldsymbol{x}}_i,\boldsymbol{w}\rangle + b) \geq 1 - \tilde{\xi}_i$
KBLP	$\mu_1\sum_{i=1}^{N}z_i + \mu_2\zeta$	$-\boldsymbol{z} \leq \boldsymbol{K}(\boldsymbol{X}^T,\boldsymbol{B}^T)\boldsymbol{u} + \boldsymbol{K}\boldsymbol{D}\boldsymbol{\alpha} \leq \boldsymbol{z}$, $\boldsymbol{d}^T\boldsymbol{u} - b + 1 \leq \zeta$
NLKBLP	$\mu\sum_{p}^{N_p}z_p$	$f(\boldsymbol{x}_p) - h(\boldsymbol{x}_p) + \boldsymbol{v}^T\boldsymbol{g}(\boldsymbol{x}_p) + z_p \geq 0$, $p = 1,\dots,N_p$

VSVM and NLKBLP methods are radically different. The first one aims at improving the boundary between two regions with training samples that are close but that belong to different classes. The second one becomes of particular interest when enhancing the decision function in a region of the input space that lacks training samples but on which prior knowledge is available.

The methods of section 2.4.3 for the incorporation of invariances conform obviously to this framework since they are initially formulated as the minimization of a composite criterion. However, as they correspond in practice to a modification of the kernel, they do not appear in Table 2.2, which focuses on the practical implementation of the methods.

Kernel methods: invariance via distance measure, but in which space?

Most of the kernel methods presented in section 2.4.2 implement transformation invariance by modifying the computation of the distance between the patterns. These methods are summarized in Table 2.3. It can be noticed that while the jittering kernel minimizes a distance in the feature space F, the sample-to-object distance is considered in the input space. On the other hand, the Haar-integration kernel does not look for a minimal distance, but rather computes the distance in F between averages over transformed samples.

Table 2.3: Invariant kernel functions. $\mathcal{Z}$ contains all the jittered forms of $\boldsymbol{z}$.

Method	kernel function	specificity
Jittering kernels	$k(\boldsymbol{x},\boldsymbol{z}) = k(\boldsymbol{x},\hat{\boldsymbol{z}})$	$\hat{\boldsymbol{z}} = \operatorname*{arg\,min}_{\boldsymbol{z}\in\mathcal{Z}}\|\boldsymbol{\Phi}(\boldsymbol{x}) - \boldsymbol{\Phi}(\boldsymbol{z})\|_F$
Sample-to-object	$k(\boldsymbol{x},\boldsymbol{z}) = k(\boldsymbol{x},\hat{\boldsymbol{z}})$	$\hat{\boldsymbol{z}} = \operatorname*{arg\,min}_{\boldsymbol{z}\in P_{\boldsymbol{z}}}\|\boldsymbol{x} - \boldsymbol{z}\|$
Object-to-object	$k(\boldsymbol{x},\boldsymbol{z}) = k(\hat{\boldsymbol{x}},\hat{\boldsymbol{z}})$	$(\hat{\boldsymbol{x}},\hat{\boldsymbol{z}}) = \operatorname*{arg\,min}_{(\boldsymbol{x},\boldsymbol{z})\in P_{\boldsymbol{x}}\times P_{\boldsymbol{z}}}\|\boldsymbol{x} - \boldsymbol{z}\|$
Haar-integration	$k(\boldsymbol{x},\boldsymbol{z}) = \overline{k_0(\boldsymbol{x},\boldsymbol{z})}$	$\overline{k_0(\boldsymbol{x},\boldsymbol{z})} = \langle\overline{\boldsymbol{\Phi}(\boldsymbol{z})},\overline{\boldsymbol{\Phi}(\boldsymbol{x})}\rangle$

Combinations of methods

Combining methods from different categories (w.r.t. our categorization) seems possible since they act on different parts of the problem. Thus nothing prevents us from using a jittering kernel on an extended training set with prior knowledge on polyhedral regions of input space. But this does not exclude combinations of methods of the same category. Actually all the methods of Table 2.2 can be combined since they all amount to the addition of a term in the criterion and optionally some constraints. Such combinations become interesting when two methods are used for different purposes.

However, the combination of methods leads to an increase of the algorithm complexity. Therefore, the methods must be chosen with care by looking at their complementarity in order to yield a respectable improvement.

2.6 Application to handwritten digit recognition

Handwriting recognition has always been a challenging task in pattern recognition. But since handwriting depends much on the writer and because we do not always write the same character in exactly the same way, building a general recognition system that would recognize any character with good reliability in every application is not possible. Typically, the recognition systems are tailored to specific applications to achieve better performances. In particular, handwritten digit recognition has been applied to recognize amounts written on checks for banks or zip codes on envelopes for postal services (the USPS database). In these two cases, good results were obtained.

A handwritten digit recognition system can be divided into several stages: preprocessing (filtering, segmentation, normalization, thinning. . .), feature extraction (and selection), classification and verification. This section focuses on feature extraction and classification. The main purpose of this example is to show that learning the features in a black box manner can be very efficient, while transformation-invariance is necessary to achieve state-of-the-art performance.

The recognition system is applied to handwritten digit recognition and compared to other methods on the MNIST database [161] (famous digit database often used as a benchmark). Since the aim here is to show how to use virtual samples in a practical application, the reader is referred to the original presentation of these results [153] for more details on the feature extractor and the experiment setup[5].

Feature extraction by learning

A feature extractor processes the raw data (the gray-scaled image in this case) to generate a feature vector. This vector has a smaller dimension than the orginal data while holding the maximum amount of useful information given by the data. As an example, a feature extractor might build a feature vector whose component i is the number of crossing points in the ith line of the image. This feature extractor is constructed from prior knowledge on the application, because we know that the crossing points possess pertinent information for differentiating digits. Another approach to this problem is to consider the feature extractor as a black box trained to give relevant features as outputs with no prior knowledge on the data. In the following, a neural network, originally developed as a classifier, is used as a feature extractor.

Amongst all the classifiers that have been applied to character recognition, neural networks became very popular in the 80's as demonstrated by the performances obtained by LeCun's LeNet family of neural networks [160]. These are convolutional neural networks that are sensitive to the topological properties of the input (here, the image) whereas simple fully connected networks are not. As a convolutional neural network, LeNet-5 extracts the features in its first layers. For a 10-class problem, the last layer has 10 units, one for each class. The outputs of this layer can be considered as membership probabilities and an input pattern is assigned to the class corresponding to the maximal probability. The weights (parameters of the network) are trained by minimizing the errors between the outputs of the last layer and the targets encoding the class-labels.

[5] Note that the results presented here were obtained prior to the thesis. However, submission and publication of [153] took place during the thesis.

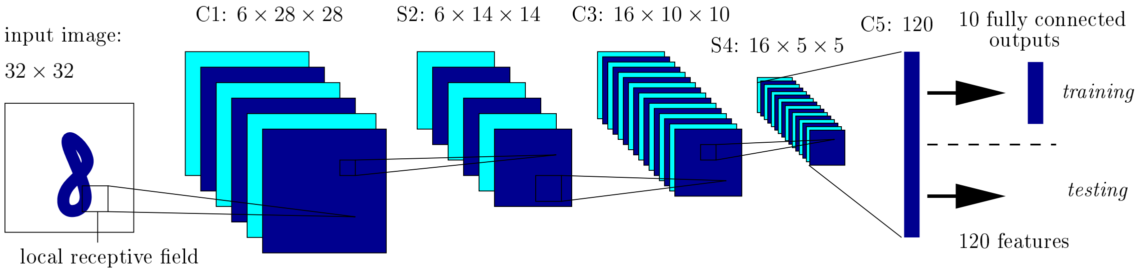


Figure 2.4: The architecture of the trainable feature extractor (TFE). The outputs of the layer C5 are either directed to the 10 outputs for the training phase or used as features for the testing phase.

Here the idea is to use an architecture where the last layers of the original LeNet-5, originally containing nonlinear units, are replaced by a set of 10 linear output units. These units will thus perform linear classification of the samples on the basis of features given by the previous layer, i.e. the last convolutional layer $C5$ as shown on Fig. 2.4. According to the training procedure minimizing the output error, the 120 outputs of layer $C5$ are optimized so that the samples can be linearly separated by the output layer. Once the network has been trained, these features can be used as inputs for any other classifier. The resulting system is a trainable feature extractor (TFE) that can quickly be applied to a particular image recognition application without prior knowledge on the features.

Transformation-invariance

In character recognition, the input takes the form of a 2-dimensional image containing rows of pixels. For gray level images, a pixel is represented by its coordinates (x, y) and its value p (usually a number between 0 and 255 indicating its darkness or brightness). It is clear that an image representing a character will still represent the same character if, for instance, translated by one pixel. Thus, one often looks for classifiers that can incorporate some translation-invariance as prior knowledge.

If the number of training samples is small, generating additional data using transformations (such as translations) may improve the performances of character recognition [240]. Using neural networks, results on the MNIST database were improved by applying transformations on the data and thus multiplying the size of the training set by ten [160]. This shows that one can create new training samples by using prior knowledge on transformation-invariance properties in order to increase the recognition ability of the classifier. The following describes two image transformations typically used in character recognition.

Simple distortions such as translations, rotations and scaling can be generated by applying affine displacement fields to images. For each pixel (x, y), a target location (u, v) is computed w.r.t. the displacement fields $\Delta x(x, y)$ and $\Delta y(x, y)$ at this position by $(u, v) = (x + \Delta x(x, y),\ y + \Delta y(x, y))$. For instance if $\Delta x(x, y) = \alpha x$ and $\Delta y(x, y) = \alpha y$, the image is scaled by α.

The new grey level in the transformed image for the pixel at position (x, y) is the grey level of the pixel at position (u, v) in the original image. For affine transformations, the target location (u, v) is computed by

$$\begin{bmatrix} u \\ v \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} x \\ y \end{bmatrix} + \boldsymbol{b} + \begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.83}$$

where the 2×2-matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$ are the parameters of the transformation, e.g.

- for scaling: $\boldsymbol{A} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$,
- for translations of one pixel: $\boldsymbol{A} = \boldsymbol{0}$ and $\boldsymbol{b}$ takes 8 different values, in the 4 main directions and in the 4 diagonals, i.e. $[1, 0]^T$, $[0, 1]^T$, $[-1, 0]^T$, $[0, -1]^T$, $[1, 1]^T$, $[-1, 1]^T$, $[-1, -1]^T$ and $[1, -1]^T$.

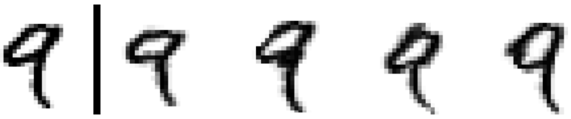


Figure 2.5: Samples generated by elastic distortion from the original pattern shown on the left.

Table 2.4: Best results on the MNIST database. The second column shows whether virtual samples were used and with which transformations they were created.

Classifier	distortion	reference	test error (%)
SVM		[66]	1.4
LeNet-5		[160]	0.95
TFE-SVM		[153]	0.83
VSVM	affine	[66]	0.8
LeNet5	affine	[160]	0.8
boosted LeNet-4	affine	[160]	0.7
VSVM2	affine	[66]	0.68
VSVM2 + deskewing	affine	[66]	0.56
TFE-SVM	elastic	[153]	0.56
TFE-SVM	affine	[153]	0.54
convolutional neural net. (NN)	elastic	[240]	0.4
large conv. NN + pretraining	elastic	[217]	0.39
human		[162]	0.2

For transformed pixels with non-integer target locations (u, v), bilinear interpolation is used [240].

Elastic distortion is another image transformation, introduced by [240] to imitate the variations of the handwriting. The generation of the elastic distortion is as follows. First, random displacement fields are created from a uniform distribution between -1 and $+1$. They are then convolved with a Gaussian of standard deviation σ. After normalization and multiplication by a scaling factor α that controls the intensity of the deformation, they are applied on the image. A small σ (called here the elastic coefficient) means more elastic distortion. For a large σ, the deformation approaches affine, and if σ is very large, then the displacements become translations. Figure 2.5 shows some samples generated by elastic distortion.

Recognition system and results

The proposed recognition system is composed of the trainable feature extractor (TFE) of Fig. 2.4 connected to a multitude of binary SVMs for the testing phase[6]. It is thus labeled TFE-SVM in the following experiments. The multiclass approach for the SVMs is either the one-vs-one method that needs 44 binary classifiers to separate every couples of classes or the one-vs-all method that involves 10 classifiers, each one assigned to the separation of one class from the others. In the results of Table 2.4, both methods are applied and only the best of the two results is shown.

In character recognition, the generation of virtual samples became very popular and almost necessary to achieve first-class performances as can be seen in Table 2.4 showing the best results on the MNIST database (also available and updated at the MNIST homepage [161]). Other transformations, such as morphing [138], were specifically developed and it appears that some of the best results are obtained by elastic distortions [240, 217] even if it is based on random displacement of pixels in the image. This highlights the fact that more samples help to learn better even if they are not absolutely accurate.

An analysis of the errors performed in [153] led to the conclusion that the performance could not be increased above a certain limit, because of bad samples in the test set not recognizable without

[6] Note that the idea of using the extracted features of a convolutional network for another classifier can be found in [162]. The last layers of a LeNet-4 network were replaced by a K-Nearest Neighbors (K-NN) classifier to work on the extracted features. However, this method did not improve the results compared to a plain LeNet-4 network.

ambiguity by humans. Nonetheless some error samples are very clear and are misrecognized because of their structure and the lack of samples of the same prototype in the training set. Regarding these errors, the generation of more samples of rare prototypes could lead to further improvement.

2.7 Conclusion

The fundamentals of the Support Vector Machines (SVMs) for classification have been presented together with the different formulations of the optimization problem resulting from the training of such machines. A review of the literature concerning the incorporation of prior knowledge in SVMs has been exposed. The methods are classified in three categories depending on the implementation approach (via samples, in the kernel or in the problem formulation). Two main types of prior knowledge that can be included by these methods have been considered: class-invariance and knowledge on the data. Most of the work in this field has been focused so far on transformation-invariance, either via the kernel function or via extended training sets. Nonetheless, a recent approach considers prior knowledge on a polyhedral domain of the input space for which the class is known. It has been extended here for arbitrary regions and results in the addition of linear constraints to the optimization problem, thus providing an easy mean to code knowledge on multiple regions for the two classes. Finally, a regularization framework has been used to regroup both the sample and constrained methods from an implementation point of of view.

Being able to include expert knowledge in the learning will be a key element in the future for the increase of classifiers performance on benchmark datasets and practical applications. Further research might explore other forms or combinations of prior knowledge together with optimized algorithms for their efficient implementations.

2.8 Further reading

Support Vector Machines. Other formulations of SVMs have been proposed, of which a non-exhaustive list is given here. The Smooth SVM [164] uses a smooth unconstrained reformulation of the quadratic program, solved by a Newton-Armijo algorithm. The Lagrangian SVM [180] minimizes the squares of the slack variables and is thus not required to impose positivity of these variables. This leads to an unconstrained quadratic program (involving only nonnegativity constraints) very efficiently solved for linear SVM or for small scale nonlinear problems. In a Bayesian framework, the Relevance Vector Machine (RVM) [261] extends the SVMs to provide confidence on the predictions. Probabilistic outputs are also considered in [206] through sigmoid functions.

Kernel methods. The kernel trick allows to easily extend linear methods to the nonlinear case. This idea has been used to propose other kernel machines based on standard linear methods such as the Kernel Fisher Discriminant (KFD) [191] for classification, Kernel Principal Component Analysis (KPCA) [231] for dimensionality reduction or Kernel Independent Component Analysis (KICA) [9] for blind source separation (see [192] or [236] for overviews).

Prior knowledge. The review of the methods for the incorporation of prior knowledge in SVMs (Sect. 2.4) is based on work from 2006 and published in [149]. Therefore more recent references and methods available by now may be missing from the review (such as [279] for instance). In addition, an interesting problem not covered in this review, though it may be regarded as the inclusion of prior knowledge, is the classification of structured inputs such as graphs or strings. Regarding this issue, the reader should refer to the studies on kernels between graphs presented in the papers [145, 56, 91, 139] or to the book [236] and the references therein that provide a framework for building kernels for structured data.

Multiclass problem. The multiclass problem has also attracted much attention from the machine learning community. An empirical study of various ways for combining binary classifiers is given e.g. in [74]. On the other hand, multiclass SVM classifiers dealing directly with this issue are described in [300, 61, 103] with a QP based approach, [22, 31, 21] with a LP based approach, and in [254] for a LS-SVM approach.

Chapter 3

Support Vector Machines for regression

Abstract. *This chapter presents the kernel methods in the regression framework. The Support Vector Machines are reformulated for regression, leading to the so-called* Support Vector Regression *(SVR) method described here in its quadratic and linear programming forms. The least squares SVM (LS-SVM) is then introduced as a simplification of SVR. Finally, the methods are experimentally studied on a nonlinear system identification example. Some experimental results on the robustness to outliers of the algorithms are also given at the end of the chapter.*

In this chapter we show how kernel methods including Support Vector Regression (SVR) and Least Squares SVM (LS-SVM) can be applied to the regression problem. In nonlinear function approximation, Support Vector Regression (SVR) has proved to be able to give excellent performances in various applications [193, 247, 186]. Similar to SVMs for classification, SVR originally leads to a quadratic programming (QP) problem [62, 244]. Other formulations of the SVR problem minimizing the ℓ_1-norm of the parameters instead of the ℓ_2-norm of the weights can be derived to yield linear programs (LP) [299, 21, 246, 181]. Some advantages of this latter approach can be noticed compared to the QP formulation such as the sparsity of support vectors [299, 21, 246] or the ability to use more general kernels [179].

A lot of work is now available in the literature on SVR. This chapter describes three state-of-the-art SVR training methods (QP-based SVR, LP-based SVR and LS-SVM), while highlighting their pros and cons, as well as their links and relationships with other classical approaches. The aim is to introduce the prerequisites for the main contributions of the thesis presented in chapters 4 and 5.

SVR is first presented in its quadratic programming form before introducing the equivalent linear programs, as in chapter 2 for classification. Least Squares SVMs are then described in the regression framework. An example showing the application of SVR to nonlinear system identification is given at the end of this chapter together with some experimental illustration of the robustness to outliers.

3.1 Support Vector Regression (SVR)

In this chapter, we consider the regression problem as defined in section 1.3 on page 6. The goal is thus to approximate the data $(\boldsymbol{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R}$ by a model f. A general description of how does Support Vector Regression (SVR) build linear models is given below. The next subsections will then use the kernel trick to extend SVR to the nonlinear case and introduce the algorithms.

Support vector regression aims at finding the *flattest* linear function f, defined as

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b, \tag{3.1}$$

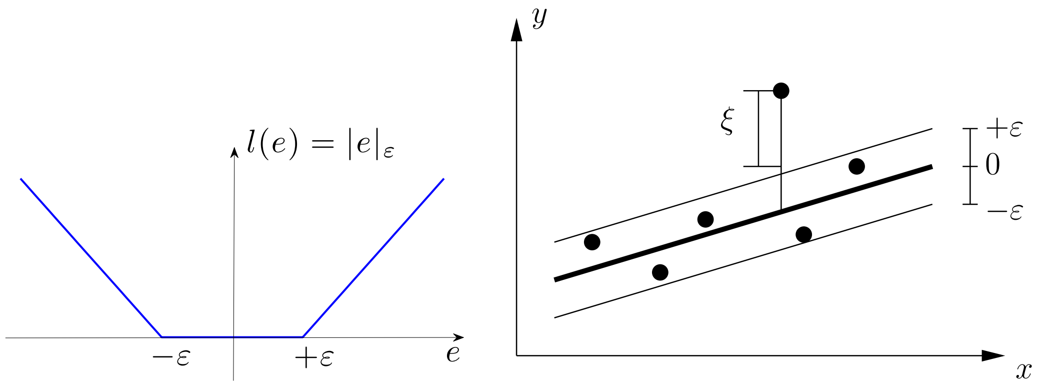


Figure 3.1: *Left:* the *ε-insensitive* loss function for $e = y - f(\boldsymbol{x})$. *Right:* the corresponding tube of insensitivity.

that deviates from the training data by ε at most, i.e. that satisfies

$$|f(\boldsymbol{x}_i) - y_i| \leq \varepsilon, \; i = 1, \dots, N. \tag{3.2}$$

Note that this approach can be related to bounded-error estimation approaches, see e.g. [291, 124]. As the flattest of the linear functions is the constant function, the algorithm looks for a function f that satisfies the constraints (3.2) with minimal norm of the weights $\boldsymbol{w}$, i.e.

$$\begin{aligned} &\min_{\boldsymbol{w},b} \; \|\boldsymbol{w}\|_2^2 && (3.3)\\ &\text{s.t. } |y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i\rangle - b| \leq \varepsilon, \; i = 1, \dots, N. \end{aligned}$$

However, this formulation of the problem considers that a solution f approximating the training set with a precision ε everywhere actually exists. Moreover, in the presence of noisy data and outliers, it is better to allow some errors than to fit exactly to the data. As for classification (see the soft-margin SVM, section 2.1.1 on page 18), a relaxation of the constraints is used in practice. This amounts to minimize an ε-insensitive loss function, shown in Fig. 3.1 and defined as in [244] by[1]

$$l(f(\boldsymbol{x}), y) = |y - f(\boldsymbol{x})|_\varepsilon = \begin{cases} 0 & \text{if } |y - f(\boldsymbol{x})| \leq \varepsilon, \\ |y - f(\boldsymbol{x})| - \varepsilon & \text{otherwise.} \end{cases} \tag{3.4}$$

As shown on the right of Fig. 3.1, this loss function builds a tube of insensitivity in which the errors are meaningless. Points outside of the tube and leading to errors larger than ε, which is half of the tube section w.r.t. y, are penalized linearly.

3.1.1 Quadratic programming (QP-SVR)

As the development of the SVR method is very similar to the one exposed in chapter 2 for classification, the nonlinear case is directly considered in the following. The nonlinear model is obtained by replacing the sample vector $\boldsymbol{x}$ in (3.1) by its image in feature space $\boldsymbol{\Phi}(\boldsymbol{x})$. This leads to

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x})\rangle + b. \tag{3.5}$$

In the original SVR algorithm, the ε-insensitive loss function (3.4) is implemented by adding slack variables ξ_i and ξ_i^* to the constraints (3.2), and the ℓ_2-norm of the weights $\boldsymbol{w}$ is minimized

[1] The ε-insensitive loss function is originally defined in [275] as $|y - f(\boldsymbol{x})|_\varepsilon = \varepsilon$, if $|y - f(\boldsymbol{x})| \leq \varepsilon$, and $|y - f(\boldsymbol{x})|$, otherwise. However, here we use the more popular definition from [244], equivalent for the algorithm.

in order to maximize the flatness. The problem can thus be written in its QP form as

$$\begin{aligned}\min_{\boldsymbol{w},b,\xi_i,\xi_i^*} \ & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N}(\xi_i+\xi_i^*) \\ \text{s.t.}\ & y_i - \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle - b \leq \varepsilon + \xi_i, \quad i=1,\dots,N \\ & \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b - y_i \leq \varepsilon + \xi_i^*, \quad i=1,\dots,N \\ & \xi_i \geq 0,\ \xi_i^* \geq 0, \quad i=1,\dots,N.\end{aligned} \tag{3.6}$$

The solution is found by minimizing the Lagrangian

$$\begin{aligned}L =& \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}(\xi_i+\xi_i^*) - \sum_{i=1}^{N}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ & - \sum_{i=1}^{N}\alpha_i(\varepsilon + \xi_i - y_i + \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b) - \sum_{i=1}^{N}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle - b),\end{aligned} \tag{3.7}$$

where the Lagrange multipliers $\eta_i,\ \eta_i^* \geq 0$ and $\alpha_i,\ \alpha_i^* \geq 0$ are associated respectively to the postivity constraints on $\xi_i,\ \xi_i^*$ and the constraints on the data points. Computing the derivatives of L with respect to the primal variables $\boldsymbol{w}$, b, ξ_i, and ξ_i^* yields

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{N}(\alpha_i^* - \alpha_i)\boldsymbol{\Phi}(\boldsymbol{x}_i) = 0 \tag{3.8}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0 \tag{3.9}$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \eta_i = 0 \tag{3.10}$$

$$\frac{\partial L}{\partial \xi_i^*} = C - \alpha_i^* - \eta_i^* = 0. \tag{3.11}$$

The equations (3.10) and (3.11) allow to remove the variables $\eta_i = C - \alpha_i$ and $\eta_i^* = C - \alpha_i^*$, while the weights are recovered by (3.8) as

$$\boldsymbol{w} = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\boldsymbol{\Phi}(\boldsymbol{x}_i). \tag{3.12}$$

Introducing these results in (3.7) gives the dual Lagrangian L_D that must be maximized w.r.t. the dual variables [244]:

$$\begin{aligned}\max_{\alpha_i,\alpha_i^*} \ & L_D = -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)k(\boldsymbol{x}_i,\boldsymbol{x}_j) - \varepsilon\sum_{i=1}^{N}(\alpha_i+\alpha_i^*) + \sum_{i=1}^{N}y_i(\alpha_i-\alpha_i^*) \\ \text{s.t.}\ & \sum_{i=1}^{N}(\alpha_i-\alpha_i^*) = 0 \\ & 0 \leq \alpha_i \leq C,\ i=1,\dots,N \\ & 0 \leq \alpha_i^* \leq C,\ i=1,\dots,N,\end{aligned} \tag{3.13}$$

where the kernel function $k(\boldsymbol{x}_i,\boldsymbol{x}_j) = \langle \boldsymbol{\Phi}(\boldsymbol{x}_i), \boldsymbol{\Phi}(\boldsymbol{x}_j)\rangle$ has been introduced to replace the inner products. As for classification, kernels are used to deal with nonlinear problems while avoiding the curse of dimensionality. The Karush-Kuhn-Tucker (KKT) conditions, stating that the products between the dual variables and the constraints vanish at the solution, can be written

$$\alpha_i(\varepsilon + \xi_i - y_i + \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b) = 0 \tag{3.14}$$

$$\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle - b) = 0 \tag{3.15}$$

$$(C - \alpha_i)\xi_i = 0 \tag{3.16}$$

$$(C - \alpha_i^*)\xi_i^* = 0. \tag{3.17}$$

From these, the bias b can be computed, for instance by (3.14) with (3.12). Indeed, for a particular point $(\boldsymbol{x_i},\ y_i)$, ξ_i is known and α_i, α_i^* are given by the solution of the optimization. Usually an average over several points is used. The bias can equivalently be computed from the active constraints of (3.6) that convert to equalities.

By using (3.12) in (3.5) and replacing the inner product $\langle \boldsymbol{\Phi}(\boldsymbol{x}), \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle$ by the kernel function, the SVR model is finally given by

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) k(\boldsymbol{x}_i, \boldsymbol{x}) + b. \tag{3.18}$$

It can be noticed that, since (3.16) and (3.17) hold, only the points for which the corresponding Lagrange multipliers satifsy $\alpha_i = C$ or $\alpha_i^* = C$ can stand outside of the tube of insensitivity (i.e. $\xi_i, \xi_i^* \neq 0$). Besides, for the points inside the tube ($|y_i - f(\boldsymbol{x_i})| < \varepsilon$ and $\xi_i, \xi_i^* = 0$), the conditions (3.14) and (3.15) imply that α_i and α_i^* are zero. This leads to one of the key features of SVR: sparsity. Only the samples with non-zero α_i, α_i^* are required to compute the model output (3.18). These samples are called *Support Vectors* (SVs) and usually represent only a small percentage of the training data.

3.1.2 Linear programming (LP-SVR)

In kernel regression, the model is assumed to be in the form of a kernel expansion over all the training samples. In matrix form, the model can be written as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\boldsymbol{\alpha} + b, \tag{3.19}$$

where $\boldsymbol{\alpha}$ and b are the parameters of the model and $\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T) = [k(\boldsymbol{x}, \boldsymbol{x}_1)\ k(\boldsymbol{x}, \boldsymbol{x}_2)\ \dots\ k(\boldsymbol{x}, \boldsymbol{x}_N)]$, as defined in the notations (page XXI).

In kernel regression via linear programming (LP), the ℓ_1-norm is used both for regularization and as a loss function. This leads to

$$\min_{\boldsymbol{\alpha}, b} \|\boldsymbol{\alpha}\|_1 + C \sum_{i=1}^{N} |y_i - f(\boldsymbol{x}_i)|, \tag{3.20}$$

where a trade-off parameter C is introduced to tune the effect of the regularization w.r.t. the accuracy to the data. As for classification (section 2.1.2), this problem can be implemented as a linear program by introducing two sets $\boldsymbol{a}$ and $\boldsymbol{\xi}$ of N positive variables as in

$$\begin{aligned} &\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi} \geq \mathbf{0}, \boldsymbol{a} \geq \mathbf{0}} \quad \mathbf{1}^T\boldsymbol{a} + C\mathbf{1}^T\boldsymbol{\xi} \\ &\text{s.t.} \quad \begin{array}{rcl} -\boldsymbol{\xi} \leq & \boldsymbol{K}\boldsymbol{\alpha} + b\mathbf{1} - \boldsymbol{y} & \leq \boldsymbol{\xi} \\ -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a}. \end{array} \end{aligned} \tag{3.21}$$

where the kernel matrix $\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{X}^T, \boldsymbol{X}^T)$ and the target vector $\boldsymbol{y}$ are defined as in the notations of page XXI.

Instead of the ℓ_1-norm of the errors, the ε-insensitive loss function (3.4) can also be used to yield the Linear Programming SVR (LP-SVR) algorithm. A possible formulation of the corresponding problem involves $4N + 1$ variables [246]. Here, we follow the approach of [181] that involves only $3N + 1$ variables. In this scheme, the optimization problem becomes

$$\begin{aligned} &\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi}, \boldsymbol{a} \geq \mathbf{0}} \quad \mathbf{1}^T\boldsymbol{a} + C\mathbf{1}^T\boldsymbol{\xi} \\ &\text{s.t.} \quad \begin{array}{rcl} -\boldsymbol{\xi} \leq & \boldsymbol{K}\boldsymbol{\alpha} + b\mathbf{1} - \boldsymbol{y} & \leq \boldsymbol{\xi} \\ 0 \leq & \mathbf{1}\varepsilon & \leq \boldsymbol{\xi} \\ -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a}. \end{array} \end{aligned} \tag{3.22}$$

Similar to classification, the LP formulation of SVR does not require a kernel satisfying Mercer's condition [179].

Remark 5. *In LP-SVR, the form of the model is* assumed *from the start, whereas, for QP-SVR, it corresponds to the solution derived from the dual formulation of the problem. This is one of the reasons why formulating the dual form of the LP-SVR problem does not help in solving it.*

Remark 6. *As mentioned at the beginning of this chapter, in practice ℓ_1-norm regularization leads to an increased sparsity of support vectors compared to ℓ_2-norm regularization [299, 21, 246]. This idea will be used for feature selection in chapter 5.*

3.1.3 Automatic tuning of the insensitivity tube

The parameter ε of the ε-insensitive loss function, related to the width of the insensitivity tube, can be tuned by considering the noise level of the data. However, in many real-world applications, the noise level is unknown. A technique, called ν-SVR, allows to automatically tune the threshold ε during the SVR training. It consists in adding a term in the optimization problem in order to minimize ε. This approach can be interpreted as searching for a model with minimum complexity that can minimize the error measured by the most sensitive loss function.

This technique, presented for QP-SVR and LP-SVR below, will be extended to hybrid system identification in chapter 5.

Quadratic programming: ν-SVR

The problem (3.6) is rewritten as

$$\begin{aligned} \min_{\boldsymbol{w}, b, \xi_i \geq 0, \xi_i^* \geq 0, \varepsilon} \ & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\left(\sum_{i=1}^{N}(\xi_i + \xi_i^*) + N\nu\varepsilon\right) \qquad (3.23) \\ \text{s.t. } & y_i - \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle - b \leq \varepsilon + \xi_i, \ i = 1, \dots, N \\ & \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b - y_i \leq \varepsilon + \xi_i^*, \ i = 1, \dots, N, \\ & \xi_i \geq 0, \ \xi_i^* \geq 0, \quad i = 1, \dots, N, \end{aligned}$$

with the tuning constant $\nu > 0$. The dual problem is given by

$$\begin{aligned} \max_{\alpha_i, \alpha_i^*} \ & L_D^\nu = -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*) \qquad (3.24) \\ \text{s.t. } & \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \\ & 0 \leq \alpha_i \leq C, \ i = 1, \dots, N \\ & 0 \leq \alpha_i^* \leq C, \ i = 1, \dots, N \\ & \sum_{i=1}^{N}(\alpha_i + \alpha_i^*) \leq C\nu N. \qquad (3.25) \end{aligned}$$

The optimization program that automatically tunes ε is thus very similar to the original program (3.13) but with a missing term in L_D^ν and a supplementary constraint (3.25). The tuning of ε can thus be obtained by a convex quadratic program, which uses a new hyperparameter ν. This hyperparameter, if chosen such that $0 < \nu \leq 1$, can be interpreted both as the percentage of errors and as the percentage of SVs at the end of the training, as exposed in [233]. This paper also gives the asymptotic optimal choice for ν when the noise distribution type (Gaussian, Laplacian,...) is known.

So far, the width of the tube is constant over the whole input space. Note that a possible extension has been proposed in [233] to consider the insensitivity zone no longer as a tube but as provided by a parametric model.

Linear programming: μ-LPSVR

The parameter ε can be introduced as a variable in the objective function of the LP-SVR (3.22) to be tuned automatically by the algorithm. In [181], the following linear program is proposed[2]:

$$\begin{aligned}
\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi}, \boldsymbol{a} \geq \mathbf{0}, \varepsilon} \quad & \mathbf{1}^T\boldsymbol{a} + \mathbf{1}^T\boldsymbol{\xi} - CN\mu\varepsilon \\
\text{s.t.} \quad & \begin{array}{rcl} -\boldsymbol{\xi} \leq & \boldsymbol{K\alpha} + b\mathbf{1} - \boldsymbol{y} & \leq \boldsymbol{\xi} \\ 0 \leq & \mathbf{1}\varepsilon & \leq \boldsymbol{\xi} \\ -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a}. \end{array}
\end{aligned} \tag{3.26}$$

It can be shown [181] that this formulation is equivalent to the ν-Linear Programming Regression (ν-LPR) proposed in [246] with a relationship between μ and ν.

3.1.4 Theory and related approaches

This section places support vector regression in three frameworks that provide some insight into its properties: statistical learning theory, maximum likelihood estimation and robust statistics. As a complete description of these frameworks would be out of the scope of the thesis, the following extracts the main ideas and provides pointers to relevant references including all the omitted details. The concluding paragraph will summarize the main advantages of the SVR approach.

Statistical learning theory: from classification to regression

Statistical learning theory has originally been developed for classification purposes. However, the same framework can be used to derive generalization error bounds similar to (2.27) for real-valued models in a function approximation setting.

For the squared loss function $l(f(\boldsymbol{x}), y) = (y - f(\boldsymbol{x}))^2$, the following bound linking the generalization error (or expected risk) $R[f]$ (1.2) to the empirical error $E_S[f]$ (1.3) holds with probability $1 - \delta$ [48]:

$$R[f] \leq E_S[f] \left(1 - c\sqrt{\frac{h(\ln(aN/h) + 1) - \ln\delta}{N}}\right)_+^{-1}, \tag{3.27}$$

where N is the size of the training set S, h is the VC dimension of the set $\mathcal{F}$ of admissible functions f, c is a constant reflecting the tails of the loss function distribution (i.e. the probability of observing large values of the loss), a is a theoretical constant (that can be set to a value close to 1 [276]), $(z)_+$ is z if $z > 0$ and 0 otherwise. This bound decreases with the complexity of the model, measured by the VC dimension, and when the number N of data increases. Note that this bound can be related to classical criteria in statistics such as the Finite Prediction Error (FPE) or the Minimum Description Length (MDL), see [251] or [276] for details.

In [62], bounds for the ε-insensitive loss function are derived from margin-based classification bounds such as (2.28). The main idea is to consider that a test error (as defined in classification) occurs if the error on a test sample (as defined in regression) is larger than a threshold test accuracy θ. Thus, the aim is to bound the probability that a randomly drawn test point will have accuracy less than θ. In this context, the margin in the regression accuracy $\gamma = \theta - \varepsilon$ (playing a similar role to the margin of a classifier) measures the allowed difference between the test accuracy θ and the training accuracy ε. A training error (as defined in classification) occurs if the corresponding error is greater than $\varepsilon = \theta - \gamma$. The resulting "soft-margin" bound holding with probability $1 - \delta$ can be written as

$$R_\theta[f] \leq \frac{c}{N}\left(\frac{\|\boldsymbol{w}\|_2^2 R^2 + \|\boldsymbol{\xi}\|_1^2 \log_2(1/\gamma)}{\gamma^2} \log_2^2 N + \log_2 \frac{1}{\delta}\right), \tag{3.28}$$

where $R_\theta[f]$ considers a θ-insensitive loss, $\boldsymbol{\xi}$ is the vector of slack variables evaluated on the training set with $\xi_i = |y_i - f(\boldsymbol{x}_i)|_\varepsilon$ and R is the radius of the minimal ball enclosing the data in feature space, i.e. for all $\boldsymbol{x}$, we have $\|\boldsymbol{\Phi}(\boldsymbol{x})\|_2 = \sqrt{\langle \boldsymbol{\Phi}(\boldsymbol{x}), \boldsymbol{\Phi}(\boldsymbol{x})\rangle} \leq R$. This bound decreases when the number of data N, the margin γ or its confidence level δ increase. It also clearly shows that $\|\boldsymbol{w}\|_2$ can be used as a regularizer for capacity control.

[2] In [181], the objective function is originally written as $1/N\mathbf{1}^T\boldsymbol{a} + C/N\mathbf{1}^T\boldsymbol{\xi} - C\mu\varepsilon$.

SVR in the Maximum Likelihood (ML) framework

The maximum likelihood (ML) approach consists in finding the function f that most likely generated the data. Thus, for N i.i.d. samples, the aim is to maximize the functional $p(\boldsymbol{x}, y|f)$ w.r.t. f, leading to

$$\max_f p(\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}, \{y_1, \dots, y_N\}|f) \Leftrightarrow \max_f \prod_{i=1}^N p(\boldsymbol{x}_i, y_i|f) \Leftrightarrow \max_f \prod_{i=1}^N p(y_i|\boldsymbol{x}_i, f)p(\boldsymbol{x}_i). \quad (3.29)$$

Taking the log of the likelihood, the products convert into sums that can then be conveniently maximized. Furthermore, as the terms depending on $p(\boldsymbol{x}_i)$ are fixed, they can be dropped from the maximization problem. As minimization is preferred to maximization, the ML estimator is finally defined as the minimizer of the *log-likelihood* given by

$$J^{ML} = \sum_{i=1}^N - \ln p(y_i|\boldsymbol{x}_i, f). \quad (3.30)$$

Implicit noise model for SVR. Consider the estimator minimizing the ε-insensitive loss function,

$$\min_f \sum_{i=1}^N |y_i - f(\boldsymbol{x}_i)|_\varepsilon. \quad (3.31)$$

This estimator can be interpreted in the ML framework by considering the implicit noise density model given by

$$p(y_i|\boldsymbol{x}_i, f) = \frac{1}{2(1+\varepsilon)} \exp(-|y_i - f(\boldsymbol{x}_i)|_\varepsilon), \quad (3.32)$$

which is plotted on Figure 3.2. In other words, solving (3.31) is equivalent to minimizing J^{ML} (3.30) with (3.32).

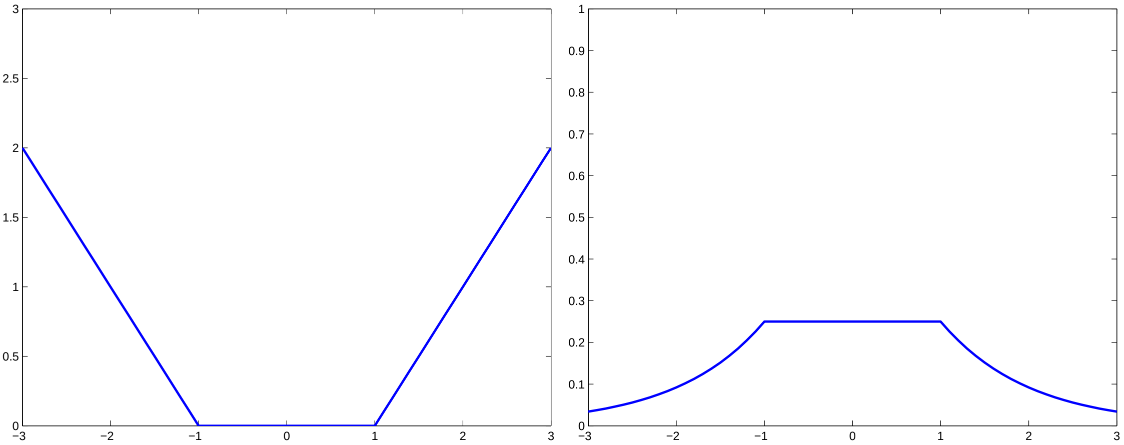


Figure 3.2: The ε-insensitive loss function (*left*) and its corresponding noise density model (*right*) plotted w.r.t. the error $y - f(\boldsymbol{x})$ for $\varepsilon = 1$.

Support vector regression can be seen as a regularized version of the ε-insensitive loss minimizer.

Asymptotic efficiency. In the context of parameter estimation (see remark 2 on page 6), the asymptotic efficiency of a single parameter unbiased estimator, given by

$$\text{eff} = \frac{1}{IB}, \quad (3.33)$$

measures how close the variance B of the estimator is to the minimal achievable variance, given by the Cramér–Rao bound $B \geq 1/I$, where I is the Fisher information. If the Cramér–Rao bound is attained, the estimator is said to be asymptotically efficient and $\text{eff} = 1$.

As shown in [232], for Gaussian noise with zero mean and variance σ, the efficiency of an estimator using the ε-insensitive loss function is

$$\text{eff} = \frac{1}{\exp\left(\frac{\varepsilon^2}{\sigma^2}\right)\left(1 - \text{erf}\frac{\varepsilon}{\sqrt{2}\sigma}\right)}. \tag{3.34}$$

In this case, the efficiency is maximized for $\varepsilon = 0.612\sigma$, which provides an asymptotically optimal value for the ε hyperparameter in the SVR method. Of course, the exact noise model and noise variance σ are usually unknown. However, the optimal value of ε (in terms of asymptotic efficiency) can be shown to scale linearly with σ for any symmetric noise density [232]. Moreover, making ε adaptive as in the ν-SVR method of section 3.1.3 allows to extend this result to a form that does not depend on σ.

Note that these theoretic results consider the asymptotic estimation of a one-parametrical model, which is a gross oversimplification for SVR, where a nonparametric function is estimated from a limited number of observations [232]. However, these results have been verified in extensive experiments [36], also indicating that there is an optimal area rather than a sharp optimum for ν.

Robustness to outliers

Robust statistics [119, 222] studies the behavior of estimators in the presence of outliers in the data. In this context, the maximal influence of a single data point is crucial.

Influence function. For M-estimators minimizing $\sum_{i=1}^N l(e_i)$, where $e_i = y_i - f(\boldsymbol{x}_i)$, this influence can be measured by the influence function, i.e. the derivative $l'(e_i) = dl(e_i)/de_i$. For instance, the least squares estimator (1.18), known to behave poorly w.r.t. outliers, has an influence function growing linearly with the error: $l'(e_i) = e_i$. In this case, the effect of a single data point can thus be arbitrary large. On the other hand, SVR minimizes the ε-insensitive loss, which leads to an influence function taking only three values, $l'(e_i) = -1$, 0 or 1, and thus bounded by 1.

Breakdown point. For a number N_{out} of arbitrary outliers in a training set of size N, the breakdown point [222] of an estimator is defined as the minimum ratio N_{out}/N leading to an infinite bias. For instance, as a single outlier can have an arbitrary influence on the least squares estimator, its breakdown point is $1/N$. Using the ε-insensitive loss allows to obtain a higher breakdown point w.r.t. outliers in y. However, as for ℓ_1-norm based regression, the breakdown point taking into account outliers in the input $\boldsymbol{x}$ (leverage points) is still $1/N$.

ν-Support vector regression. In [233], a link between ν-SVR and robust estimators is stated by showing that a change in the value y_i for a point outside the insensitivity tube (i.e. $|y_i - f(\boldsymbol{x}_i)| > \varepsilon$) does not influence the solution. In this context, the hyperparameter ν can be related to the breakdown point of the corresponding robust estimator.

Other related approaches

Support Vector Regression can be interpreted in the framework of Reproducing Kernel Hilbert Spaces (RKHS). Indeed, for a particular RKHS $\mathcal{H}$, there is a unique reproducing kernel k satisfying $\langle f, k(\boldsymbol{x}, \cdot)\rangle_{\mathcal{H}} = f(\boldsymbol{x})$, $\forall f \in \mathcal{H}$ (see e.g. [79] for an overview on RKHS). In the context of RKHS, the representer theorem, originally introduced in [143] and then extended to non-quadratic loss functions in [60] and general regularizers in [229], states that solutions to a class of regularized optimization problems can be expressed as a finite kernel expansion over the training samples only (which is the form of the SVR model[3] (3.18)). Using the RKHS framework, SVR has been shown to be equivalent to regularization networks [79].

[3] See [229] for comments on how to include the bias term in the solution.

In the Bayesian framework, one defines explicitly the regularizer for the the Maximum *a posteriori* (MAP) estimator on the basis of a prior density $p(\boldsymbol{w})$ on the model parameters. In particular, for a Gaussian prior $p(\boldsymbol{w})$, the Bayesian approach leads to the regularizer $\lambda\|\boldsymbol{w}\|_2$, where λ is a constant. In this case, the MAP estimator can be shown to be equivalent to the QP-SVR [154] (see also [50] and references therein for a unified presentation of SVR in a Bayesian framework). For Vapnik [275], using a prior $p(\boldsymbol{w})$ can be seen as imposing a form of capacity control, which explains why Bayesian inference works even when the true function does not satisfy the prior.

Conclusions

To summarize, Support Vector Regression has the following features.

- SVR is based on statistical learning theory, that has shown its efficiency in the classification framework, and that provides non-asymptotic bounds on the generalization error. The theory shows that capacity (or complexity) control, included in SVR learning, is crucial. This can also be seen as intrinsic regularization and allows to *generalize from few data.*
- SVR uses kernels to deal with nonlinear function approximation, which allow the models to generalize in high dimensional spaces and limit the number of parameters to $N + 1$.
- SVR uses a robust loss function w.r.t. to outliers in the output value y.
- SVR amounts to solve a convex optimization program and can thus be implemented very efficiently with the guarantee of finding a unique solution. Note however that the general problem of training a SVM on a particular task is not convex w.r.t. the hyperparameters.
- SVR leads to sparse nonlinear models with automatic selection of the basis functions (obtained here by the selection of the support vectors).

In addition, the ν-SVR method provides some interpretations for its hyperparameter ν, related to the number of SVs, the asymptotic efficiency w.r.t. the noise density, and the breakdown point of the estimator.

3.2 Least Squares SVM for regression

As for classification (see section 2.2), the Least Squares Support Vector Machine (LS-SVM) allows to simplify the SVR problem to a system of linear equalities by considering a square loss function. However, the sparsity, which was originally obtained from the ε-insensitive loss function, is lost. In addition, due to the use of the square loss, the method is sensitive to outliers.

This section starts with the description of the algorithm before exposing available methods to recover sparsity and robustness. At the end, a note will discuss the links between the LS-SVM and other related methods.

3.2.1 Algorithm (LS-SVM)

For nonlinear regression, the Least Squares SVM [251] is derived by replacing the ε-insensitive loss function in the SVR problem by the squared loss function (1.14). This leads to the optimization problem

$$\begin{aligned} \min_{\boldsymbol{w},b,e_i} \ & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\frac{1}{2}\sum_{i=1}^{N} e_i^2 \\ \text{s.t. } \ & y_i = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b + e_i,\ i = 1,\dots,N. \end{aligned} \tag{3.35}$$

Since the LS-SVM does not include any insensitive zone, no inequality constraint is required, only equalities. The corresponding Lagrangian involving the dual variables α_i is given by

$$L = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\frac{1}{2}\sum_{i=1}^{N} e_i^2 - \sum_{i=1}^{N} \alpha_i[\langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b + e_i - y_i]. \tag{3.36}$$

Canceling the derivatives with respect to the primal variables yields

$$\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{w}} &= \boldsymbol{w} - \sum_{i=1}^{N} \alpha_i \boldsymbol{\Phi}(\boldsymbol{x}_i) = \boldsymbol{0} \;\Rightarrow\; \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \boldsymbol{\Phi}(\boldsymbol{x}_i) && (3.37)\\
\frac{\partial L}{\partial b} &= \sum_{i=1}^{N} \alpha_i = 0 && (3.38)\\
\frac{\partial L}{\partial e_i} &= Ce_i - \alpha_i = 0 \;\Rightarrow\; \alpha_i = Ce_i,\; i = 1, \dots, N. && (3.39)\\
& && (3.40)
\end{aligned}$$

As for classification, (3.37) and (3.39) allow to replace $\boldsymbol{w}$ and e_i in the constraints of (3.35) to yield

$$y_i = \sum_{j=1}^{N} \alpha_j \langle \boldsymbol{\Phi}(\boldsymbol{x}_j), \boldsymbol{\Phi}(\boldsymbol{x}_i) \rangle + b + \frac{1}{C}\alpha_i,\; i = 1, \dots, N. \tag{3.41}$$

Finally, by using the above in addition to (3.38), the parameters are recovered as the solution of the linear system of equations

$$\begin{bmatrix} \boldsymbol{K} + \frac{1}{C}\boldsymbol{I} & \boldsymbol{1} \\ \boldsymbol{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{y} \\ 0 \end{bmatrix}, \tag{3.42}$$

where $\boldsymbol{K}$ is the kernel matrix. The model is then given by

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b. \tag{3.43}$$

3.2.2 Sparse LS-SVM (SLS-SVM)

As seen above, the use of the squared loss function does not directly yield sparsity. However, the following iterative procedure has been proposed in [250] to reduce the number of support vectors by pruning the parameters.

1. Train a standard LS-SVM by solving (3.35).
2. Remove from the training set a small amount of samples with smallest values in the sorted $|\alpha_i|$ "spectrum".
3. Retrain a new LS-SVM (the final Sparse LS-SVM) on the reduced training set.
4. Go to step 2 until a user-defined performance index degrades.

Note that this technique can be similarly applied to the LS-SVM for classification of section 2.2.

Remark 7 (Fixed-size LS-SVM)**.** *Another approach has been proposed in [78] to impose sparsity in the LS-SVM model. Instead of pruning the resulting model, this method selects a subset of samples at the beginning of the training by maximizing a suitable entropy criterion. Then the problem is considered in its primal form (3.35) where the nonlinear mapping Φ is approximated by a finite-dimensional mapping obtained from the eigendecomposition of the small kernel matrix computed on the subset of samples.*

3.2.3 Robust LS-SVM (RLS-SVM)

An iterative procedure has been proposed in [249] in order to deal with the presence of outliers in the data. Instead of considering a standard robust loss function, the Robust LS-SVM (RLS-SVM) works bottom up and tries to tune an appropriate loss function from the data by weighting the error variables e_i in the problem that now becomes

$$\begin{aligned} \min_{\boldsymbol{w},b,e_i} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\frac{1}{2}\sum_{i=1}^{N} v_i e_i^2 \\ \text{s.t. } y_i = \langle \boldsymbol{w}, \boldsymbol{\Phi}(\boldsymbol{x}_i)\rangle + b + e_i,\ i = 1,\dots,N, \end{aligned} \tag{3.44}$$

where v_i is the weight associated with e_i. Three steps are required in the procedure.

1. Train a standard LS-SVM by solving (3.35).

2. Estimate the weights v_i, for instance by

$$v_i = \begin{cases} 1, \text{ if } |e_i/\hat{s}| \leq c_1 \\ \dfrac{c_2 - |e_i/\hat{s}|}{c_2 - c_1}, \text{ if } c_1 \leq |e_i/\hat{s}| \leq c_2 \\ 10^{-4}, \text{ otherwise,} \end{cases} \tag{3.45}$$

   where $\hat{s}$ is a robust estimate of the standard deviation of the error variables e_i, e.g. based on the interquartile range (IQR) or the median absolute deviation from the median (MAD) such as

$$\hat{s} = \frac{\text{IQR}}{2 \times 0.6745} \quad \text{or} \quad \hat{s} = 1.483 \times \text{MAD}(e_i), \tag{3.46}$$

   for a Gaussian noise.

3. Train a weighted LS-SVM by solving (3.44).

The unweigthed LS-SVM of step 1 considers Gaussian noise, while the steps 2 and 3 are used to correct the distribution in the case where it is not normal. The procedure can be iterated but one pass is often sufficient in practice [249].

3.2.4 Related methods

Looking at the problem formulation (3.35), one may recognize a regularized least squares problem, in which the data are first preprocessed by a nonlinear mapping Φ. This problem, also known as ridge regression, was originally studied in [226], but without a bias term.

In the standard SVM approach, support vectors are provided by the ε-insensitive loss function. The name "support vectors" comes from the fact that this subset of data contains the required amount of information to solve the problem and that these vectors actually support the model. In LS-SVM, the support vectors still support the model, but no selection of the most relevant SVs is performed by the standard algoritm. However, two common properties of the SVM and the LS-SVM can be highlighted:

- The use of kernels and dual representation
- The use of regularization by minimizing the norm of the weights $\|\boldsymbol{w}\|_2$

Table 3.1 shows the links and differences between the QP-SVR, LP-SVR and LS-SVM algorithms. In [251], relationships between LS-SVM for regression and other methods, including Gaussian processes, regularization networks and kriging, are detailed.

3.3 Numerical examples

As seen in chapter 1, system identification amounts in particular cases (for ARX based models) to solving a regression problem. Thus, all the methods presented in this chapter also apply to system identification. The following gives some experimental results in such a setting, which show that all SVM based algorithms perform roughly the same on average. However, the differences will be

Table 3.1: Comparison between the Support Vector Regression algorithms, where $e_i = y_i - f(\boldsymbol{x}_i)$.

	QP-SVR	LP-SVR	LS-SVM
loss function	$\|e_i\|_\varepsilon$	$\|e_i\|_\varepsilon$	$\frac{1}{2}e_i^2$
noise density model	$\frac{1}{2(1+\varepsilon)}\exp(-\|e_i\|_\varepsilon)$	$\frac{1}{2(1+\varepsilon)}\exp(-\|e_i\|_\varepsilon)$	$\frac{1}{2\sigma}\exp\left(-\frac{e_i^2}{2\sigma^2}\right)$
regularizer	$\\|\boldsymbol{w}\\|_2^2$	$\\|\boldsymbol{\alpha}\\|_1$	$\\|\boldsymbol{w}\\|_2^2$
optim. problem class	QP	LP	Linear System of Eq.
admissible kernels	Mercer	any	Mercer
sparsity	automatic	automatic	by pruning

emphasized in Sect. 3.3.2, where the robustness of the algorithms w.r.t. outliers is studied on a function approximation problem.

SVR has also been applied to the reconstruction of chaotic systems and time-series prediction in [186, 194] and in [255] for LS-SVM. However, these studies considered larger training sets, whereas here the aim is to study the performance of SVR on the basis of few training samples.

Note that an exhaustive comparison of SVR with other nonlinear modeling methods (such as the ones mentioned in Sect. 1.5) is out of the scope of this part, which only aims at illustrating some theoretical properties of the different SVR algorithms and test them in the non-asymptotic case or in the presence of outliers.

3.3.1 A system identification example

This example studies the case where a small amount of data is available. Experiments in a similar setting were presented in [148].

Data. Consider the nonlinear dynamical system from the illustrative example 2 of section 1.4, also used for benchmark purposes in [45, 34, 39],

$$y_k = \left(0.8 - 0.5\exp(-y_{k-1}^2)\right) y_{k-1} - \left(0.3 + 0.9\exp(-y_{k-1}^2)\right) y_{k-2} + 0.1\sin(\pi y_{k-1}) + e_k, \tag{3.47}$$

where e_k stands for a zero mean noise.

In order to show the generalization capacity of SVR, a noisy trajectory of 130 points y_i is generated from the system (3.47) with initial conditions $y_0 = \ y_{-1} = 0$. The regression vector with two lagged outputs $\boldsymbol{x}_i = [y_{i-1}\ y_{i-2}]^T$ is considered and the last 30 points are retained for validation of the hyperparameters (validation set). Four training sets of sizes $N = 20$, 34, 50 and 100 are then built by taking one point out of 5, 3, 2 and 1 from the remaining data $(\boldsymbol{x}_i, y_i)$, $i = 1, \dots, 100$. Tests are performed with three different noise distributions for e_k: Gaussian noise of standard deviation 0.1, uniform noise in the interval $e_k \in [-0.4, 0.4]$ and Laplacian noise of standard deviation 0.3.

Methods. The comparison is performed between the following methods using standard implementations[4]:

- quadratic programming SVR (QP-SVR, see Sect. 3.1.1),
- linear programming SVR (LP-SVR, see Sect. 3.1.2),
- ν-QPSVR (see Sect. 3.1.3),
- μ-LPSVR (see Sect. 3.1.3),
- Least Squares SVM (LS-SVM, see Sect. 3.2), and
- Sparse LS-SVM (SLS-LSM, see Sect. 3.2.2).

[4] Standard Matlab toolboxes are used: LibSVM [40] for QP-SVR and ν-QPSVR, Matlab optimization toolbox for LP-SVR and μ-LPSVR, LS-SVMlab 1.5 [249] for LS-SVM and SLS-SVM.

Hyperparameters. All these algorithms use Gaussian RBF kernels with the width σ heuristically tuned on the training data as the radius of the minimum enclosing ball: $\sigma = \max_{j=1,2}(\max_{i=1,\dots,N} x_i^j - \min_{i=1,\dots,N} x_i^j)/2$. For all training set sizes N, the hyperparameter C of the SVM-based methods is tuned on the validation set with values in the range $C \in \{1, 2, 3, 5, 7, 10, 15, 20, 50, 100, 200, 300, 500\}$. The threshold on the ε-insensitive loss functions used by the QP-SVR and LP-SVR algorithms is tuned in a similar manner in the range of values $\varepsilon \in \{0.001, 0.01, 0.02, 0.05, 0.08, 0.1, 0.2\}$ for Gaussian noise and in $\{0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4\}$ for uniform and Laplacian noise. For the automatic tuning methods ν-QPSVR and μ-LPSVR, ν and μ are tuned in the range $\{0.1, 0.2, 0.3, \dots, 1\}$. The LS-SVM model is pruned with default parameters[5] to yield the SLS-SVM model.

Test. The test is based on the one-step-ahead prediction error evaluated over $N_t = 100$ test points by (1.15), where y_i is the output of the system (3.47) without noise and $f(\boldsymbol{x}_i)$ is the predicted output. The test points are generate with the initial conditions $y_0 = y_{-1} = 0.1$. Table 3.2 shows the results averaged over 100 trials with different noise sequences. The minimal test error[6] appears in boldface. However the choice of the "best" model should also consider the number of support vectors.

Table 3.2: Averages and standard deviations of the number of SVs (*above*) and of the one-step-ahead prediction error MSE×10^3 (*below*) of the models using RBF kernels with parameter σ and trained on a varying number N of samples corrupted by Gaussian, uniform or Laplacian noise.

Gaussian noise

	QP-SVR	LP-SVR	ν-QPSVR	μ-LPSVR	LS-SVM	SLS-SVM
$N = 100$	69 ± 30	61 ± 46	65 ± 29	60 ± 45	100	49 ± 10
$\sigma = 1.233$	1.48 ± 0.71	1.41 ± 0.61	1.42 ± 0.58	1.49 ± 0.57	$\mathbf{1.22 \pm 0.52}$	4.40 ± 1.88
$N = 50$	35 ± 12	22 ± 19	34 ± 13	25 ± 20	50	31 ± 6
$\sigma = 1.233$	2.41 ± 1.06	2.30 ± 1.03	$\mathbf{2.26 \pm 0.97}$	2.31 ± 0.98	2.24 ± 1.10	5.20 ± 2.60
$N = 34$	21 ± 8	11 ± 7	20 ± 8	11 ± 8	34	23 ± 3
$\sigma = 1.193$	3.43 ± 2.36	3.01 ± 2.08	3.00 ± 1.70	$\mathbf{2.75 \pm 1.34}$	3.24 ± 2.06	6.71 ± 4.17
$N = 20$	15 ± 4	7 ± 2	14 ± 4	7 ± 3	20	16 ± 1
$\sigma = 0.968$	51.06 ± 61.38	46.35 ± 52.33	46.07 ± 55.30	$\mathbf{43.01 \pm 49.84}$	71.76 ± 78.81	181.11 ± 189.26

Uniform noise

	QP-SVR	LP-SVR	ν-QPSVR	μ-LPSVR	LS-SVM	SLS-SVM
$N = 100$	36 ± 26	28 ± 39	38 ± 23	46 ± 46	100	40 ± 15
$\sigma = 1.418$	173.87 ± 29.45	177.92 ± 30.36	$\mathbf{171.94 \pm 30.95}$	178.69 ± 34.36	173.03 ± 27.02	192.28 ± 60.91
$N = 50$	22 ± 13	10 ± 10	19 ± 10	19 ± 20	50	26 ± 8
$\sigma = 1.388$	$\mathbf{176.15 \pm 33.68}$	180.61 ± 41.38	179.79 ± 39.31	182.82 ± 44.49	184.31 ± 39.79	224.33 ± 82.84
$N = 34$	19 ± 8	6 ± 2	19 ± 8	10 ± 10	34	21 ± 4
$\sigma = 1.233$	$\mathbf{195.80 \pm 37.95}$	206.70 ± 51.80	195.20 ± 42.35	205.83 ± 45.48	195.05 ± 50.96	222.95 ± 75.60
$N = 20$	13 ± 5	5 ± 2	12 ± 5	5 ± 2	20	13 ± 2
$\sigma = 1.289$	192.23 ± 47.68	206.32 ± 49.31	$\mathbf{185.41 \pm 48.33}$	192.66 ± 46.04	191.76 ± 51.31	226.54 ± 87.89

Laplacian noise

	QP-SVR	LP-SVR	ν-QPSVR	μ-LPSVR	LS-SVM	SLS-SVM
$N = 100$	63 ± 31	46 ± 45	60 ± 29	52 ± 46	100	40 ± 15
$\sigma = 1.740$	7.51 ± 6.63	7.53 ± 7.17	7.81 ± 7.14	7.74 ± 6.30	$\mathbf{7.41 \pm 4.51}$	22.05 ± 12.51
$N = 50$	63 ± 31	46 ± 45	60 ± 29	52 ± 46	100	40 ± 15
$\sigma = 1.682$	$\mathbf{11.38 \pm 7.18}$	12.30 ± 7.25	11.94 ± 7.47	12.59 ± 7.09	12.92 ± 8.05	30.09 ± 23.21
$N = 34$	22 ± 9	8 ± 3	23 ± 8	9 ± 7	34	21 ± 5
$\sigma = 1.616$	14.16 ± 9.76	14.93 ± 10.03	$\mathbf{13.61 \pm 8.14}$	15.23 ± 9.80	16.72 ± 9.95	32.64 ± 20.96
$N = 20$	13 ± 5	6 ± 2	14 ± 5	6 ± 2	20	14 ± 2
$\sigma = 1.532$	$\mathbf{28.62 \pm 24.51}$	30.89 ± 21.74	28.61 ± 24.81	29.67 ± 22.07	32.56 ± 25.84	76.16 ± 75.92

In these experiments, all the methods perform roughly as well as the others on average. However, the following remarks can be stated.

- *Regularization.* During these experiments, the minimum test error on the validation set is never obtained for the largest value of C. This illustrates the need for regularization to make

[5] The defaults parameters of the *sparselssvm* Matlab function [249] are 5 % of SVs removed at each iteration and 75 % tradeoff in model accuracy after pruning.

[6] The minimal test error is understood as the minimal [(average of MSE)^2 + (standard deviation of MSE)^2].

the model able to generalize. Note however that in a different context, e.g. in neural networks, regularization may be included by other means. For this, one can use early stopping during the optimization or impose a predefined structure to constrain the model capacity.

- *Asymptotic optimality.* Though being asymptotically optimal for Gaussian noise, the LS-SVM method does not lead to better results than the other non-optimal methods for $N \leq 50$. This illustrates the shortcomings of "asymptomania": asymptotic optimality usually does not hold in non-asymptotic scenarios. Note that this does not put into question the usefulness of asymptotic theories, but simply advocates the need for more robust and non-asymptotic ones.

- *Sparsity.* More support vectors does not mean better performance. For instance, using all the points as SVs, the LS-SVM does not always lead to the lowest test error. The automatic selection method intrinsically included in QP-SVR selects relevant SVs while leading to a low test error. In comparison, the Sparse LS-SVM algorithm yields models as sparse as the QP-SVR models, but degrades significantly the generalization performance. Note that, by tuning the pruning parameters of the SLS-SVM, the obtained models are much better in terms of test error, but are not as sparse as the QP-SVR models. Finally, the ℓ_1-norm regularization used by the LP-SVR algorithm leads to sparser models on average than the ℓ_2-norm regularization used in QP-SVR, as expected (see Sect. 3.1.2, remark 6). However, this is not guaranteed as the sparsity of LP-SVR models highly depends on the data for $N \geq 50$, as emphasized by the standard deviations of the number of SVs.

Comments. Note that the methods could also be compared on the basis of their computational complexity or computing time. Such a study is out of the scope of the present example, which only aimed at illustrating some theoretical properties of the methods. We refrain from giving here the computing times of the various algorithms, as their direct comparison would be irrelevant, due to the variability of the implementations (native Matlab code versus C compiled code, particular choice of toolboxes based on usability more than speed, and so on). However, independently of the implementation, it is clear that the LS-SVM algorithm is faster than the standard (either QP or LP) SVR algorithms. These latter suffer from a higher computational complexity and also have one more hyperparameter (either ε, ν or μ) to tune.

Asymptotically optimal hyperparameters

Similar experiments are now performed in order to determine the validity of the theoretic values for the hyperparameters of the SVR-based algorithms w.r.t. the number of training data N. The difference with the previous experiments is that the hyperparameters ε, ν and μ are set to their asymptotically optimal values. According to [232], considering Gaussian noise with variance β^2 leads to the threshold $\varepsilon = 0.612\beta^2$ and $\nu = 0.54$. μ is set to $1-\nu = 0.46$. The other hyperparameters (C and σ) are set as previously. Table 3.3 reports the results for $\beta = 0.1$. The results obtained previously by tuning ε, ν and μ on a validation set are also recalled in Table 3.3.

As expected, the asymptotical optimality of certain hyperparameter values does not hold for small numbers of samples. However, the optimality is verified for $N = 100$, in which case the theoretic values lead to lower test errors than values tuned on a validation set. Note that the optimal value of ε scales linearly with the variance of the noise and is thus very small compared to the noise standard deviation. For the QP-SVR and LP-SVR models, this leads to very large numbers of SVs and the loss of sparsity, since most of the points lie outside of the tube of insensitivity.

3.3.2 Robustness to outliers

Robustness of the QP-SVR, LP-SVR, LS-SVM and Robust LS-SVM (RLS-SVM, see Sect. 3.2.3) to outliers is studied on a one dimensional example: the approximation of the sinc function by RBF models. This example is here for illustrative purposes and therefore does not constitute an exhaustive study of the issue.

The data are generated as follows. An uncorrupted data set $\widehat{S}$ of $N = 100$ points in the range $-5 \leq x_i < 5$ with $y_i = \text{sinc}(x_i)$ is used. To create the corrupted training sets, an increasing

Table 3.3: One-step-ahead prediction error (MSE $\times 10^{-3}$) and number of SVs N_{SV} of the models trained with the main hyperparameter either tuned on a validation set or fixed to its asymptotically optimal value. Boldface indicates the minimum test error between these two settings for each model and each training set size N.

	Hyperparameter (ε, ν or μ)		QP-SVR	LP-SVR	ν-QPSVR	μ-LPSVR
$N = 100$ $\sigma = 1.177$	tuned	N_{SV}	69 ± 30	61 ± 46	65 ± 29	60 ± 45
		MSE	$\mathbf{1.48 \pm 0.71}$	1.41 ± 0.61	1.42 ± 0.58	1.49 ± 0.57
	asympt. optimal	N_{SV}	96 ± 2	77 ± 40	62 ± 3	75 ± 41
		MSE	1.55 ± 0.73	$\mathbf{1.34 \pm 0.62}$	$\mathbf{1.30 \pm 0.60}$	$\mathbf{1.38 \pm 0.64}$
$N = 50$ $\sigma = 1.177$	tuned	N_{SV}	69 ± 30	61 ± 46	65 ± 29	60 ± 45
		MSE	$\mathbf{2.41 \pm 1.06}$	$\mathbf{2.30 \pm 1.03}$	$\mathbf{2.26 \pm 0.97}$	$\mathbf{2.31 \pm 0.98}$
	asympt. optimal	N_{SV}	48 ± 1	24 ± 20	34 ± 2	34 ± 20
		MSE	2.89 ± 1.36	2.59 ± 1.20	2.41 ± 1.24	2.34 ± 1.16
$N = 34$ $\sigma = 1.161$	tuned	N_{SV}	21 ± 8	11 ± 7	20 ± 8	11 ± 8
		MSE	$\mathbf{3.43 \pm 2.36}$	$\mathbf{3.01 \pm 2.08}$	$\mathbf{3.00 \pm 1.70}$	$\mathbf{2.75 \pm 1.34}$
	asympt. optimal	N_{SV}	33 ± 1	12 ± 8	25 ± 2	18 ± 13
		MSE	4.13 ± 2.54	3.92 ± 2.01	3.42 ± 1.71	3.34 ± 1.65
$N = 20$ $\sigma = 0.649$	tuned	N_{SV}	15 ± 4	7 ± 2	14 ± 4	7 ± 3
		MSE	$\mathbf{51.06 \pm 61.38}$	$\mathbf{46.35 \pm 52.33}$	$\mathbf{46.07 \pm 55.30}$	$\mathbf{43.01 \pm 49.84}$
	asympt. optimal	N_{SV}	19 ± 1	6 ± 2	16 ± 2	6 ± 2
		MSE	66.49 ± 76.93	57.31 ± 63.51	67.63 ± 73.22	59.22 ± 65.24

number $N_{out} \in [0, N/2]$ of outliers is then introduced at random positions i by setting $y_i = -10$. The models trained on these corrupted data will be tested on the clean data set $\widehat{S}$ and evaluated w.r.t. an indicator defined as follows. The degradation point Δ_r at level r is defined as the minimal percentage of outliers leading to a test error increase of at least r% of the maximal increase obtained for $N_{out} = N$:

$$\Delta_r = \min \left\{ \frac{N_{out}^\star}{N} \;\middle|\; \frac{\text{MSE}(N_{out}) - \text{MSE}(0)}{\text{MSE}(N) - \text{MSE}(0)} > \frac{r}{100},\ \forall N_{out} \geq N_{out}^\star \right\}, \tag{3.48}$$

where MSE(N_{out}) denotes the test error obtained by a model trained on a data set corrupted by N_{out} outliers. Using this definition, $\Delta_{0.01}$ represents the ratio of the data that can be outliers without significantly affecting the model.

Two settings are considered in the following. First, hyperparameters are fixed to arbitrary values. Then, since it might be advised to tune the hyperparameters differently in the presence of outliers, the effect of outliers on their tuning by cross validation is studied.

Fixed hyperparameters

Here, the aim is to study the influence of outliers on the algorithms without tuning the hyperparameters accordingly. These are thus set to the following fixed values: $\varepsilon = 0.01$, $\sigma = 1$ and $C_{QP} = 10$, $C_{LP} = 20$, $C_{LS} = C_{RLS} = 50$, respectively for QP-SVR, LP-SVR, LS-SVM and RLS-SVM[7]. These values are arbitrarily chosen such that all SVM algorithms lead to the same test error when trained on the uncorrupted data set ($N_{out} = 0$). The curves in plain line on Fig. 3.3 show the resulting test error as a function of N_{out} for each algorithm[8]. As expected, the QP-SVR and LP-SVR algorithms using the ε-insensitive loss function are mostly insensitive to outliers below 30% of corruption. On the other hand, the LS-SVM, based on a quadratic criterion, suffers from the presence outliers even in low numbers. However, the Robust LS-SVM allows to increase the robustness of the LS-SVM. The test error of the RLS-SVM is comparable to the ones of the QP

[7] The *robustlssvm* function of the LS-SVMlab toolbox [249] is used for the Robust LS-LSVM.

[8] As $N = 100$, N_{out} also corresponds to the percentage of outliers.

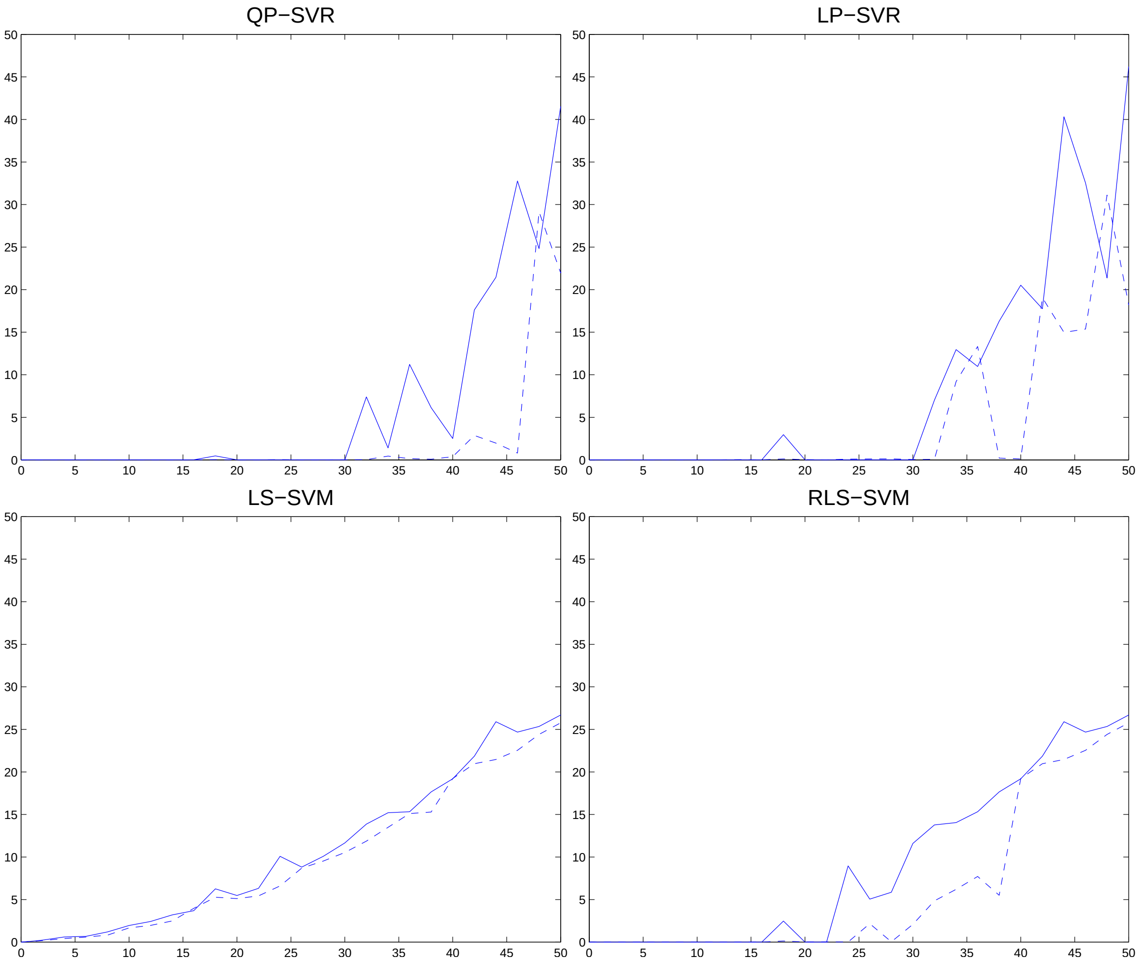


Figure 3.3: Test error versus the percentage of outliers for the models trained with the hyperparameter C fixed (—) or tuned by 5-fold cross validation (- -).

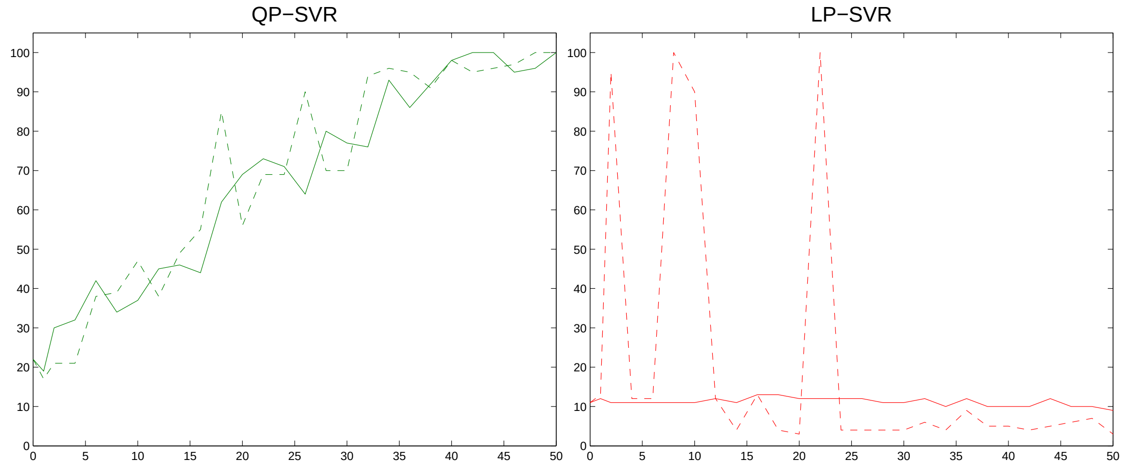


Figure 3.4: Number of SVs versus the percentage of outliers for the QP-SVR (*left*) and LP-SVR (*right*) algorithms with fixed (—) and tuned (- -) hyperparameters C.

or LP-SVR for $N_{out} \leq 22$. Nonetheless, for a higher percentage of corruption, the RLS-SVM leads to higher test errors than the QP and LP-SVR.

Regarding the number of SVs, plotted in plain line on Fig. 3.4, the LP-SVR is mostly insensitive to outliers and leads to a sparse model for all N_{out}. On the other hand, for the QP-SVR, since the SVs are the points outside of the insensitivity tube (i.e. points leading to large errors), the number of SVs increases with the number of outliers. The LS-SVM and RLS-SVM models retain the N training samples as SVs without selection.

Tuned hyperparameter C

Now, the study focuses on the effect of outliers on the tuning of the trade-off hyperparameter C. The tuning of this hyperparameter should take into account the confidence on the data, that degrades with the number of outliers. In the experiments, C is tuned by 5-fold cross validation (see section 1.1 on page 2), while ε is kept fixed to 0.01. The curves in dash line on Fig. 3.3 show the resulting test error as a function of N_{out} for each algorithm. In most cases, these curves remain below the test error obtained with a fixed hyperparameter C.

Table 3.4: Degradation points Δ_r at various levels r for the different algorithms with fixed and tuned hyperparameter C.

Degradation point (in %)	C	QP-SVR	LP-SVR	LS-SVM	RLS-SVM
$\Delta_{0.01}$	fixed	32	32	1	24
	tuned	32	24	1	18
$\Delta_{0.1}$	fixed	32	32	1	24
	tuned	40	34	2	35
Δ_5	fixed	42	32	16	28
	tuned	48	42	18	34
Δ_{10}	fixed	42	34	30	30
	tuned	48	42	30	40

Table 3.4 compares the degradation points in the two settings (C tuned or fixed). In this example, tuning C by 5-fold cross validation allows to increase the degradation points of the robust models (QP-SVR, LP-SVR and RLS-SVM) by approximately 10. However, this is not verified for $\Delta_{0.01}$.

Regarding the values of the hyperparameter C, plotted in Fig. 3.5, all the methods tend to select high values for less than 15% of corrupted data. The 5-fold cross validation procedure allows to decrease the value of C when the percentage of outliers is between 15 and 50.

Remark 8. *In the presence of outliers in the data, robust cross-validation estimates of the generalization error could be used. Such estimates are studied in [251].*

Automatic tuning of ε (ν-QPSVR and μ-LPSVR)

The aim here is to illustrate the relationship between the hyperparameter ν (or μ) of the automatic tuning algorithms of Sect. 3.1.3 and the robustness to outliers. The ν-QPSVR (3.24) and μ-LPSVR (3.26) problems are solved for $C_{QP} = 10$, $C_{LP} = 20$ and different values of the hyperparameters ν and μ in the range $]0, 1]$. For each value, the degradation point $\Delta_{0.01}$ is evaluated. The results, plotted in Fig. 3.6, show a linear dependency between $\Delta_{0.01}$ and the hyperparameter, either ν or μ. The plots also suggest a relationship between ν and μ of the type $\nu = 1 - \mu$, similar to the one derived in [181] for the equivalence between μ-LPSVR and the ν-LPR method of [246].

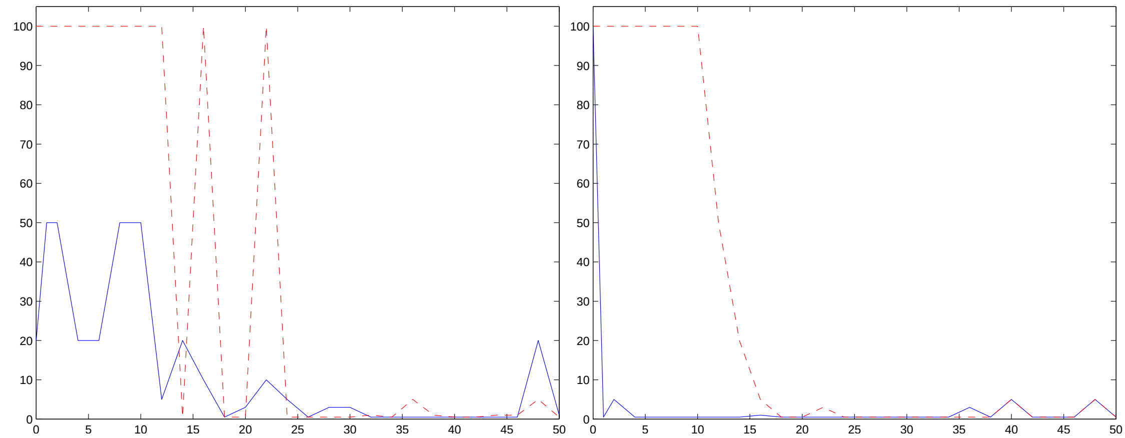


Figure 3.5: Value of C tuned by 5-fold cross validation versus the percentage of outliers. *Left:* for the QP-SVR (—) and the LP-SVR (- -) algorithms. *Right:* for the LS-SVM (—) and the RLS-SVM (- -) algorithms.

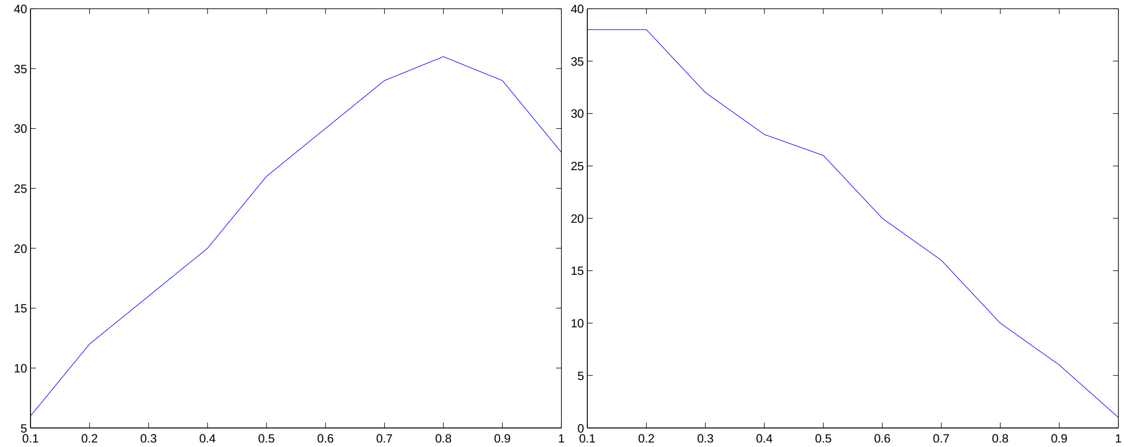


Figure 3.6: *Left:* degradation point $\Delta_{0.01}$ versus the hyperparameter ν for ν-QPSVR. *Right:* degradation point $\Delta_{0.01}$ versus the hyperparameter μ for μ-LPSVR.

3.4 Conclusion

Kernel methods and Support Vector Regression offer an attractive approach to the problem of nonlinear function approximation. Benefiting from nice generalization properties, these methods are also interesting for solving some difficult nonlinear system identification problems, i.e. when few data are available or when the data are corrupted by non-Gaussian noise. However, regarding system identification some open issues remain such as the development of algorithms for recurrent (or Output-Error) models.

Various formulations for the SVM problem are derived from the choice of the loss function (either ε-insensitive or squared loss). However, only the ε-insensitive function intrinsically provides sparsity. But the choice of a particular loss function (and thus of a particular algorithm) should also be made in accordance with prior knowledge on the noise if available. In particular, in the presence of outliers, a robust loss function such as the ε-insensitive loss or the Robust version of the LS-SVM algorithm are advised. Moreover, if sparsity is an issue, the LP-SVR, for which the number of SVs is mostly insensitive to outliers, might be preferred. When no information on the noise level is available, using the ε-insensitive loss function involves the tuning of the threshold ε. However, algorithms such as ν-QPSVR and μ-LPSVR allow to automatically tune ε by adding a new hyperparameter for which interpretations and asymptotically optimal values for specific noise distributions are available. But as shown by the experiments, when training a model from few data, invoking asymptotic optimality to choose a particular algorithm or hyperparameter value may be misleading.

3.5 Further reading

Algorithms. Recently, some authors studied the benefits of solving the QP-SVR problem in primal space [41]. This approach is also studied in the framework of LS-SVM [77]. The use of RBF kernels with varying width and an Orthogonal Least Square (OLS) forward selection procedure [46] has been considered in [296]. Online methods, taking points into account in a sequential manner, are studied in [185] and [175]. As for classification, the Relevant Vector Machine (or sparse Bayesian learning) [261] has been proposed to predict error bars instead of point evaluations. Another probabilistic framework for SVR with Gaussian kernels, allowing to predict error bars, can be found in [88].

SVM and automatic control. The first use of SVMs for system identification can be found in [71] where it is shown that SVMs have the general good features for system identification, though showing that it does not always yield a parsimonious model. Different papers followed [39, 163, 308, 5, 295] and different modifications were made to adapt SVR to system identification (see [47], [87] and references therein). Finally, adaptive inverse control based on SVR is proposed in [292].

The LS-SVM simplification of SVR allows for more flexibility and adaptability to other tasks. In [255], the LS-SVM method is reformulated for recurrent models. However, the problem is simplified by neglecting the regularization term. For Gaussian RBF kernels, this simplified problem is equivalent to minimizing the training error of a recurrent RBF network [251]. Optimal control by LS-SVM has been proposed in [256].

Error-in-variables. Robust formulations for SVR have been proposed in order to deal with uncertainties in the inputs (also known as the error-in-variables problem in system identification). By considering bounded noise on the inputs, [268] proposed a Second Order Cone Programming (SOCP) formulation of SVR, also tested with Gaussian noise in [267]. For classification, this problem has been studied in [23], leading to a non-convex formulation. Another SOCP formulation has been proposed to deal with arbitrary distributions of noise with bounded mean and covariance both for classification and regression [237]. These methods have strong links with robust optimization.

Chapter 4

Incorporating prior knowledge in Support Vector Regression

ABSTRACT. *This chapter explores the addition of constraints to the linear programming formulation of the support vector regression problem for the incorporation of prior knowledge. Equality and inequality constraints are studied with the corresponding types of prior knowledge that can be considered for the method. These include particular points with known values, prior knowledge on any derivative of the function either provided by a prior model or available only at some specific points and bounds on the function or any derivative in a given domain. Moreover, a new method for the simultaneous approximation of multiple outputs linked by some prior knowledge is proposed. This method also allows the consideration of different types of prior knowledge on particular outputs while training on multiple outputs. Synthetic examples show that incorporating a wide variety of prior knowledge becomes easy, as it leads to linear programs, and helps to improve the approximation in difficult cases. The benefits of the method are finally shown on a real-life application, the estimation of in-cylinder residual gas fraction in spark ignition engines, which is representative of numerous situations met in engineering.*

SUPPORT Vector Regression aims at learning an unknown function based only on a training set of N input-output pairs $(\boldsymbol{x}_i, y_i)$ in a black box modelling approach. Nonetheless, in real world applications, such as system identification, some information is usually known beforehand. This prior knowledge can take many forms from the positiveness of a physical variable to the shape of the function on a particular region. Incorporating this knowledge into the learning scheme can improve the quality of the model in different ways. The function can be approximated in regions of the input space where the data are sparse, specific properties of the function such as a maximum or a curvature at some point can be introduced in the model, and so on. However, incorporating prior knowledge into support vector learning is not trivial and is still a partially open issue. Developed for neural networks, the methods that use prior knowledge for the network initialization [7] or structure selection (KBANN) [266] cannot be directly transferred to SVMs, since these tasks are typically handled by support vector learning. The smoothness assumption is also another simple form of prior knowledge implicitly included in support vector machines (SVM) [245, 79]. On the other hand, a certain amount of work has been done in the past decade to build support vector machine classifiers or kernels, whose outputs are invariant under a known transformation of the input (see the review in chapter 2). However, such type of prior knowledge is rarely available in the regression framework.

The present chapter explores the addition of equality or inequality constraints of a rather general form, linear w.r.t. the parameters, to the LP-SVR problem (see Sect. 3.1.2) and exposes the different types of prior knowledge that can be included in the learning with this technique. Interesting issues are considered such as: knowledge in a region of the input space without data, knowledge on any derivative provided either by a prior model or only at particular points and prior knowledge between multiple outputs. In addition, the use of potential support vectors, located at the points of discretization of the prior knowledge, is discussed. When local kernels such as the RBF kernel are used, adding potential support vectors becomes necessary to satisfy the prior knowledge given in regions of input space that lack training samples.

The first part of the chapter presents the related work and discusses the differences and advantages of the proposed method. A note on the derivatives of the LP-SVR model is first made in section 4.2 before exposing the proposed method. In particular, section 4.3 is dedicated to the introduction of equality constraints to the optimization problem, while section 4.4 considers prior knowledge in the form of inequalities. Synthetic examples are given throughout the paper to illustrate the interest of the methods, while section 4.5 presents their application to practical problems. In particular, section 4.5.3, applies the method to a real-life problem: the estimation the in-cylinder residual gas fraction of a spark ignited engine.

In this chapter it is assumed that the reader is familiar with the linear programming SVR (LP-SVR) as described in section 3.1.2. This chapter is based on the works [150], [26] and [27].

4.1 Related work

Classification. As seen in chapter 2, a certain amount of work has been done in the past decade to build SVM classifiers or kernels, whose outputs are invariant under a known transformation of the input. Such invariances can be, for instance, incorporated by creating new samples to enlarge the training set. This method can be easily implemented in the context of pattern recognition. The virtual sample method can also directly be applied to regression, but, in this case, prior knowledge considering invariance of the output is not easily available (except for a few special cases such as symmetric or periodic functions). Nonetheless the idea of virtual samples can be modified to be used for regression when prior knowledge on the function is given in terms of output values or derivatives instead of invariance as in the proposed method of section 4.3.1.

Another simple and straightforward way to incorporate prior knowledge is to weight the samples. In practice, this amounts to weight the errors or to choose a different trade-off parameter C for different samples in the objective function. The application of weighted formulations such as [129] or [306] to regression is straightforward. In [257], the weighting of samples is applied to non-stationary time-series forecasting where it was noticed that the distant past data were less significant than the recent past data. To include this forgetting effect, the weights C_i applied on the errors are given by an increasing function of time.

Constrained regression. The topic of linearly constrained least squares regression has a long history and is involved in a variety of applications, both with equality constraints or inequality constraints. Incorporating linear equality constraints is useful whenever one or several coefficients must be expressed as a linear combination of the others. Inequality constraints may arise from requirements such as positivity, monotonicity, and convexity [155]. Many points presented here are close in spirit to these approaches, in the sense that prior knowledge is also incorporated by the addition of constraints.

Spline models. In function modeling by fitting separate models in different regions of the input space, the introduction of constraints to take into account continuity knowledge at the boundary points, denoted knots, is the basic idea of spline models [113]. Moreover, in smoothing splines, this idea is extended to avoid the selection of knots by using a maximal set of knots and controlling the resulting function smoothness by regularization. Thus some of the ideas presented here have strong links with spline models. In particular, section 4.3.4 focuses on building multi-models for continuous function approximation. However, it presents in a general way the nature of the submodels and extends the fitting criterion to the ℓ_1-norm. Besides, a certain amount of work has been done to incorporate more prior knowledge in the form of linear inequalities into spline models, see for instance [288] and [189].

System identification. As in this thesis, the incorporation of prior knowledge in the context of system identification is often referred to as grey box modeling [241, 271]. As exposed in [198], parallel or series structures can be used to combine a prior model with a black box model trained to estimate the residuals of the prior model. Combinations of neural networks and first-principles models have been studied for chemical applications in [213, 258], showing that the models require less training samples and better generalize thanks to the prior knowledge. Besides, parameter-constrained identification, related to constrained regression, has been considered in [97, 11, 63, 49, 260]. In addition, the inclusion of prior knowledge in fuzzy modeling has been studied in [168, 1] and the identification of polynomial models with known fixed points has been considered in a multi-objective framework in [199]. Finally, a general optimization framework including various forms of prior knowledge in RBF networks has been proposed in [131]. However, this framework does not handle non-quadratic loss functions nor select the basis functions as done in the SVM framework.

Semiparametric modeling. In [243], prior knowledge is incorporated in SVR by moving from nonparametric to semiparametric modeling. In this scheme, the bias term of the the kernel expansion is replaced by a parametric model whose parameters are determined by the optimization procedure. Semiparametric modeling can be included either in the quadratic or linear programming form of SVR.

Knowledge in a region of input space. In [86], the authors introduced prior knowledge on polyhedral regions in the context of linear programming SVM for linear classification. It was then extended to the nonlinear case in [85] via a reformulation of the kernel. In [182], it was finally adapted for regression and later on developed to apply nonlinear bounds on the model in any nonlinear region [184]. An advantage of this method is that the problem remains in a linear programming (LP) form. However, whenever the nonlinear regions are not given as explicit sets of a finite number of points, these regions have to be discretized before including the prior knowledge in the learning as a finite set of inequalities. Though also requiring the knowledge to be in the form of a discrete set of constraints, the method proposed here is more simple and involves less variables in the optimization program. The increase of complexity for Mangasarian's method is justified in the case of polyhedral regions, for which a finite set of constraints ensures that the prior knowledge is satisfied for all points in these regions. But when discretization occurs, as often required for nonlinear regions, this is no more true since adding constraints for the points of discretization only cannot guarantee that the nonlinear bound on the function holds in the whole nonlinear region. Thus, in this case, it is sufficient to consider simpler constraints as will be described in this paper. The method proposed here is also more general in the sense that it allows for the inclusion of knowledge on the derivatives of the model. Though Mangasarian's method could be extended to include prior knowledge on derivatives of the function, it has not been done so far.

Knowledge on the derivatives. In [158], prior knowledge on the derivatives has been incorporated in the QP formulation of the SVR problem by considering a training set containing at every point the target values, not only for the function but also for the derivative. The optimization problem is extended to include the minimization of the error on the derivative and thus simultaneously approximate the function and its derivative. As a result, the derivative of the kernel function is involved in the approximation function, which may slow down the estimations in the test phase. However, another formulation [157] has been proposed to circumvent this drawback. This general approach has also been formulated in the LS-SVM framework [125]. Whereas these methods consider knowledge on the derivatives at the training points only, the method proposed here allows to impose derivative values for any point. This is of particular interest when incorporating prior knowledge in regions not covered by the training data.

Support vector regression with multiple outputs. The issue of multiple outputs regression has been previously studied for SVR in [225]. In this approach, the multiple ε-insensitive loss functions are replaced by a ℓ_2-norm based loss function, resulting in a single scalar error value taking into account the errors on all the outputs. Though providing a mean for the regression of multiple outputs with dependencies, the method does not allow for the inclusion of prior knowledge on these dependencies. In [278], SVR is related to kriging. In this context, extending SVR to multiple outputs amounts to use an appropriate covariance function for the kernel. Thus the training algorithm is not modified. Moreover, information about relations between the outputs can be incorporated by designing the structure of the covariance accordingly. Another approach, proposed in [297], allows to incorporate prior knowledge between multiple outputs. In this approach, the loss function is considered as a distance in an output feature space and thus computed by a kernel function defined over the output space. Prior information on the outputs such as specific loss functions or invariances can be embedded in this output kernel. Being a very general framework, applying to regression as well as classification, this method also requires more complex steps. These include the decomposition of the outputs $\boldsymbol{\Phi}_{out}(\boldsymbol{y}_i)$ in the output feature space by Kernel Principal Component Analysis (KPCA), learning multiple mappings from the input to the resulting principal components and, finally, solving the pre-image problem for every new test sample, i.e. finding the pre-image $\boldsymbol{y}_t$ of the output $\boldsymbol{\Phi}_{out}(\boldsymbol{y}_t)$ estimated in feature space. In particular, the pre-image problem is still a partially open issue. The method proposed in section 4.3.4 allows for a simple formulation of the dependencies between multiple outputs and their inclusion as a finite set of constraints that leaves the nature of the learning problem unchanged. In [177], the method of [182] for knowledge-based kernel approximation is extended to provide advice to a reinforcement learner. In this setting, the value of each output determines if a specific action must be taken. Prior knowledge may indicate that under a set of conditions, one action is preferable to another. This amounts to consider the inequality: $f_1(\boldsymbol{x}) \geq f_2(\boldsymbol{x}) + \beta$, for $\boldsymbol{x}$ satisfying a set of conditions, where the output f_1 represents the preferred action to the one represented by f_2 and β accounts for how much preferable it is. The general framework proposed in section 4.4.2 includes this form of prior knowledge as a particular case.

Conclusion. All the related approaches, that allow the use of prior knowledge for SVR, focus on particular types of prior knowledge. The strength of the proposed method thus lies in its generality, but also in its simplicity as it amounts to add linear constraints to the problem. Incorporating prior knowledge by the addition of constraints has been extensively studied in other areas such as smoothing splines or least squares regression. The present chapter proposes to transfer these techniques to the LP-SVR framework and explores the consequences of this transfer, from the inclusion of basic forms of prior knowledge to the handling of constrained multi-output regression.

4.2 Linearity of the derivatives of a kernel expansion

Noticing that the kernel expansion

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\boldsymbol{\alpha} + b, \tag{4.1}$$

is linear w.r.t. the parameters $\boldsymbol{\alpha}$ allows to write the derivative of the model output with respect to the jth component x^j of $\boldsymbol{x} \in \mathbb{R}^p$ as

$$\frac{\partial f(\boldsymbol{x})}{\partial x^j} = \sum_{i=1}^{N} \alpha_i \frac{\partial k(\boldsymbol{x}_i, \boldsymbol{x})}{\partial x^j} = \boldsymbol{r}_1(\boldsymbol{x})^T \boldsymbol{\alpha}, \tag{4.2}$$

where $\boldsymbol{r}_1(\boldsymbol{x}) = [\partial_{x^j} k(\boldsymbol{x}_1, \boldsymbol{x}) \ \dots \ \partial_{x^j} k(\boldsymbol{x}_i, \boldsymbol{x}) \ \dots \ \partial_{x^j} k(\boldsymbol{x}_N, \boldsymbol{x})]^T$. The derivative (4.2) is also linear in $\boldsymbol{\alpha}$. In fact, the form of the kernel expansion implies that all the derivatives are linear in $\boldsymbol{\alpha}$ such as, for instance, the Laplacian

$$\nabla^2 f(\boldsymbol{x}) = \sum_{j=1}^{p} \frac{\partial^2 f(x)}{\partial {x^j}^2}. \tag{4.3}$$

This second-order derivative is a measure of the roughness of the function on which prior knowledge may be considered. The linearity in $\boldsymbol{\alpha}$ allows to write any scalar derivative $f^{(k)}$ of order k as

$$f^{(k)}(\boldsymbol{x}) = \boldsymbol{r}_k(\boldsymbol{x})^T \boldsymbol{\alpha}, \tag{4.4}$$

where $\boldsymbol{r}_k(\boldsymbol{x})$ contains the coefficients for the kth order derivative that only depend on the kernel and the training set. $\boldsymbol{r}_1(\boldsymbol{x})$ and $\boldsymbol{r}_2(\boldsymbol{x})$ are given for a Gaussian RBF kernel in Appendix A.

Setting up $\boldsymbol{r}(\boldsymbol{x})$ accordingly, $\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{\alpha}$ can be any component or linear combination of components of any derivative of f. This general remark allows the incorporation of constraints on any derivative of a kernel expansion f into a linear program such as (3.22).

4.3 Adding equality constraints

In this section, prior knowledge on the function is considered via n_{SC} sets of equality constraints of the type

$$g_k(\boldsymbol{z}) = h_k(\boldsymbol{z}), \ \forall \boldsymbol{z} \in Z_k, \tag{4.5}$$

where g_k can be any linear combination of f and its derivatives, h_k is an arbitrary function modeling the prior information and the set $Z_k = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_p, \dots, \boldsymbol{z}_{|Z_k|}\}$ contains the points of interest for the kth set of constraints. The points of Z_k can be chosen in the training set but are not restricted to these data. As will be seen in the following, the equalities (4.5) can be, for many interesting prior knowledge forms, reformulated as equality constraints that are linear w.r.t. the parameters $\boldsymbol{\theta} = [\boldsymbol{\alpha}^T \ b]^T$ and of the general form

$$\boldsymbol{\Gamma}_k(Z_k)\boldsymbol{\theta} = \boldsymbol{\beta}_k(Z_k). \tag{4.6}$$

The matrix $\boldsymbol{\Gamma}_k(Z_k)$ is built from rows $\boldsymbol{\gamma}_k(\boldsymbol{z})^T$ as

$$\boldsymbol{\Gamma}_k(Z_k) = \begin{bmatrix} \boldsymbol{\gamma}_k(\boldsymbol{z}_1)^T \\ \vdots \\ \boldsymbol{\gamma}_k(\boldsymbol{z}_p)^T \\ \vdots \\ \boldsymbol{\gamma}_k(\boldsymbol{z}_{|Z_k|})^T \end{bmatrix}, \tag{4.7}$$

where each row corresponds to a point of interest in Z_k. The constraints (4.6) can easily be introduced into (3.22) leading to the following linear programming SVR with equality constraints (LPSVR-EC)

$$\begin{aligned} &\min_{\boldsymbol{\alpha},b,\boldsymbol{\xi},\boldsymbol{a}\geq\boldsymbol{0}} \quad \boldsymbol{1}^T\boldsymbol{a} + C\boldsymbol{1}^T\boldsymbol{\xi} \\ &\text{s.t.} \quad \begin{array}{rcl} -\boldsymbol{\xi} \leq & \boldsymbol{K}\boldsymbol{\alpha} + b\boldsymbol{1} - \boldsymbol{y} & \leq \boldsymbol{\xi} \\ 0 \leq & \boldsymbol{1}\varepsilon & \leq \boldsymbol{\xi} \\ -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a} \\ & \boldsymbol{\Gamma}_k(Z_k)\boldsymbol{\theta} & = \boldsymbol{\beta}_k(Z_k), \ k = 1, \dots, n_{SC}. \end{array} \end{aligned} \tag{4.8}$$

The optimization problem to solve involves $3N + 1$ variables, $\sum_{k=1}^{n_{SC}} |Z_k|$ equality constraints and $5N$ inequality constraints.

Though adding equality constraints does not change the linear programming nature of the optimization problem, this can lead to an infeasible problem. To deal with the case where all the constraints cannot be satisfied simultaneously, the equalities can also be enforced by soft constraints. Introducing a set of n_{SC} vectors $\boldsymbol{u}_k = [u_1^k \ \dots \ u_p^k \ \dots \ u_{|Z_k|}^k]^T$ of positive slack variables and a set of trade-off parameters λ_k leads to the linear programming SVR with soft equality constraints (LPSVR-SEC)

$$\begin{aligned} &\min_{\boldsymbol{\alpha},b,\boldsymbol{\xi},\boldsymbol{a}\geq\boldsymbol{0},\boldsymbol{u}_k\geq\boldsymbol{0}} \quad \boldsymbol{1}^T\boldsymbol{a} + C\boldsymbol{1}^T\boldsymbol{\xi} + \sum_{k=1}^{n_{SC}} \lambda_k \boldsymbol{1}^T\boldsymbol{u}_k \\ &\text{s.t.} \quad \begin{array}{rcl} -\boldsymbol{\xi} \leq & \boldsymbol{K}\boldsymbol{\alpha} + b\boldsymbol{1} - \boldsymbol{y} & \leq \boldsymbol{\xi} \\ 0 \leq & \boldsymbol{1}\varepsilon & \leq \boldsymbol{\xi} \\ -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a} \\ -\boldsymbol{u}_k \leq & \boldsymbol{\Gamma}_k(Z_k)\boldsymbol{\theta} - \boldsymbol{\beta}_k(Z_k) & \leq \boldsymbol{u}_k, \ k = 1, \dots, n_{SC}, \end{array} \end{aligned} \tag{4.9}$$

that involves $3N + 1 + \sum_{k=1}^{n_{SC}} |Z_k|$ variables and $5N + 2\sum_{k=1}^{n_{SC}} |Z_k|$ inequality constraints. This problem formulation also offers the possibility to weight the effect of the prior knowledge on the solution. Large values of λ_k increase the fit to the prior knowledge, whereas small values imply a fit closer to the data.

The problem (4.9) includes the minimization of the ℓ_1-norm of the errors for the soft equality constraints. Using the ε-insensitive loss function on these errors (with a different ε than the one used for the training set) is also possible and straightforward. It can be used to include almost exact or biased knowledge in an interval by authorizing violations of the equality constraints that are less than a threshold ε_k. This leads to

$$\begin{array}{llcl}
 & \min\limits_{\boldsymbol{\alpha},b,\boldsymbol{\xi},\boldsymbol{a}\geq\mathbf{0},\boldsymbol{u}_k} & \mathbf{1}^T\boldsymbol{a} + C\mathbf{1}^T\boldsymbol{\xi} + \sum\limits_{k=1}^{n_{SC}} \lambda_k \mathbf{1}^T\boldsymbol{u}_k & \\
\text{s.t.} & -\boldsymbol{\xi} \leq & \boldsymbol{K\alpha} + b\mathbf{1} - \boldsymbol{y} & \leq \boldsymbol{\xi} \\
 & \mathbf{0} \leq & \mathbf{1}\varepsilon & \leq \boldsymbol{\xi} \\
 & -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a} \\
 & -\boldsymbol{u}_k \leq & \boldsymbol{\Gamma}_k(Z_k)\boldsymbol{\theta} - \boldsymbol{\beta}_k(Z_k) & \leq \boldsymbol{u}_k,\ k=1,\dots,n_{SC} \\
 & \mathbf{0} \leq & \mathbf{1}\varepsilon_k & \leq \boldsymbol{u}_k,\ k=1,\dots,n_{SC}.
\end{array} \tag{4.10}$$

This type of prior knowledge can be equivalently considered with inequality constraints as studied in section 4.4.

Remark 9 (Balanced formulations)**.** *In problems (4.9) and (4.10),* balanced *formulations of the objective functions can be obtained by adding normalizing factors. For instance, in problem (4.10), the sums* $\mathbf{1}^T\boldsymbol{a}$*,* $\mathbf{1}^T\boldsymbol{\xi}$ *and* $\mathbf{1}^T\boldsymbol{u}_k$*,* $k = 1,\dots,n_{SC}$*, involve respectively* N*,* N*, and* $|Z_k|$*,* $k = 1,\dots,n_{SC}$*, terms, leading to the balanced objective function* $1/N\mathbf{1}^T\boldsymbol{a} + C/N\mathbf{1}^T\boldsymbol{\xi} + \sum_{k=1}^{n_{SC}} \lambda_k/|Z_k|\mathbf{1}^T\boldsymbol{u}_k$*. When the sums involve different numbers of terms, as is the case here, this normalization simplifies the tuning of the hyperparameters by maintaining the same order of magnitude between the regularization, error and prior knowledge terms in the objective function. This allows to ease the choice of their value based on the application goals and confidence on the prior knowledge. Hence, the hyperparameters become problem-size independent. These balanced formulations will be used in Sect. 4.5.3 for the real-life application.*

Remark 10 (Potential support vectors)**.** *The incorporation of prior knowledge can help to enhance the models in regions of input space that lack training data. However, when using local kernels such as the RBF kernel, the models may fail due to a lack of support vectors. In this case, the samples in* Z_k*,* $k = 1,\dots,n_{SC}$*, can be added to the set of potential support vectors, leading to extended models* $f(\boldsymbol{x}) = \boldsymbol{K}(\boldsymbol{x}, [\boldsymbol{X}^T\ \boldsymbol{Z}_1^T\ \dots\ \boldsymbol{Z}_{n_{SC}}^T])\boldsymbol{\alpha} + b$*, where* $\boldsymbol{\alpha}$ *now contains* $N + \sum_{k=1}^{n_{SC}} |Z_k|$ *parameters. Including the potential SVs in the algorithms above amounts to replace the kernel matrix* $\boldsymbol{K}$ *by* $\boldsymbol{K}(\boldsymbol{X}^T, [\boldsymbol{X}^T\ \boldsymbol{Z}_1^T\ \dots\ \boldsymbol{Z}_{n_{SC}}^T])$ *and compute* $\boldsymbol{\Gamma}_k(Z_k)$ *accordingly. Note that the choice of adding the samples in* Z_k*,* $k = 1,\dots,n_{SC}$*, as potential SVs can be different for each* k*.*

The following presents typical applications of constrained optimization for the introduction of prior knowledge with the corresponding settings of $\boldsymbol{\Gamma}_k(Z_k)$ and $\boldsymbol{\beta}_k(Z_k)$ for (4.6). Illustrative examples are provided throughout the presentation. Since the purpose of these examples is mainly to show the application of the method on simple and easy to visualize problems, the hyperparameters are chosen arbitrarily without fine tuning. The mean squared error (MSE) is computed w.r.t. the true function being approximated and on the same set of points in the x variable than the one used for training. Numerical experiments on more concrete examples are given in section 4.5.

4.3.1 Prior knowledge on particular points

Prior knowledge on the function to approximate can take the form of $|Z_0|$ particular points $(\boldsymbol{z}_p, y_p) \in Z_0$ for which the values are certain (intercept, maximal values, equilibrium points…). For these points, we would like the model to give an exact value and not just an approximation based on noisy data or a dataset that lacks samples around these points. One way to tackle this problem is to apply, for these points, hard constraints of the type

$$f(\boldsymbol{z}_p) = \boldsymbol{K}(\boldsymbol{z}_p, \boldsymbol{X}^T)\boldsymbol{\alpha} + b = y_p, \tag{4.11}$$

while soft constraints are applied to the training set. Defining the matrix $\boldsymbol{Z}_0 = [\boldsymbol{z}_1 \ \dots \ \boldsymbol{z}_p \ \dots \ \boldsymbol{z}_{|Z_0|}]^T$ and the vector $\boldsymbol{y}^{(0)} = [y_1 \ \dots \ y_p \ \dots \ y_{|Z_0|}]^T$, the following setting

$$\boldsymbol{\Gamma}_0(Z_0) = \begin{bmatrix}\boldsymbol{K}(\boldsymbol{Z}_0^T, \boldsymbol{X}^T) & \boldsymbol{1}\end{bmatrix}, \quad \boldsymbol{\beta}_0(Z_0) = \boldsymbol{y}^{(0)}, \tag{4.12}$$

is used for the problems (4.8) or (4.9) to incorporate the prior knowledge.

The points $(\boldsymbol{z}_p, y_p)$ can also be considered as virtual samples with high (or infinite) confidence and thus can be added to the training set as potential SVs. In this case, the linear program with hard constraints on the virtual samples can be formulated as

$$\begin{array}{llcl}
\displaystyle\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi}, \boldsymbol{a} \geq \boldsymbol{0}} & & \boldsymbol{1}^T\boldsymbol{a} + C\boldsymbol{1}^T\boldsymbol{\xi} & \\
\text{s.t.} & -\boldsymbol{\xi} \leq & \boldsymbol{K}(\boldsymbol{X}^T, [\boldsymbol{X}^T \ \boldsymbol{Z}_0^T])\boldsymbol{\alpha} + b\boldsymbol{1} - \boldsymbol{y} & \leq \boldsymbol{\xi} \\
 & 0 \leq & \boldsymbol{1}\varepsilon & \leq \boldsymbol{\xi} \\
 & -\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a} \\
 & & \boldsymbol{K}(\boldsymbol{Z}_0^T, [\boldsymbol{X}^T \ \boldsymbol{Z}_0^T])\boldsymbol{\alpha} + b\boldsymbol{1} & = \boldsymbol{y}^{(0)},
\end{array} \tag{4.13}$$

where the set of vectors $[\boldsymbol{X}^T \ \boldsymbol{Z}_0^T]^T$ supports the model given by

$$f(\boldsymbol{x}) = \boldsymbol{K}(\boldsymbol{x}, [\boldsymbol{X}^T \ \boldsymbol{Z}_0^T])\boldsymbol{\alpha} + b, \tag{4.14}$$

where the vector $\boldsymbol{\alpha}$ contains $N + |Z_0|$ parameters. Note that a certain number of these parameters will be zero due to the sparsity of LP-SVR.

Example: adding information about one point. In this example, 61 training points are generated in the interval $-3 \leq x \leq 3$ for the approximation of sinc(x). A Gaussian noise of standard deviation 0.2 and mean 0 is added to the training data. To show the efficiency of the method, one virtual sample of coordinates $(0, 1)$ is added to the training set. Since this point is known for certain, it is assigned to a hard constraint implemented by equality as in (4.13). Figure 4.1 shows the improvement over a simple LP-SVR trained on the training set with the virtual sample considered as a standard sample.

4.3.2 Prior knowledge on the derivatives

Typically, knowledge on the derivatives may include the general shape of the function, local maxima or minima, saddle-points, high peaks, and so on. This type of knowledge is often available but, to our knowledge, there is no general method for its incorporation in SVR.

Consider that prior knowledge on the kth order derivative $f^{(k)}$ of the function f is available as

$$f^{(k)}(\boldsymbol{z}_p) = y_p^{(k)}, \ \forall \boldsymbol{z}_p \in Z_k. \tag{4.15}$$

This prior knowledge can be enforced in the training by using (4.4) and setting

$$\boldsymbol{\Gamma}_k(Z_k) = \begin{bmatrix} \boldsymbol{r}_k(\boldsymbol{z}_1)^T & 0 \\ \vdots & \\ \boldsymbol{r}_k(\boldsymbol{z}_p)^T & 0 \\ \vdots & \\ \boldsymbol{r}_k(\boldsymbol{z}_{|Z_k|})^T & 0 \end{bmatrix}, \quad \boldsymbol{\beta}_k(Z_k) = \begin{bmatrix} y_1^{(k)} \\ \vdots \\ y_p^{(k)} \\ \vdots \\ y_{|Z_k|}^{(k)} \end{bmatrix}, \tag{4.16}$$

in one of the problems (4.8) or (4.9). This method allows to incorporate prior knowledge on any derivative of order k for any set of points possibly different for each k, while keeping the model in the form of (4.1). In comparison, the methods described in [158] and [157] includes information on the derivatives at the points of the training set only.

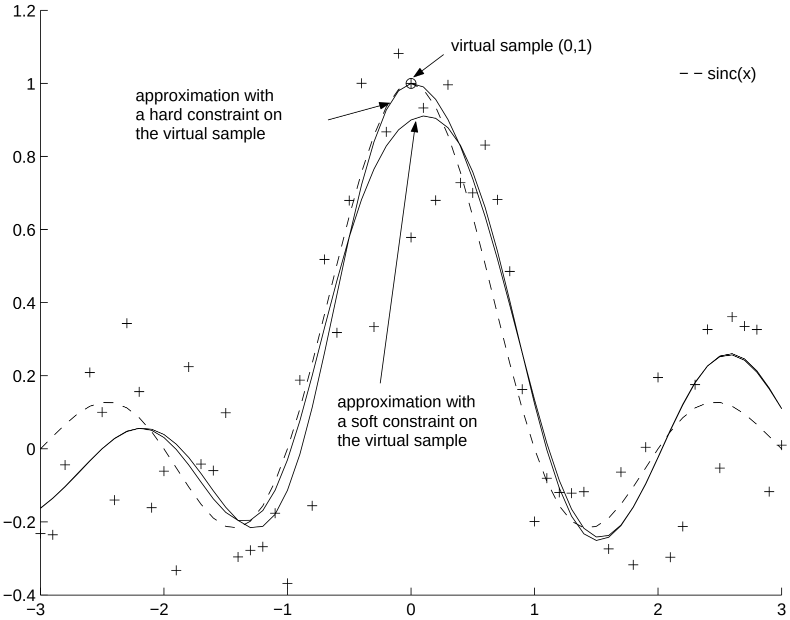


Figure 4.1: Approximation of the sinc function with an added virtual sample $(0, 1)$. The values for the different parameters are: $C = 10$, $\sigma = 0.5$, $\varepsilon = 0.1$. Applying a hard constraint on the virtual sample (known for certain) improves the accuracy around this point.

Example: knowledge on the derivatives at particular points. Here, we show the improvement in the quality of the approximation when only partial information about the shape is known at particular points via a mixture of derivatives (first and second order). Consider the problem of approximating the sinc function from noise-free data but without data around zero, that is for x in the interval $-1 < x < +1$. In this setting, the central peak of the sinc function cannot be approximated without prior knowledge. To show the effectiveness of the method, we consider only sparse and grossly approximate prior knowledge: $y^{(1)}(0) = 0$, $y^{(1)}(-0.5) = 1$, and $y^{(2)}(0) = -2$. This corresponds to a function with a maximum at $x = 0$, increasing around $x = -0.5$ and with a peak around $x = 0$. Figure 4.2 shows that incorporating this simple prior knowledge in (4.9) by (4.16) allows to recover the shape of the sinc function. Moreover, it must be noticed that the function is approximated without any training samples around 0 and thus without support vectors around 0. However, the number of SVs increases from 8 to 12.

4.3.3 Prior knowledge from a prior model

In some applications, one may wish to retain certain properties of a previous model such as the shape or the roughness at some specific points $\boldsymbol{z}_p$ in Z_k. This problem corresponds to learn a new model while constraining certain of its derivatives to equal those of the prior model at these particular points.

Assume now that the the prior model f^{prior} is a kernel expansion on N^{pr} samples, then $f^{prior}(\boldsymbol{x}) = \sum_{i=1}^{N^{pr}} \alpha_i^{prior} k(\boldsymbol{x}, \boldsymbol{x}_i) + b^{prior}$. Using the remark of section 4.2, its derivatives can also be expressed in a linear form (4.4) w.r.t. the parameters α_i^{prior} as $f^{prior\,(k)}(\boldsymbol{z}_p) = \boldsymbol{r}_k^{pr}(\boldsymbol{z}_p)^T \boldsymbol{\alpha}^{prior}$. Thus, in order to retain the kth order derivative from a prior model in kernel expansion form, the

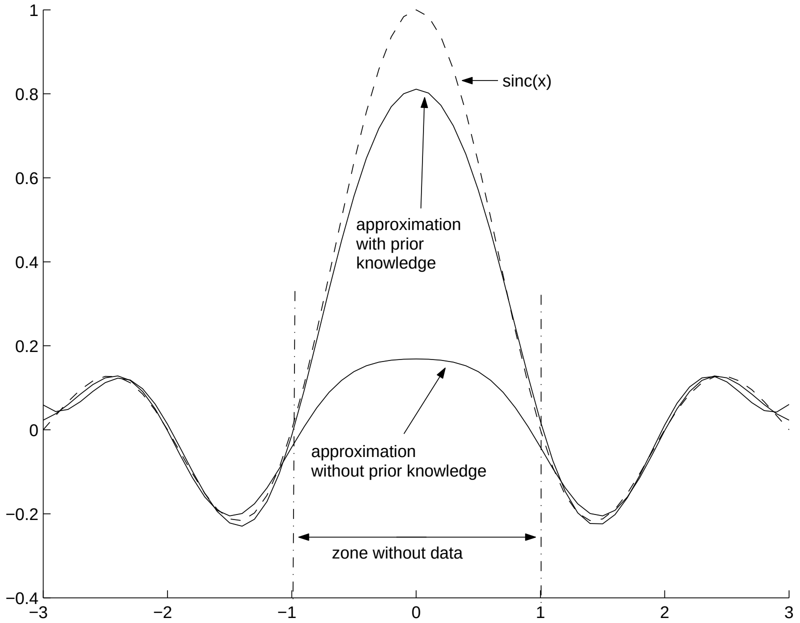


Figure 4.2: Approximation of the sinc function without data for x in the interval $-1 < x < +1$ and with prior knowledge on the derivatives at 2 particular points ($y^{(1)}(0) = 0$, $y^{(1)}(-0.5) = 1$, $y^{(2)}(0) = -2$). The values for the different parameters are: $\lambda = 50$, $C = 10$, $\sigma = 0.5$, $\varepsilon = 0.001$. A small amount of information on the derivatives helps to recover the shape of the overall function.

new model is trained by solving (4.8) or (4.9) with (4.6) set as

$$\boldsymbol{\Gamma}_k(Z_k) = \begin{bmatrix} \boldsymbol{r}_k(\boldsymbol{z}_1)^T & 0 \\ \vdots & \\ \boldsymbol{r}_k(\boldsymbol{z}_p)^T & 0 \\ \vdots & \\ \boldsymbol{r}_k(\boldsymbol{z}_{|Z_k|})^T & 0 \end{bmatrix}, \quad \boldsymbol{\beta}_k(Z_k) = \begin{bmatrix} \boldsymbol{r}_k^{pr}(\boldsymbol{z}_1)^T \boldsymbol{\alpha}^{prior} \\ \vdots \\ \boldsymbol{r}_k^{pr}(\boldsymbol{z}_p)^T \boldsymbol{\alpha}^{prior} \\ \vdots \\ \boldsymbol{r}_k^{pr}(\boldsymbol{z}_{|Z_k|})^T \boldsymbol{\alpha}^{prior} \end{bmatrix}. \tag{4.17}$$

Notice that $\boldsymbol{r}_k(\boldsymbol{z}_p)$ depends only on the kernel and the training set, with one component for each training sample. Thus, if the training set $\boldsymbol{X}$ of the new model includes the N^{pr} points on which the prior model has been trained and if the same kernel is used for both models, then N^{pr} components of $\boldsymbol{r}_k(\boldsymbol{z}_p)$ are equal to the components of $\boldsymbol{r}_k^{pr}(\boldsymbol{z}_p)$. The procedure can thus be speeded up by computing these components only once.

Example: knowledge on the derivative from a prior model helps recovering empty regions. This example shows the enhancement of the approximation when the general shape of the function is known via a prior model and incorporated in the learning by the previously described method. The shape is enforced by maximizing the similarity of the first order derivatives of the approximation to the derivatives of the prior model.

Consider the problem of approximating the sinc function based on a set of training points X without noise and a prior model approximating $s(x) = \text{sinc}(x) + \delta$. Only the shape of the function $s(x)$ is retained as prior knowledge, the added constant δ is considered unknown. The prior model is a standard LP-SVR, trained on 31 points of $s(x)$ (with $\delta = 1$) in the interval $-3 \leq x \leq 3$, yielding the parameters $\boldsymbol{\alpha}^{prior}$. On the other hand, the approximation f is trained on 34 points on the intervals $-3 \leq x \leq -1.4$ and $1.4 \leq x \leq 3$. Figure 4.3 shows the output of the prior model, the true sinc function and two approximations: one that includes the prior shape in the learning (4.9) by (4.17) and the other that does not (3.22). It is clear that the incorporation of prior knowledge on the shape of the function improves considerably the accuracy of the approximation with a small increase of the number of SVs from 8 to 12.

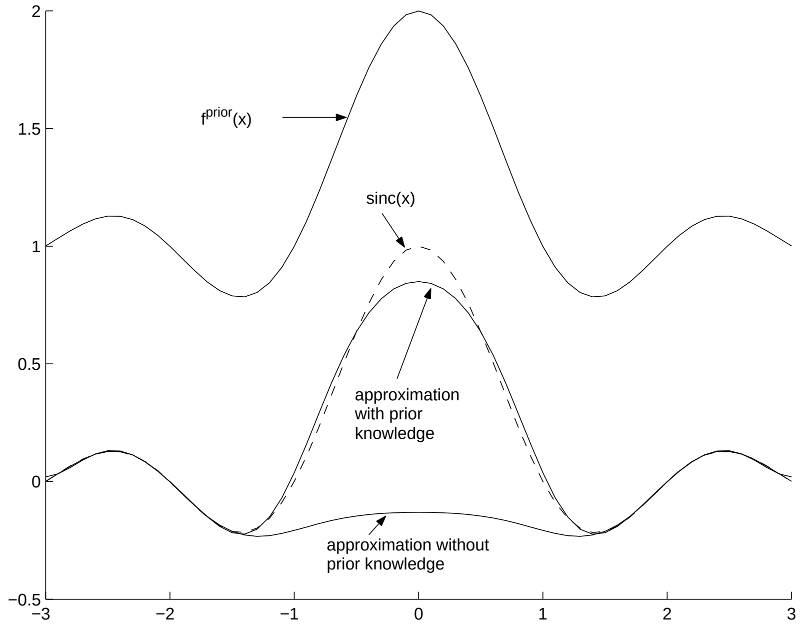


Figure 4.3: Approximation of the sinc function with prior knowledge on the shape from a prior model f^{prior}. The values for the different parameters are: $\lambda = 10$, $C = 13$, $\sigma = 0.5$, $\varepsilon = 0.001$.

Example: knowledge on the derivative from a prior model helps against the noise. The next example shows how prior knowledge on the shape can reduce the effect of noise. Consider the problem of approximating the sinc function from noisy data. A set of 31 training samples is now generated with additive Gaussian noise of mean 0 and standard deviation σ_{noise}, for x_i in the interval $-3 \leq x_i \leq 3$. Figure 4.4 shows the results for a large noise level $\sigma_{noise} = 1$ (considering the amplitude of $\text{sinc}(x) \approx 1.2$). It is clear that a standard LP-SVR applied to these data cannot approximate correctly the function. But taking the prior shape into account in the learning (4.9) by (4.17) filters the noise. Though the prior shape is only given by an approximative prior model provided by a LP-SVR training (3.22) on 31 points of the prior shape $s(x) = \text{sinc}(x) + \delta$, with $\delta = 1$, this yields a respectable approximation of the sinc function. The number of SVs increases slightly from 7 to 11. These results were obtained with a "handpicked" trade-off parameter $\lambda = 5$ without any tuning. This trade-off parameter appearing in the problems (4.8) and (4.9) is used to weight the effect of the prior knowledge on the solution. Based on prior knowledge of the noise level, and so, of the relative relevance of the data, the approximation can still be enhanced by increasing λ. A large λ increases the fit to the prior shape as opposed to the fit to the data implied by a small λ.

This problem is close to the setting of [158] and [157] as prior knowledge on the derivatives is given at every training point. However, here, the derivatives values are not directly available and are estimated by a prior model trained on another set of samples.

4.3.4 Prior knowledge between outputs

Simultaneous approximation of multiple functions is easily performed by neural networks by simply considering a multidimensional output layer. On the other hand, SVMs are single-output machines. The common approach for multi-outputs problems is thus to train as many independent SVMs as needed, one for each function to approximate. However, in some cases, these functions may be interdependent and the inclusion of these dependencies in the learning cannot be directly handled.

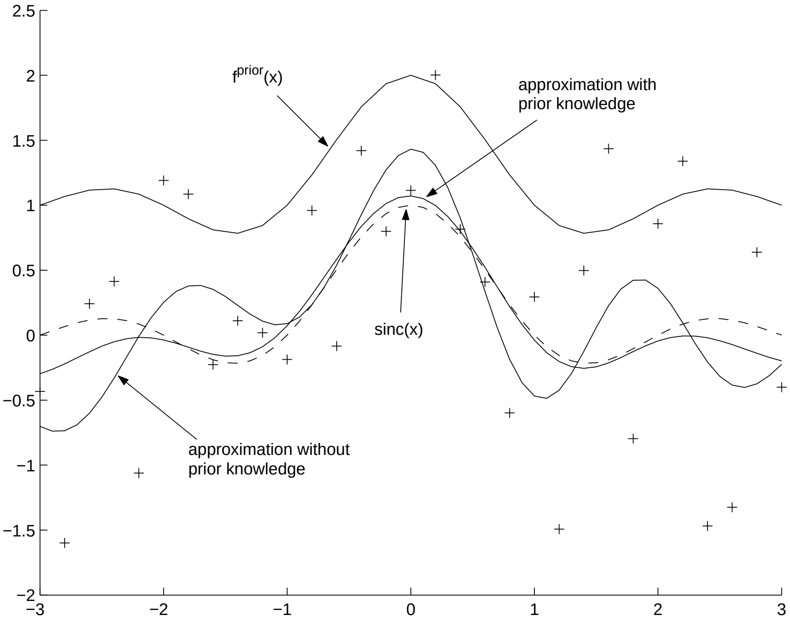


Figure 4.4: Approximation of the sinc function with noisy data and prior knowledge on the shape from a prior model f^{prior}. The values for the different parameters are: $\lambda = 5$, $C = 10$, $\sigma = 0.5$, $\varepsilon = 1$, $\sigma_{noise} = 1$. The prior shape helps to filter the noise.

This section presents a multi-outputs SVR approach to approximate multiple functions of the same inputs when some prior knowledge on the dependencies between these functions is available. After the description of the method, an example will show how to consider the results of section 4.3.2 in this framework to simultaneously approximate a function and its derivative. Then another example will consider the approximation of a continuous piecewise function by multiple models.

The multi-outputs SVR implements m functions f_j of the same inputs that are learned from the training set $(\boldsymbol{X}, \boldsymbol{y}_1, \dots, \boldsymbol{y}_j, \dots, \boldsymbol{y}_m)$. The approximated output vector $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}) \ \dots \ f_j(\boldsymbol{x}) \ \dots \ f_m(\boldsymbol{x})]^T$ is given by

$$\boldsymbol{f}(\boldsymbol{x})^T = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\boldsymbol{A} + \boldsymbol{b}^T, \tag{4.18}$$

where the matrix $\boldsymbol{A} = [\boldsymbol{\alpha}_1 \ \dots \ \boldsymbol{\alpha}_j \ \dots \ \boldsymbol{\alpha}_m]$ and the vector $\boldsymbol{b} = [b_1 \ \dots \ b_j \ \dots \ b_m]^T$ collect the parameters.

It is assumed that some prior knowledge on a linear combination of the m outputs with coefficients μ_j is available in the form

$$\sum_{j=1}^{m} \mu_j f_j(\boldsymbol{z}) = h(\boldsymbol{z}), \ \forall \boldsymbol{z} \in Z, \tag{4.19}$$

for an arbitrary function h. Defining the parameter vector $\tilde{\boldsymbol{\theta}} = [\boldsymbol{\alpha}_1^T \ b_1 \ \dots \ \boldsymbol{\alpha}_m^T \ b_m]^T$ and setting

$$\boldsymbol{\beta}(Z) = \begin{bmatrix} h(\boldsymbol{z}_1) \\ \vdots \\ h(\boldsymbol{z}_p) \\ \vdots \\ h(\boldsymbol{z}_{|Z|}) \end{bmatrix}, \tag{4.20}$$

the matrix $\boldsymbol{\Gamma}(Z)$ is built as in (4.7) from the rows $\boldsymbol{\gamma}(\boldsymbol{z}_p)^T$ in order to rewrite the prior knowledge (4.19) in the general form $\boldsymbol{\Gamma}(Z)\tilde{\boldsymbol{\theta}} = \boldsymbol{\beta}(Z)$. For a single point $\boldsymbol{z}_p$ of the set Z, this leads to

$$\boldsymbol{\gamma}(\boldsymbol{z}_p)^T\tilde{\boldsymbol{\theta}} = \mu_1 f_1(\boldsymbol{z}_p) + \dots + \mu_m f_m(\boldsymbol{z}_p). \tag{4.21}$$

Writing the jth output f_j as

$$f_j(\boldsymbol{z}_p) = \begin{bmatrix}\boldsymbol{K}(\boldsymbol{z}_p, \boldsymbol{X}^T) & 1\end{bmatrix} \begin{bmatrix}\boldsymbol{\alpha}_j \\ b_j\end{bmatrix}, \tag{4.22}$$

allows to rewrite (4.21) as

$$\boldsymbol{\gamma}(\boldsymbol{z}_p)^T \tilde{\boldsymbol{\theta}} = \begin{bmatrix}\boldsymbol{K}(\boldsymbol{z}_p, \boldsymbol{X}^T) & 1\end{bmatrix} \left(\mu_1 \begin{bmatrix}\boldsymbol{\alpha}_1 \\ b_1\end{bmatrix} + \dots + \mu_m \begin{bmatrix}\boldsymbol{\alpha}_m \\ b_m\end{bmatrix}\right), \tag{4.23}$$

in which the parameter vector $\tilde{\boldsymbol{\theta}}$ can be introduced by

$$\boldsymbol{\gamma}(\boldsymbol{z}_p)^T \tilde{\boldsymbol{\theta}} = \begin{bmatrix}\boldsymbol{K}(\boldsymbol{z}_p, \boldsymbol{X}^T) & 1\end{bmatrix} \begin{bmatrix}\mu_1 \boldsymbol{I} & \dots & \mu_m \boldsymbol{I}\end{bmatrix} \tilde{\boldsymbol{\theta}}, \tag{4.24}$$

where $\boldsymbol{I}$ stands for the identity matrix of size $N+1$. This formulation gives $\boldsymbol{\gamma}(\boldsymbol{z}_p)^T$ and thus yields, for the whole set Z, the matrix

$$\boldsymbol{\Gamma}(Z) = \begin{bmatrix}\boldsymbol{K}(\boldsymbol{Z}^T, \boldsymbol{X}^T) & \boldsymbol{1}\end{bmatrix} \begin{bmatrix}\mu_1 \boldsymbol{I} & \dots & \mu_m \boldsymbol{I}\end{bmatrix}. \tag{4.25}$$

Training a multi-outputs SVR can thus be performed by solving a problem similar to (4.9) but with multiple outputs:

$$\begin{aligned}
&\min_{\tilde{\boldsymbol{\theta}}, \boldsymbol{\xi}_j, \tilde{\boldsymbol{a}} \geq \boldsymbol{0}, \boldsymbol{u} \geq \boldsymbol{0}} \quad \boldsymbol{1}^T \tilde{\boldsymbol{a}} + \sum_{j=1}^{m} C_j \boldsymbol{1}^T \boldsymbol{\xi}_j + \lambda \boldsymbol{1}^T \boldsymbol{u} \\
&\text{s.t.} \quad \begin{array}{rcl}
-\boldsymbol{\xi}_j \leq & \boldsymbol{K}\boldsymbol{\alpha}_j + b_j \boldsymbol{1} - \boldsymbol{y}_j & \leq \boldsymbol{\xi}_j, \ j = 1, \dots, m \\
0 \leq & \boldsymbol{1}\varepsilon_j & \leq \boldsymbol{\xi}_j, \ j = 1, \dots, m \\
-\tilde{\boldsymbol{a}} \leq & \tilde{\boldsymbol{\alpha}} & \leq \tilde{\boldsymbol{a}} \\
-\boldsymbol{u} \leq & \boldsymbol{\Gamma}(Z)\tilde{\boldsymbol{\theta}} - \boldsymbol{\beta}(Z) & \leq \boldsymbol{u},
\end{array}
\end{aligned} \tag{4.26}$$

where $\tilde{\boldsymbol{a}} = [\boldsymbol{a}_1^T \ \dots \ \boldsymbol{a}_m^T]^T$, $\tilde{\boldsymbol{\alpha}} = [\boldsymbol{\alpha}_1^T \ \dots \ \boldsymbol{\alpha}_m^T]^T$, $\tilde{\boldsymbol{\theta}} = [\boldsymbol{\alpha}_1^T \ b_1 \ \dots \ \boldsymbol{\alpha}_m^T \ b_m]^T$, $\boldsymbol{\Gamma}(Z)$ and $\boldsymbol{\beta}(Z)$ are set respectively as in (4.25) and (4.20), and where the subscript j indicates the jth output. Different hyperparameters C_j and ε_j are assigned to each output f_j. This problem involves $m(3N+1)+2|Z|$ variables and $5Nm + |Z|$ constraints.

The prior knowledge between multiple outputs is easily incorporated in a linear program by adding equality constraints (then relaxed by slack variables) in a general form similar to the one used for the other types of prior knowledge. Thus all these types of prior knowledge can be mixed together, as shown in the following examples.

Approximating a function and its derivative by multi-outputs SVR with prior knowledge

Consider the problem[1] of approximating two functions knowing that one is the derivative of the other w.r.t. a particular component x^l. Then, using the results of both sections 4.3.2 and 4.3.4, this prior knowledge can be used to enhance the training. In this case, the prior knowledge for a point $\boldsymbol{x}$ is written

$$\frac{\partial f_1(\boldsymbol{x})}{\partial x^l} - f_2(\boldsymbol{x}) = \boldsymbol{r}_1(\boldsymbol{x})^T \boldsymbol{\alpha}_1 - \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\boldsymbol{\alpha}_2 - b_2 = 0. \tag{4.27}$$

The points on which this prior knowledge is considered are the points of the training set ($Z = X$). The corresponding setting is thus

$$\boldsymbol{\Gamma}(Z) = \begin{bmatrix}
\boldsymbol{r}_1(\boldsymbol{x}_1)^T & 0 & -\boldsymbol{K}(\boldsymbol{x}_1, \boldsymbol{X}^T) & -1 \\
\vdots & & & \\
\boldsymbol{r}_1(\boldsymbol{x}_i)^T & 0 & -\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{X}^T) & -1 \\
\vdots & & & \\
\boldsymbol{r}_1(\boldsymbol{x}_N)^T & 0 & -\boldsymbol{K}(\boldsymbol{x}_N, \boldsymbol{X}^T) & -1
\end{bmatrix}, \tag{4.28}$$

[1] A similar problem has been considered in [278] with the kriging approach.

and

$$\boldsymbol{\beta}(Z) = \mathbf{0}, \tag{4.29}$$

for the problem (4.26).

Example. Consider two output variables y_1 and y_2, functions of a single input variable and related by $dy_1(x)/dx = y_2(x)$. The particular example studied here uses $y_1(x) = \sin(x)$ and $y_2(x) = \cos(x)$. The training data is composed of 31 samples in the interval $0 \leq x \leq 3$ with an additive centered Gaussian noise of standard deviation 0.2. Figure 4.5 shows the resulting approximations on both functions y_1 and y_2 with and without prior knowledge between these outputs considered. The prior knowledge $f_1'(x) = f_2(x)$ is included in (4.26) by (4.28) and (4.29) with a trade-off parameter set at $\lambda = 5$. The other hyperparameters are set at the same values for both methods and for both outputs: $C_1 = C_2 = 10$, $\sigma = 0.3$, $\varepsilon_1 = \varepsilon_2 = 0.1$. The MSE decreases dramatically when adding the prior knowledge above: from 0.016 to 0.0007 for y_1, and from 0.0106 to 0.0069 for y_2. Moreover the general shapes of the functions are recovered without being much affected by the noise.

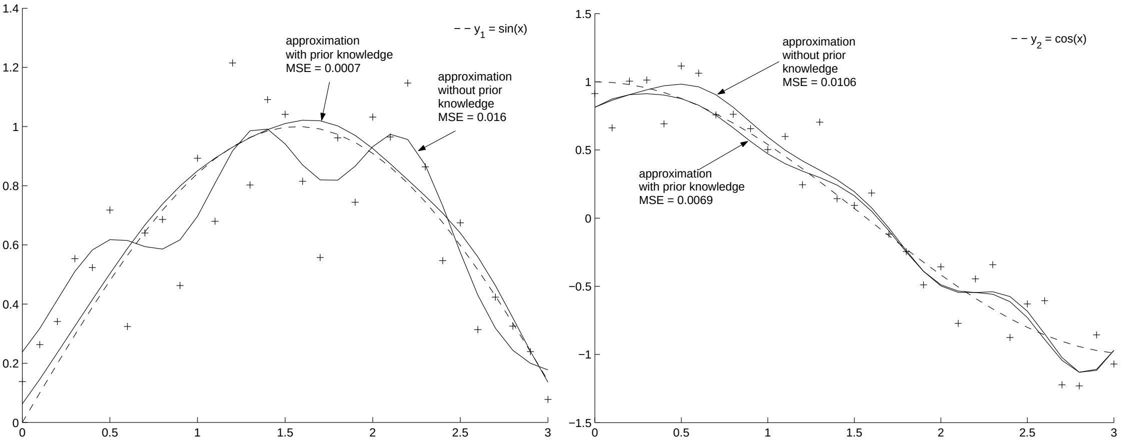


Figure 4.5: Simultaneous approximation of two functions $y_1 = \sin(x)$ (left) and $y_2 = \cos(x)$ (right) with prior knowledge between outputs: $f_1'(x) = f_2(x)$. The values for the different parameters are: $C_1 = C_2 = 10$, $\sigma = 0.3$, $\varepsilon_1 = \varepsilon_2 = 0.1$, $\lambda = 5$.

Multi-models for continuous piecewise function approximation

The following assumes that the function to approximate is composed of different subfunctions in different known regions of the input space. Though the method can be applied to multidimensional inputs, only the one-dimensional case is presented here for the sake of clarity. In this case, the prior knowledge leads to consider the approximation function $f(x)$ computed from a multitude of submodels $f_j(x)$ as

$$f(x) = \begin{cases} f_1(x) & \text{, if } l_1 \leq x \leq u_1 \\ f_2(x) & \text{, if } l_2 \leq x \leq u_2 \\ \vdots & \\ f_n(x) & \text{, if } l_n \leq x \leq u_n. \end{cases} \tag{4.30}$$

Moreover, if the function to approximate is continuous, then a coupling between the submodels can be expressed as

$$f_j(x) = f_{j+1}(x), \text{ for } x = u_j = l_{j+1}, \tag{4.31}$$

where f_j is the submodel for the input region $l_j \leq x \leq u_j$, trained only on this region (training set X_j).

The simultaneous training of the submodels with continuity prior enforced by hard constraints is, in this case, expressed as

$$\begin{array}{llll}
\min\limits_{\tilde{\boldsymbol{\alpha}},\boldsymbol{b},\boldsymbol{\xi}_j,\tilde{\boldsymbol{a}}\geq\mathbf{0}} & \multicolumn{3}{l}{\mathbf{1}^T\tilde{\boldsymbol{a}} + \sum_{j=1}^{m} C_j \mathbf{1}^T\boldsymbol{\xi}_j} \\
& & & \\
\text{s.t.} & -\boldsymbol{\xi}_j \leq & \boldsymbol{K}_j\boldsymbol{\alpha}_j + b_j\mathbf{1} - \boldsymbol{y}_j & \leq \boldsymbol{\xi}_j,\ j = 1,\dots,m \\
& 0 \leq & \mathbf{1}\varepsilon_j & \leq \boldsymbol{\xi}_j,\ j = 1,\dots,m \\
& -\tilde{\boldsymbol{a}} \leq & \tilde{\boldsymbol{\alpha}} & \leq \tilde{\boldsymbol{a}} \\
& & \boldsymbol{\Gamma}(Z)\tilde{\boldsymbol{\theta}} & = \boldsymbol{\beta}(Z),
\end{array} \tag{4.32}$$

where $\boldsymbol{K}_j = \boldsymbol{K}(\boldsymbol{X}_j^T, \boldsymbol{X}_j^T)$,

$$\boldsymbol{\Gamma}(Z) = \begin{bmatrix} \boldsymbol{K}(u_1, \boldsymbol{X}_1^T) & \mathbf{1} & -\boldsymbol{K}(u_1, \boldsymbol{X}_2^T) & -\mathbf{1} & 0 & \dots & & \\ \mathbf{0} & 0 & \boldsymbol{K}(u_2, \boldsymbol{X}_2^T) & \mathbf{1} & -\boldsymbol{K}(u_2, \boldsymbol{X}_3^T) & -\mathbf{1} & 0 & \dots \\ & & \ddots & & & & & \end{bmatrix}, \tag{4.33}$$

and

$$\boldsymbol{\beta}(Z) = \mathbf{0}. \tag{4.34}$$

Depending on the partitioning of the training set, the parameter vectors $\boldsymbol{\alpha}_j$ may be of different sizes. Thus, the formulation of the output (4.18) is not valid anymore. In this case, the outputs are computed separately by

$$f_j(x) = \boldsymbol{K}(x, \boldsymbol{X}_j^T)\boldsymbol{\alpha}_j + b_j. \tag{4.35}$$

Example: piecewise affine (PWA) function approximation. In this example, the continuous function to approximate is composed of three affine parts: $y = 0.5x$, for $0 \leq x \leq 3$; $y = 2x - 4.5$, for $3 \leq x \leq 6$; $y = -2x + 19.5$, for $6 \leq x \leq 10$. Note that in this example the partition of the input space is considered to be known[2]. The training set contains 101 points with additive Gaussian noise of standard deviation 1 and mean 0. Figure 4.6 shows the training data, the function, its approximation with a single SVR using RBF kernels and an approximation using the previously described method (4.32) with linear kernels and the settings (4.33) and (4.34). The test error decreases when considering a multitude of linear submodels (MSE= 0.078) instead of using a single RBF model (MSE= 0.117), whereas the number of SVs is equivalent and equals 3 for both methods. This number is highly correlated to the number of affine pieces in the global function. Comparing to independent linear models using only the information that the function is piecewise affine, the overall MSE is also reduced thanks to the prior knowledge on the continuity. Moreover, the continuity ensured by the proposed method cannot be obtained by independent linear models.

4.4 Adding inequality constraints

As seen in the previous section, equality constraints allow to include a large variety of prior knowledge. However, some interesting types of knowledge cannot be written in equalities and requires the use of inequalities such as, for instance, the monotonicity of the function: $f^{(1)}(x) \leq 0$ or $f^{(1)}(x) \geq 0$ for all x. This section proposes to include these types of knowledge on the function by considering inequality constraints such as

$$g_k(\boldsymbol{z}) \leq h_k(\boldsymbol{z}),\ \forall \boldsymbol{z} \in Z_k, \tag{4.36}$$

where the set Z_k contains the $|Z_k|$ points of interest for the kth set of constraints. As for equalities, these inequalities can be reformulated as inequality constraints that are linear w.r.t. the parameters $\boldsymbol{\theta} = [\boldsymbol{\alpha}^T\ b]^T$, of the general form

$$\boldsymbol{\Gamma}_k(Z_k)\boldsymbol{\theta} \leq \boldsymbol{\beta}_k(Z_k), \tag{4.37}$$

[2] The case where this partition is unknown will be considered in chapter 5 in the context of hybrid system identification.

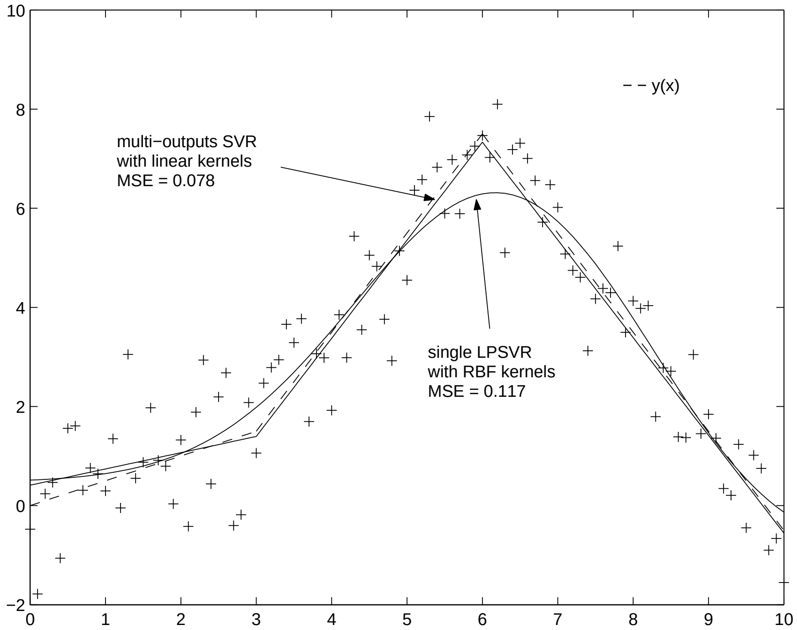


Figure 4.6: Approximation of a piecewise affine function. The values for the different parameters of the multi-outputs SVR are $C_j = 0.2$, $\varepsilon_j = 1$, for $j = 1, 2$, and 3. For the single SVR with RBF kernels these are: $C = 1$, $\varepsilon = 1$, $\sigma = 2$. Continuity is enforced by the prior knowledge $f_1(x) = f_2(x)$ for $x = 3$ and $f_2(x) = f_3(x)$ for $x = 6$.

that can then be introduced into (3.22) without changing the linear programming nature of the optimization problem. This leads to the linear programming SVR with inequality constraints (LPSVR-IC) trained by solving

$$
\begin{aligned}
&\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi}, \boldsymbol{a} \geq \mathbf{0}} \quad \mathbf{1}^T \boldsymbol{a} + C \mathbf{1}^T \boldsymbol{\xi} \\
&\text{s.t.} \quad \begin{array}{rcl}
-\boldsymbol{\xi} \leq & \boldsymbol{K\alpha} + b\mathbf{1} - \boldsymbol{y} & \leq \boldsymbol{\xi} \\
0 \leq & \mathbf{1}\varepsilon & \leq \boldsymbol{\xi} \\
-\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a} \\
& \boldsymbol{\Gamma}_k(Z_k)\boldsymbol{\theta} & \leq \boldsymbol{\beta}_k(Z_k), \ k = 1, \dots, n_{SC}.
\end{array}
\end{aligned}
\tag{4.38}
$$

To deal with the case where all the constraints cannot be satisfied simultaneously, the inequalities can also be enforced by soft constraints. Introducing a set of n_{SC} vectors $\boldsymbol{u}_k = [u_1^k \ \dots \ u_p^k \ \dots \ u_{|Z_k|}^k]^T$ of positive slack variables and a set of trade-off parameters λ_k leads to the linear programming SVR with soft inequality constraints (LPSVR-SIC) written as

$$
\begin{aligned}
&\min_{\boldsymbol{\alpha}, b, \boldsymbol{\xi}, \boldsymbol{a} \geq \mathbf{0}, \boldsymbol{u}_k \geq \mathbf{0}} \quad \mathbf{1}^T \boldsymbol{a} + C \mathbf{1}^T \boldsymbol{\xi} + \sum_{k=1}^{n_{SC}} \lambda_k \mathbf{1}^T \boldsymbol{u}_k \\
&\text{s.t.} \quad \begin{array}{rcl}
-\boldsymbol{\xi} \leq & \boldsymbol{K\alpha} + b\mathbf{1} - \boldsymbol{y} & \leq \boldsymbol{\xi} \\
0 \leq & \mathbf{1}\varepsilon & \leq \boldsymbol{\xi} \\
-\boldsymbol{a} \leq & \boldsymbol{\alpha} & \leq \boldsymbol{a} \\
& \boldsymbol{\Gamma}_k(Z_k)\boldsymbol{\theta} - \boldsymbol{\beta}_k(Z_k) & \leq \boldsymbol{u}_k, \ k = 1, \dots, n_{SC}.
\end{array}
\end{aligned}
\tag{4.39}
$$

4.4.1 Prior on the function, the derivatives and from a prior model

All the forms of prior knowledge described in section 4.3 can be considered with inequalities instead of equalities by using (4.38) or (4.39) and setting $\boldsymbol{\Gamma}_k$ and $\boldsymbol{\beta}_k$ as depicted for the equalities. This corresponds to the inclusion of bounds on the function or any derivative. For instance the following prior knowledge on the derivative $f^{(k)}(\boldsymbol{z}_p) \leq y_p^{(k)}$, for all $\boldsymbol{z}_p$ in Z_k, can be included in the learning by setting

$$\boldsymbol{\Gamma}_k(Z_k) = \begin{bmatrix} \boldsymbol{r}_k(\boldsymbol{z}_1)^T & 0 \\ \vdots & \\ \boldsymbol{r}_k(\boldsymbol{z}_p)^T & 0 \\ \vdots & \\ \boldsymbol{r}_k(\boldsymbol{z}_{|Z_k|})^T & 0 \end{bmatrix}, \quad \boldsymbol{\beta}_k(Z_k) = \begin{bmatrix} y_1^{(k)} \\ \vdots \\ y_p^{(k)} \\ \vdots \\ y_{|Z_k|}^{(k)} \end{bmatrix}, \tag{4.40}$$

for the linear programs (4.38) or (4.39). A practical use of inequalities occurs for instance when the function to estimate corresponds to a physical variable known to be positive. In this case, absurd approximations that yield negative values can be avoided by forcing bounds on the function as prior knowledge in the learning process.

Example: lower bound on the function. This example is taken from [182] and shows the benefit of including a lower bound on the function in a corrupted region of input space. The data to be modeled are generated by the sinc function with an additional Gaussian noise of mean 0 and standard deviation 0.5 for 32 points in the intervals $-3 \leq x \leq -1.43$ and $1.43 \leq x \leq 3$. Three points at $x = 0$ with output values $y = -1$, 0 and +1 are added to the training set in order to mislead the approximation. Three models are trained on these data: the standard LP-SVR without prior knowledge (3.22), the Knowledge-Based Kernel Approximation (KBKA) [182] and the proposed LP-SVR with prior knowledge (4.39). The knowledge originally used for KBKA is $f(x) \geq \text{sinc}(0.25)$ in the interval $-0.25 \leq x \leq 0.25$. In order to include this information in our method, three points $x = -0.25$, 0 and 0.25 are considered with the output value $y = \text{sinc}(0.25)$. The settings for constraints (4.37) are thus given by (4.12) with $\boldsymbol{Z}_0 = [-0.25\ 0\ 0.25]^T$ and $\boldsymbol{y}^0 = [\text{sinc}(0.25)\ \text{sinc}(0.25)\ \text{sinc}(0.25)]^T$. The hyperparameters of the KBKA are chosen as in [182]: $C = 13$, $\sigma = 0.7071$, $\mu_1 = 5$, $\mu_2 = 450$. The other models use the same values for C and σ. λ is arbitrarily set to 10. The resulting approximations for the three models appear in Figure 4.7. This example shows that even in the case of knowledge defined over polyhedral sets, our method, though requiring discretization of these sets, can yield comparable results to the ones obtained by Mangasarian's method. The MSE computed on 101 equally spaced points in the interval $[-3, 3]$ is respectively 0.176, 0.104 and 0.032 for the LP-SVR without knowledge, the KBKA and the proposed method.

Example: knowledge on the derivatives at particular points by inequalities. Taking the example of Sect. 4.3.2 and using the same prior knowledge, inequality constraints might help to approximate the second order derivative of sinc(x) at $x = 0$. In the setting of section 4.3.2, an equality constraint enforces $f^{(2)}(0) = -2$, which penalizes notably the values less than -2. On the contrary, the constraint $f^{(2)}(0) \leq -2$ is less restrictive and can let the derivative $f^{(2)}(0)$ go below -2 and reach $-\pi^2/3 \approx -3.29$. Figure 4.8 shows the approximations provided by (4.8) and (4.39) respectively using equality and inequality constraints. The two methods give comparable results. Actually, the inequality constraints bounding the magnitude of the derivatives from below are active and thus become equalities. This can be explained by the use of the RBF kernel and the minimization of the parameters α_i that penalizes non-smooth functions. Thus the algorithm looks for the smoothest function that satisfies the constraints, i.e. the function with minimum derivatives in magnitude, which corresponds in this case to the equality constraints $y^{(1)}(0) = 0$, $y^{(1)}(-0.5) = 1$ and $f^{(2)}(0) = -2$.

4.4.2 Prior knowledge between outputs

The multi-outputs SVR of section 4.3.4 can also consider prior knowledge on a linear combination of the m outputs as inequalities

$$\sum_{j=1}^{m} \mu_j f_j(\boldsymbol{z}) \leq h(\boldsymbol{z}), \ \forall \boldsymbol{z} \in Z, \tag{4.41}$$

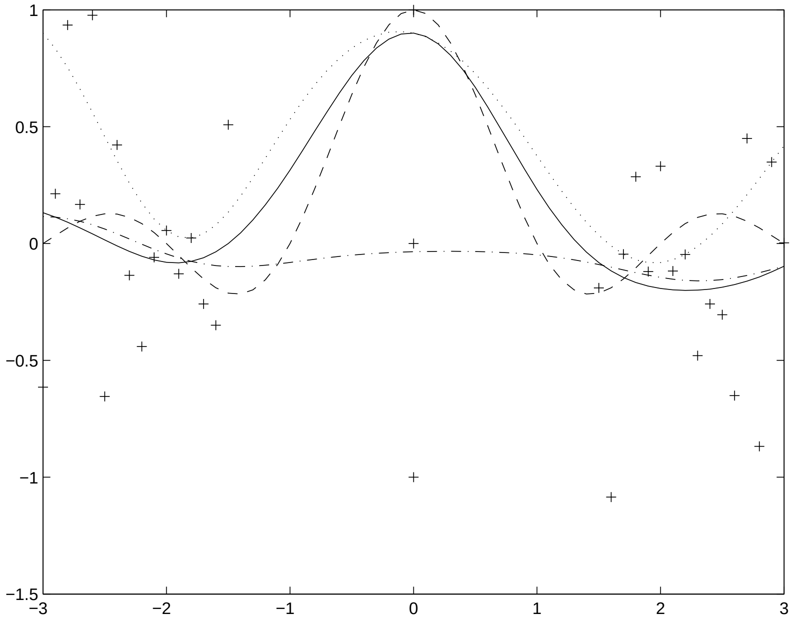


Figure 4.7: Approximation of the sinc function (dashed line) with additional wrong data at $x = 0$. The model without knowledge (dash-dotted line) cannot recover the true shape around $x = 0$. Prior knowledge stating that $f(x) \geq \text{sinc}(0.25)$ in the interval $-0.25 \leq x \leq 0.25$ is included either by the method of [182] (dotted line) or by the proposed method (solid line). The values for the different parameters are: $\lambda = 10$, $C = 13$, $\sigma = 0.7071$, $\varepsilon = 0.01$.

for an arbitrary function h. In this case, the training of a multi-outputs SVR is performed by solving, as in (4.26), the following problem

$$\begin{array}{llcll} & \displaystyle\min_{\tilde{\boldsymbol{\alpha}}, \boldsymbol{b}, \boldsymbol{\xi}_j, \tilde{\boldsymbol{a}} \geq \mathbf{0}, \boldsymbol{u} \geq \mathbf{0}} & \multicolumn{3}{l}{\mathbf{1}^T \tilde{\boldsymbol{a}} + \sum_{j=1}^{m} C_j \mathbf{1}^T \boldsymbol{\xi}_j + \lambda \mathbf{1}^T \boldsymbol{u}} \\ \text{s.t.} & -\boldsymbol{\xi}_j \leq & \boldsymbol{K}\boldsymbol{\alpha}_j + b_j \mathbf{1} - \boldsymbol{y}_j & \leq \boldsymbol{\xi}_j, \ j = 1, \dots, m \\ & 0 \leq & \mathbf{1}\varepsilon_j & \leq \boldsymbol{\xi}_j, \ j = 1, \dots, m \\ & -\tilde{\boldsymbol{a}} \leq & \tilde{\boldsymbol{\alpha}} & \leq \tilde{\boldsymbol{a}} \\ & & \boldsymbol{\Gamma}(Z)\tilde{\boldsymbol{\theta}} - \boldsymbol{\beta}(Z) & \leq \boldsymbol{u}, \end{array} \tag{4.42}$$

with the same setup (4.25) and (4.20) as for equalities. Of course, hard constraints can also be considered here, leading to

$$\begin{array}{llcll} & \displaystyle\min_{\tilde{\boldsymbol{\alpha}}, \boldsymbol{b}, \boldsymbol{\xi}_j, \tilde{\boldsymbol{a}} \geq \mathbf{0}} & \multicolumn{3}{l}{\mathbf{1}^T \tilde{\boldsymbol{a}} + \sum_{j=1}^{m} C_j \mathbf{1}^T \boldsymbol{\xi}_j} \\ \text{s.t.} & -\boldsymbol{\xi}_j \leq & \boldsymbol{K}\boldsymbol{\alpha}_j + b_j \mathbf{1} - \boldsymbol{y}_j & \leq \boldsymbol{\xi}_j, \ j = 1, \dots, m \\ & 0 \leq & \mathbf{1}\varepsilon_j & \leq \boldsymbol{\xi}_j, \ j = 1, \dots, m \\ & -\tilde{\boldsymbol{a}} \leq & \tilde{\boldsymbol{\alpha}} & \leq \tilde{\boldsymbol{a}} \\ & & \boldsymbol{\Gamma}(Z)\tilde{\boldsymbol{\theta}} & \leq \boldsymbol{\beta}(Z). \end{array} \tag{4.43}$$

The prior knowledge as inequalities between multiple outputs is thus easily incorporated in a linear program by adding inequalities in a general form similar to the one used for the other types of prior knowledge.

Example: ordered functions. Consider the problem of simultaneously approximating two scalar functions while knowing that one is greater than the other. In this case, the prior knowledge

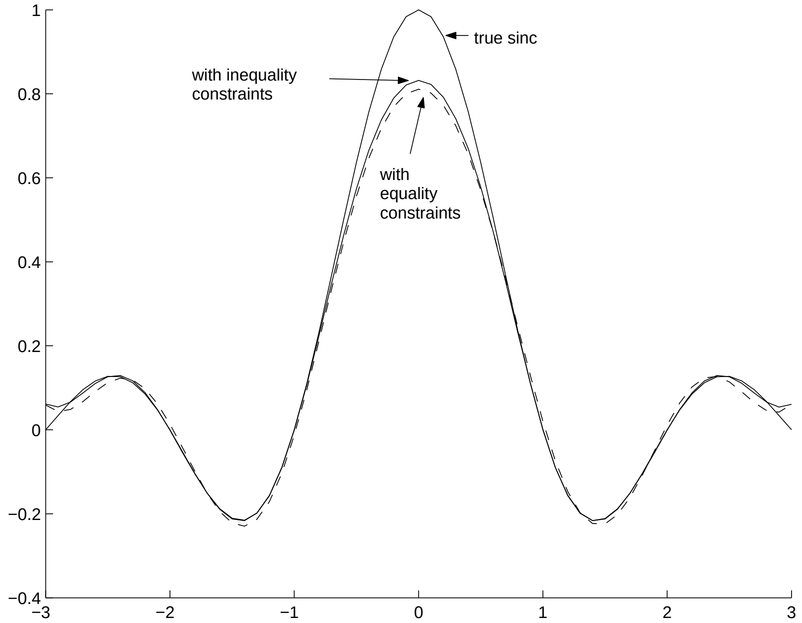


Figure 4.8: Approximation of the sinc function without data for x in the interval $-1 < x < +1$ and with prior knowledge on the derivatives at 2 particular points ($y^{(1)}(0) \leq 0.1$, $y^{(1)}(-0.5) \geq 1$, $y^{(2)}(0) \leq -2$) incorporated via inequality constraints (solid line) and equalities (dash line). The values for the different parameters are: $\lambda = 50, C = 10, \sigma = 0.5, \varepsilon = 0.001$.

$f_1(x) \leq f_2(x)$ may help to enhance the quality of the model. A somehow similar setting is studied in [177]. Here, the functions $y_1 = \sin(x)$ and $y_2 = \sin(x) + 0.1$ are approximated based on noisy data. A Gaussian noise of mean zero and standard deviation 0.2 has been added to both functions for 71 samples in the interval $0 \leq x \leq 7$. Two outliers are added to the training set of y_1: $(3.7, 0)$ and $(3.8, 0)$, as shown on Figure 4.9. On this problem, training two independent models by (4.6), with the parameters set to $C = 10$, $\varepsilon = 0.1$ and $\sigma = 0.5$, yields approximations that are significantly in violation of the prior knowledge. At the contrary, using the proposed method with hard constraints (4.43) allows to obtain better approximations that respect the constraints, as shown on Fig. 4.9. Moreover, the effect of the outliers on y_1 is reduced thanks to the relation imposed between f_1 and f_2 and the fact that the training data for y_2 do not contain outliers. The MSE for y_1 is reduced from 0.0102 to 0.0034, and for y_2 from 0.0072 to 0.0035.

4.5 Numerical examples

This section provides three examples of practical use for the proposed method. The two first examples deal with the identification of simulated nonlinear dynamical systems. The last example shows the benefit of the method on a real-life application, namely, the estimation of in-cylinder residual gas fraction in spark ignition engines.

4.5.1 Nonlinear dynamical system with known equilibrium points

Consider the discrete-time nonlinear dynamical system taken from [197] and described by

$$y_k = \frac{y_{k-1} y_{k-2} (y_{k-1} + 2.5)}{1 + y_{k-1}^2 + y_{k-2}^2} + u_{k-1} + e_k, \tag{4.44}$$

where y_k and u_k are respectively the output and the input of the system at step k. In [197], the two equilibrium points in state space $(y_{k-1} = 0, y_{k-2} = 0)$ and $(y_{k-1} = 2, y_{k-2} = 2)$ of the unforced

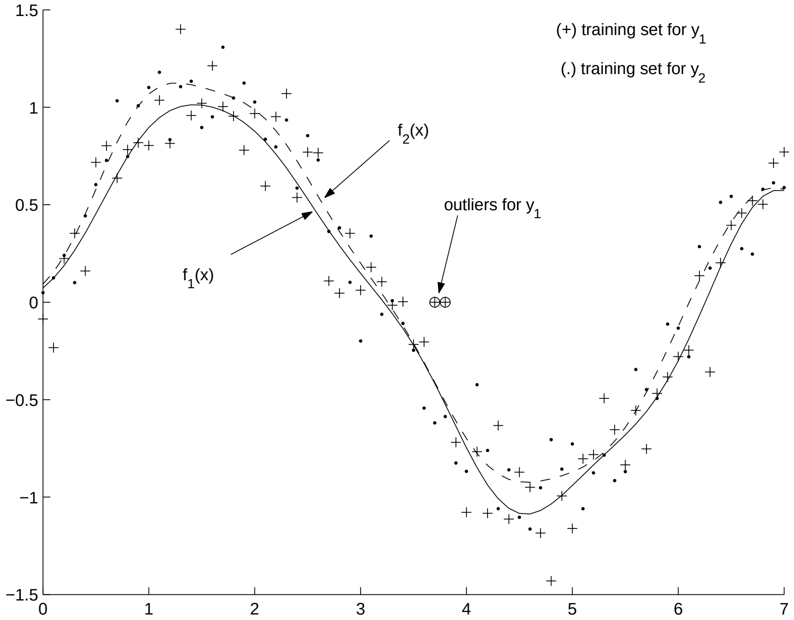


Figure 4.9: Approximation of two sine functions $y_1 = \sin(x)$ and $y_2 = \sin(x) + 0.1$ with the prior knowledge $f_1(x) \leq f_2(x)$. The values for the different parameters are: $C_1 = C_2 = 1$, $\sigma = 0.5$, $\varepsilon_1 = \varepsilon_2 = 0.1$.

system (for $u_{k-1} = 0$) are considered to be known. These points are the solutions in y of the equation (4.44), with $y_k = y_{k-1} = y_{k-2} = y$ and for $u_{k-1} = 0$.

A trajectory is generated from the initial condition ($y_0 = 1, y_{-1} = 1$) by 100 iterations of the system (4.44) with additional Gaussian noise e_k of mean 0 and standard deviation 0.1 for a uniformly distributed random input sequence in the interval $[-2, 2]$. This trajectory is used to build the training set with regression vectors $\boldsymbol{x}_i = [y_{i-1}\ y_{i-2}\ u_{i-1}]^T$ and targets y_i, for $i = 1, \dots, 100$. Two NARX models are trained on these data: a standard LP-SVR without prior knowledge (3.22) and a LP-SVR with hard constraints (4.8) on the equilibrium points such that $f([0\ 0\ 0]^T) = 0$ and $f([2\ 2\ 0]^T) = 0$. For both algorithms, Gaussian RBF kernels are used and the hyperparameters are arbitrarily chosen to be $C = 10$, $\sigma = 1$ and $\varepsilon = 0.01$. The 100-step-ahead prediction error MSE_{100} (1.28) is evaluated on the test set built from a different trajectory of 100 points generated by (4.44) for the initial condition ($y_0 = 0.5, y_{-1} = 0.5$), a sinusoidal input $u_k = \sin(2\pi k/25)$, and $e_k = 0$. Table 4.1 shows the average and standard deviation of MSE_{100} computed over 100 runs. A gain in performance is observed when prior knowledge on equilibrium points is used as the average of MSE_{100} decreases by 20.4 %.

Table 4.1: Average and standard deviation of the 100-step-ahead prediction error MSE_{100} over 100 runs for the models of the nonlinear dynamical system (4.44).

Model	MSE_{100}
with prior knowledge	0.354 ± 0.356
without prior knowledge	0.445 ± 0.457

4.5.2 Nonlinear dynamical system with prior knowledge on the input function

Consider the discrete-time nonlinear dynamical system also taken from [197] and described by

$$y_k = \frac{y_{k-1}}{1+y_{k-1}^2} + u_{k-1}^3 + e_k. \tag{4.45}$$

In this example, the predictions are based on two variables only, leading to the regression vectors $\boldsymbol{x}_i = [y_{i-1}\ u_{i-1}]^T$ for the NARX model f. The only prior knowledge extracted from the expression of the system is that the partial derivative of f w.r.t. the input u_{i-1} must be an even function.

As in the previous example, the training set is built from a trajectory that is generated from the initial condition $y_0 = 0$ by 100 iterations of the system (4.45) with additional Gaussian noise e_k of mean 0 and standard deviation 0.1 for a uniformly distributed random input sequence in the interval $[-2, 2]$. Two models with Gaussian RBF kernels are trained on these data: a standard LP-SVR without prior knowledge (3.22) and a LP-SVR (4.9) with prior knowledge on the derivatives such that $\partial f([y\ u]^T)/\partial u = \partial f([y\ -u]^T)/\partial u$. To include this information, the partial derivative of f is computed by (A.5) at every point $\boldsymbol{x}_i$ of the training set to build the matrix $\boldsymbol{R} = [\boldsymbol{r}_1(\boldsymbol{x}_1)\ \dots\ \boldsymbol{r}_1(\boldsymbol{x}_i)\ \dots\ \boldsymbol{r}_1(\boldsymbol{x}_N)]^T$ and also at the points $\tilde{\boldsymbol{x}}_i = [1\ \ -1]\boldsymbol{x}_i$, $i = 1, \dots, N$, to build a similar matrix $\tilde{\boldsymbol{R}}$. Then the prior knowledge $\boldsymbol{R\alpha} = \tilde{\boldsymbol{R}}\boldsymbol{\alpha}$ is included in the learning (4.9) by setting, as in (4.16), $\boldsymbol{\Gamma} = [(\boldsymbol{R} - \tilde{\boldsymbol{R}})\ \boldsymbol{0}]$ and $\boldsymbol{\beta} = \boldsymbol{0}$. The hyperparameters are arbitrarily chosen to be $C = 10$, $\sigma = 1$ and $\varepsilon = 0.01$. The trade-off parameter λ is set to 1. The one-step-ahead prediction error MSE (1.15) is evaluated on the test set built from a different trajectory generated from the initial condition $y_0 = 0$, the input $u_k = \sin(2\pi k/25) + \sin(2\pi k/10)$ and $e_k = 0$, as in [197]. Table 4.2 shows the average and standard deviation of the MSE computed over 100 runs. A gain in performance is observed when prior knowledge is used as the average MSE decreases by 25.7 %.

Table 4.2: Average and standard deviation of the one-step-ahead prediction error MSE over 100 runs for the models of the nonlinear dynamical system (4.45).

Model	MSE
with prior knowledge	1.158 ± 0.755
without prior knowledge	1.560 ± 0.667

4.5.3 Estimation of in-cylinder residual gas fraction

The method is now applied to a real-life problem. An extended study of the various ways of incorporating the prior knowledge from simulation data and the effect of the number of training samples was first performed and can be found in [26]. These results, not recalled here, provided the ground for the development of the final results presented in [27] and described below.

Problem description

The application deals with the estimation of residual gases in the cylinders of Spark Ignition (SI) engines with Variable Camshaft Timing (VCT). VCT allows the timing of the intake and exhaust valves to be changed while the engine is in operation. VCT is used to improve performance in terms of emissions, fuel economy, peak torque, and peak power [123].

The air path control of SI engines is a crucial task because the torque provided by the engine is directly linked to the air mass trapped in the cylinders [55]. When considering new air actuators such as VCT, the estimation of in-cylinder air mass is more involved than for basic SI engines. Indeed, VCT authorizes phenomena such as air scavenging (from intake to exhaust manifolds, with turbocharging) or backflow (from exhaust manifold to cylinders).

In this context, it is important to estimate the residual gas mass fraction

$$\chi_{res} = \frac{m_{res}}{m_{tot}}, \tag{4.46}$$

where m_{tot} is the total gas mass trapped in the cylinder and m_{res} is the mass of residual gases, which are burned gases present in the cylinder when the valves are closed before the new combustion and which are due to the dead volumes or the backflow. Knowing this fraction allows to control torque as well as pollutant emissions.

There is no standard sensor to measure this fraction online. There exists a corresponding mean value model, proposed by Fox et al. [83], that includes some constants to be identified from experiments on a particular engine. Only average values of the variables over the cycle are considered in mean value models. In this context, the residual gas mass fraction χ_{res} can be expressed as a function of the engine speed N_e, the ratio p_{man}/p_{exh}, where p_{man} and p_{exh} are respectively the intake manifold pressure and the exhaust pressure, and an overlapping factor OF, which is an image of the time during which the valves are opened together. This residual gas fraction also depends, but only slightly, on the opening and closing instants of the valves, which are not taken into account here, as in [83].

The available data are provided, on one hand, by the modeling and simulation environment Amesim [121], which uses a high frequency zero-dimensional thermodynamic model [94] and, on the other hand, by off line measurements, which are accurate, but complex and costly to obtain, by direct in-cylinder sampling [94]. The problem is thus as follows. How to obtain a simple, embeddable, black box model with a good accuracy and a large validity range for the real engine, from precise real measurements as less numerous as possible and a representative, but possibly biased, prior simulation model?

The problem thus posed, although particular, is very representative of numerous situations met in engine control, and more generally in engineering, where complex models, more or less accurate, exist and where the experimental data which can be used for calibration are difficult or expensive to obtain.

Experimental setup

Three datasets are built from the available data composed of 26 experimental samples plus 26 simulation samples:

- the training set $(\boldsymbol{X}, \boldsymbol{y})$ composed of a limited amount of real data (N samples),
- the test set composed of independent real data ($26 - N$ samples),
- the simulation set $(\boldsymbol{Z}, \tilde{\boldsymbol{y}})$ composed of data provided by the simulator (26 samples).

It must be noted that the inputs of the simulation data do not exactly coincide with the inputs of the experimental data. Various sizes N of the training set will be considered in order to study the effect of the number of training samples on the model.

The residual gas mass fraction χ_{res}, given in percentages, takes values in the range $[5, 30]$. The ranges of values for the three inputs are: N_e (rpm) $\in \{1000, 2000\}$, $p_{man}/p_{exh} \in [0.397, 0.910]$ and OF $(°CA/m) \in [0, 2.8255]$. The datasets are shown in Figure 4.10.

The simulator being biased but approximating rather well the overall shape of the function, the prior knowledge will also be incorporated on the derivatives. In order to be able to evaluate the prior derivatives a *prior model*, $f^{prior}(\boldsymbol{x}) = \sum_{i=1}^{N^{pr}} \alpha_i^{prior} k(\boldsymbol{x}, \boldsymbol{z}_i) + b^{prior}$, is first trained on the simulation data only. As in section 4.3.3, this prior model is then used to provide the values $y'_p = \boldsymbol{r}_1^{pr}(\boldsymbol{z}_p)^T \boldsymbol{\alpha}^{prior}$ of the derivative w.r.t. the input p_{man}/p_{exh}, at the points $\boldsymbol{z}_p$ of the simulation set. Define the matrix $\boldsymbol{R}(\boldsymbol{Z}^T, \boldsymbol{X}^T) = [\boldsymbol{r}_1(\boldsymbol{z}_1) \ \dots \ \boldsymbol{r}_1(\boldsymbol{z}_p) \ \dots \ \boldsymbol{r}_1(\boldsymbol{z}_{N^{pr}})]^T$, where $\boldsymbol{r}_1(\boldsymbol{x})$ corresponds to the derivative w.r.t. the input p_{man}/p_{exh} of a kernel expansion $f(\boldsymbol{x}) = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\boldsymbol{\alpha} + b$ trained on the observation matrix $\boldsymbol{X}$. Then the vector of prior derivatives $\boldsymbol{y}' = [y'_1 \ \dots \ y'_p \ \dots \ y'_{N^{pr}}]^T$, can be computed by $\boldsymbol{y}' = \boldsymbol{R}(\boldsymbol{Z}^T, \boldsymbol{Z}^T)\boldsymbol{\alpha}^{pr}$.

Different situations and various ways of incorporating the simulation data are considered in the following models.

1. *Experimental model.* The simulation data are not available. Standard LP-SVR training (3.22) on the training set $(\boldsymbol{X}, \boldsymbol{y})$ only is used to determine the model.

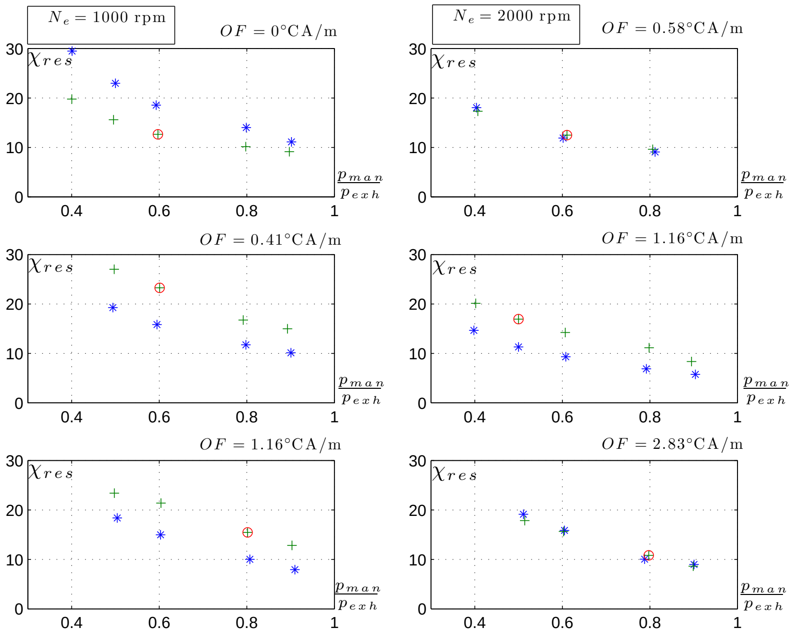


Figure 4.10: Residual gas mass fraction χ_{res} in percentages as a function of the ratio p_{man}/p_{exh} for two engine speeds N_e and different overlapping factors OF. The 26 experimental data are represented by plus signs (+), with a superposed circle ($\oplus$) for the training samples when $N = 6$. The 26 simulation data appear as asterisks ($*$).

2. *Prior model.* The experimental data are not available. LP-SVR training (3.22) on the simulation set $(\boldsymbol{Z}, \tilde{\boldsymbol{y}})$ only is used to determine the model f^{prior}.

3. *Mixed model.* All the data are available and mixed together. LP-SVR training (3.22) on the training set extended with the simulation data $([\boldsymbol{X}^T\ \boldsymbol{Z}^T]^T, [\boldsymbol{y}^T\ \tilde{\boldsymbol{y}}^T]^T)$ is used to determine the model. This may be the first and most simple approach, similar to the virtual sample approach (see section 2.4.1), which has been extensively studied in pattern recognition. However, in the case considered here, the simulation data can be biased as the physical model is not fully accurate.

4. *O-model.* The simulation data $(\boldsymbol{Z}, \tilde{\boldsymbol{y}})$ are considered as approximate knowledge on *Output* values. Algorithm (4.10) is used with (4.12) to train the model. This approach allows to take into account a possible bias of ε_p between the simulation and real data.

5. *OP-model.* Same as the *O-model* but with the simulation data as *Potential* SVs. Algorithm (4.10), with $\boldsymbol{K}$ replaced by $\boldsymbol{K}(\boldsymbol{X}^T, [\boldsymbol{X}^T\ \boldsymbol{Z}^T])$, is used to train the model. This approach allows to compensate for the local behavior of RBF kernels which may become a serious drawback when reducing the number of training samples.

6. *D-model.* The simulation data $(\boldsymbol{Z}, \tilde{\boldsymbol{y}})$ are used to build the *prior model* f^{prior}, which is then used to provide prior knowledge on *Derivative* values w.r.t. the input p_{man}/p_{exh}, taken at the simulation input points. Algorithm (4.9), with derivative constraints set as in (4.17), is used to train the final model. This approach allows to retain the overall shape of the *prior model* while fitting it to the available real data.

7. *DP-model.* Same as the *D-model* but with the simulation data $(\boldsymbol{Z}, \tilde{\boldsymbol{y}})$ added as *Potential* SVs.

8. *OD-model.* The simulation data $(\boldsymbol{Z}, \tilde{\boldsymbol{y}})$ are both considered as prior knowledge on *Output* values as for the *O-model* and used to build a *prior model* in order to give prior knowledge on *Derivative* values as for the *D-model.* Two sets of constraints respectively set as in (4.12) and (4.17) are used to train the OD-model.

9. *ODP-model.* Same as the *OD-model* but with the simulation data $(\boldsymbol{Z}, \tilde{\boldsymbol{y}})$ as *Potential* SVs.

Table 4.3 summarizes the models with the corresponding settings for the training algorithms.

Table 4.3: Setup of the optimization problems for the training of the models of the residual gas fraction.

Model	Main problem: fitting the data			Additional constraints: prior knowledge		
	Eq.	$\boldsymbol{K}$	$\boldsymbol{y}$	Eq.	$\boldsymbol{\Gamma}(Z)$	$\boldsymbol{\beta}(Z)$
1 exp. model	(3.22)	$\boldsymbol{K}(\boldsymbol{X}^T, \boldsymbol{X}^T)$	$\boldsymbol{y}$			
2 prior model	(3.22)	$\boldsymbol{K}(\boldsymbol{Z}^T, \boldsymbol{Z}^T)$	$\tilde{\boldsymbol{y}}$			
3 mixed model	(3.22)	$\boldsymbol{K}(\begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Z}\end{bmatrix}^T, \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Z}\end{bmatrix}^T)$	$\begin{bmatrix}\boldsymbol{y}\\ \tilde{\boldsymbol{y}}\end{bmatrix}$			
4 O-model	(4.10)	$\boldsymbol{K}(\boldsymbol{X}^T, \boldsymbol{X}^T)$	$\boldsymbol{y}$	(4.12)	$[\boldsymbol{K}(\boldsymbol{Z}^T, \boldsymbol{X}^T)\ \boldsymbol{1}]$	$\tilde{\boldsymbol{y}}$
5 OP-model	(4.10)	$\boldsymbol{K}(\boldsymbol{X}^T, \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Z}\end{bmatrix}^T)$	$\boldsymbol{y}$	(4.12)	$[\boldsymbol{K}(\boldsymbol{Z}^T, \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Z}\end{bmatrix}^T)\ \boldsymbol{1}]$	$\tilde{\boldsymbol{y}}$
6 D-model	(4.9)	$\boldsymbol{K}(\boldsymbol{X}^T, \boldsymbol{X}^T)$	$\boldsymbol{y}$	(4.17)	$[\boldsymbol{R}(\boldsymbol{Z}^T, \boldsymbol{X}^T)\ \boldsymbol{0}]$	$\boldsymbol{y}'$
7 DP-model	(4.9)	$\boldsymbol{K}(\boldsymbol{X}^T, \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Z}\end{bmatrix}^T)$	$\boldsymbol{y}$	(4.17)	$[\boldsymbol{R}(\boldsymbol{Z}^T, \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Z}\end{bmatrix}^T)\ \boldsymbol{0}]$	$\boldsymbol{y}'$
8 OD-model	(4.10)	$\boldsymbol{K}(\boldsymbol{X}^T, \boldsymbol{X}^T)$	$\boldsymbol{y}$	(4.12)	$[\boldsymbol{K}(\boldsymbol{Z}^T, \boldsymbol{X}^T)\ \boldsymbol{1}]$	$\tilde{\boldsymbol{y}}$
	& (4.9)			(4.17)	$[\boldsymbol{R}(\boldsymbol{Z}^T, \boldsymbol{X}^T)\ \boldsymbol{0}]$	$\boldsymbol{y}'$
9 ODP-model	(4.10)	$\boldsymbol{K}(\boldsymbol{X}^T, \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Z}\end{bmatrix}^T)$	$\boldsymbol{y}$	(4.12)	$[\boldsymbol{K}(\boldsymbol{Z}^T, \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Z}\end{bmatrix}^T)\ \boldsymbol{1}]$	$\tilde{\boldsymbol{y}}$
	& (4.9)			(4.17)	$[\boldsymbol{R}(\boldsymbol{Z}^T, \begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Z}\end{bmatrix}^T)\ \boldsymbol{0}]$	$\boldsymbol{y}'$

Remark 11. *To simplify the tuning of the hyperparameters, all the linear programs used to train the models are modified to implement* balanced *formulations of the objective functions as in remark 9.*

Remark 12. *Note that the simulation data points $\boldsymbol{z}_p$ at which the prior knowledge is considered are not the same as the points $\boldsymbol{x}$ found in the training set. Thus the methods of [158] or [157] could not be used here.*

These models are evaluated on the basis of three indicators defined below.

- *RMSE test*: root mean square error on the test set (N_{test} samples)
- *RMSE total*: root mean square error on the whole real dataset ($N + N_{test}$ samples)
- *MAE total*: maximum absolute error on the whole real dataset ($N + N_{test}$ samples)

Before training, the variables are normalized with respect to their mean and standard deviation. When both experimental and simulation data are available, the simulation data are preferred since they are supposed to cover a wider region of the input space. Thus, the mean and standard deviation are determined on the simulation data for all the models except for the *experimental model*, in which case the training set must be used to determine the normalization parameters.

The hyperparameters of the method can be classified in two categories: internal parameters, such as the kernel parameters σ or d; and the user-level parameters, such as C, λ (or λ_1 and λ_2), ε and ε_p, allowing to tune the algorithm in accordance with some prior knowledge. As kernel parameter tuning is out of the scope of the paper, the kernel parameters are set according to the following heuristics. Since all standard deviations of the inputs equal 1 after normalization, the RBF kernel width σ is set to 1. The degree d of the polynomial kernels is set to the lowest value yielding a reasonable training error on $N = 15$ samples, i.e. $d = 4$. For the threshold parameters of the ε-insensitive loss functions, ε is set to 0.001 in order to approximate the real data well. In addition, when using prior knowledge on output values (*O-*, *OP-*, *OD-* and *ODP-models*) and setting ε_p, one has both the real and simulation data at hand. Thus, it is possible to compute an estimate of the maximum absolute error (MAE) obtained by the simulator by looking at the MAE obtained by the *prior model* on the training set (available real data). ε_p is then set accordingly by taking into account the normalization step ($\varepsilon_p = 1.6$). Regarding the remaining hyperparameters, the following sections respectively discuss the use of fixed values, the sensitivity of the algorithm to their values, and their tuning by cross-validation.

Table 4.4: Errors on the residual gas mass fraction for various training set sizes N with $\lambda = C = 1000$ and a RBF kernel. ’–’ appears when the result is irrelevant (model mostly constant).

N	Model	*RMSE test*	*RMSE total*	*MAE total*
15	1 (experimental model)	1.69	1.54	**3.77**
	2 (prior model)	5.02	4.93	9.74
	3 (mixed model)	5.07	4.22	7.80
	4 (O-model)	2.26	1.47	4.11
	5 (OP-model)	3.23	2.10	4.95
	6 (D-model)	2.89	2.67	8.92
	7 (DP-model)	**1.07**	**1.24**	4.84
	8 (OD-model)	2.89	2.67	8.91
	9 (ODP-model)	**1.07**	**1.24**	4.84
6	1 (experimental model)	4.89	4.42	10.16
	2 (prior model)	4.86	4.93	9.74
	3 (mixed model)	4.84	4.88	9.75
	4 (O-model)	3.86	3.38	9.78
	5 (OP-model)	4.43	3.88	9.08
	6 (D-model)	3.99	3.50	10.42
	7 (DP-model)	**2.24**	**1.96**	5.99
	8 (OD-model)	3.15	2.83	**5.79**
	9 (ODP-model)	**2.24**	**1.96**	5.99
3	1 (experimental model)	–	–	–
	2 (prior model)	4.92	4.93	9.74
	3 (mixed model)	4.89	4.86	9.75
	4 (O-model)	–	–	–
	5 (OP-model)	5.80	5.46	11.53
	6 (D-model)	–	–	–
	7 (DP-model)	38.5	36.2	70.0
	8 (OD-model)	–	–	–
	9 (ODP-model)	**3.24**	**3.05**	**6.04**

Fixed hyperparameters

The hyperparameters C, λ (or λ_1 and λ_2) are first set as follows. One goal of the problem is to obtain a model that is accurate on both the training and test samples (the training points are part of the performance index *RMSE total*). Thus C is set to a large value ($C = 1000$) in order to ensure a good approximation of the training points. The additional hyperparameter of the method λ is set to $\lambda = C = 1000$ to give as much weight to the prior knowledge than to the data. For the *OD-* and *ODP-models*, the two types of prior knowledge are considered with the same weight, i.e. $\lambda_1 = \lambda_2 = C = 1000$.

The first experiments are performed with Gaussian RBF kernels, which exhibit a local behavior. The number N of points retained as training samples is first set to 15, which is about half of the available experimental data. The results in this setting appear at the top of Table 4.4. These show that the model of the simulator, the *prior model*, is actually biased and leads to a large error when tested on the real data. As a consequence, the *mixed model* that simply considers simulation data as training samples cannot yield good results from inhomogeneous and contradictory data. Regarding the various forms of prior knowledge considered, it seems that the information on the derivative is the most relevant. However, in order to improve the model, potential support vectors must be added, as in the *DP-* and *ODP-models*.

Now, the effect of reducing the number of training samples to $N = 6$ is studied. The results in Table 4.4 show that a good *experimental model* cannot be determined with so few samples. On the other hand, adding prior knowledge allows to obtain good results as shown on Figure 4.11 for the *ODP-model*. Incorporating prior information on the derivative, as in the *DP-* and *ODP-models*, is more efficient than simply incorporating simulation samples with a large threshold ε_p on the corresponding error as implemented in the *O-* and *OP-models*. It must be noted that the test error of the *DP-* and *ODP-models* is less than half the test errors obtained by the *prior* and *mixed models*, which correspond to the standard methods to include simulation data.

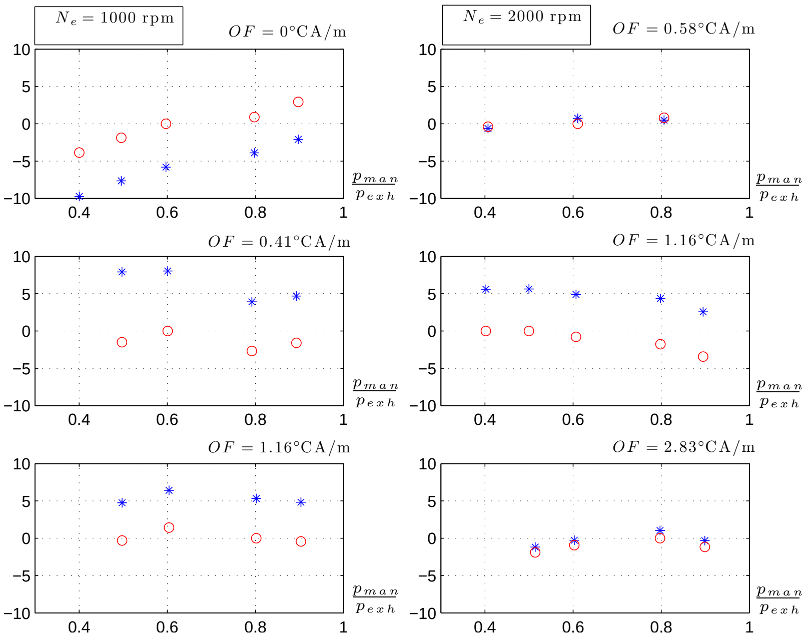


Figure 4.11: Errors on the experimental data (training and test samples) for the *prior model* ($*$) and the *ODP-model* ($\circ$) trained with a RBF kernel on only six samples ($N = 6$).

When the number of training points becomes too small ($N = 3$), the *O-*, *D-* and *OD-models*, that do not incorporate the simulation data as potential SVs, become almost constant and thus inefficient. This is due to the fact that these models have not enough free parameters (only 3 plus a bias term) and corresponding RBFs; thus they cannot accurately model the data. On the contrary, the *OP-*, *DP-* and *ODP-models* do not suffer from this problem. However, as discussed below, the method becomes more sensitive to the tuning of λ which leads to a very large error for the *DP-model*. On the other hand, the *ODP-model* achieves a reasonable performance considering that only $N = 3$ experimental samples were used.

Table 4.5 shows that, for a non-local kernel, i.e. a polynomial kernel of degree 4, the method also allows to improve the model. Due to the non-local behavior of this kernel, the effect of considering the simulation data as potential support vectors is less obvious and the *OD-model* without additional SVs yields the best performance for $N = 6$. A common feature of the experiments with the two kernels is the superiority of the prior knowledge applied to the derivatives compared to the output values.

Table 4.5: Errors on the residual gas mass fraction for various training set sizes N with $\lambda = C = 1000$ and a polynomial kernel.

N	Model	*RMSE test*	*RMSE total*	*MAE total*
15	1 (experimental model)	2.06	2.17	**4.89**
	2 (prior model)	4.92	4.89	9.77
	3 (mixed model)	4.97	4.55	9.73
	4 (O-model)	3.58	2.33	5.58
	5 (OP-model)	8.97	5.83	15.99
	6 (D-model)	2.70	2.66	7.25
	7 (DP-model)	**1.29**	**1.66**	5.10
	8 (OD-model)	2.70	2.66	7.25
	9 (ODP-model)	**1.29**	**1.66**	5.10
6	1 (experimental model)	13.79	12.31	29.59
	2 (prior model)	4.84	4.89	9.77
	3 (mixed model)	4.84	4.87	9.77
	4 (O-model)	2.93	2.57	6.17
	5 (OP-model)	4.79	4.20	14.87
	6 (D-model)	1.88	1.70	3.54
	7 (DP-model)	2.42	2.12	6.34
	8 (OD-model)	**1.75**	**1.60**	**3.53**
	9 (ODP-model)	2.42	2.13	6.35
3	1 (experimental model)	6.39	6.06	14.25
	2 (prior model)	4.87	4.89	9.77
	3 (mixed model)	4.92	4.89	9.78
	4 (O-model)	5.65	5.31	11.25
	5 (OP-model)	6.01	5.65	11.38
	6 (D-model)	5.59	5.26	10.11
	7 (DP-model)	12.52	11.77	27.65
	8 (OD-model)	5.22	4.93	11.21
	9 (ODP-model)	**3.59**	**3.38**	**5.68**

Influence of the hyperparameters C and λ

As shown in the previous experiments, the models that do not use information on the derivative (*O-* and *OP-models*) always lead to poor performance. Thus only the results for the *D-*, *DP-*, *OD-models* and *ODP-models* are reported here.

The effect of the hyperparameter λ (or λ_1 and λ_2) depends on the value of C as the ratio λ/C defines the weight of the prior knowledge with respect to the weight of the training data. The first set of experiments considers $\lambda_1 = \lambda_2 = \lambda$. The balance between λ_1 and λ_2 is studied at the end of this section. In the following, the Figures are moved to the end of the chapter for the sake of clarity.

Figures 4.12 and 4.13 show the variations of the *RMSE test* for the models using respectively RBF and polynomial kernels, trained on $N = 15$ data points with $1 \leq \lambda \leq 2C$ and $1 \leq C \leq 2000$. In this setting, the *RMSE test* is not very sensitive to λ, except for a slight increase observed for large values $\lambda > C$. The data are in sufficient number to cover the whole input space and the approximative nature of the knowledge may decrease the performance if more weight is given to the prior knowledge, i.e. if $\lambda > C$.

For $N = 6$ experimental data points, the test error is also not very sensitive to the tuning of λ, as shown in Figure 4.14 for RBF kernels and Figure 4.15 for polynomial kernels. However, the *D-* and *OD-models* with RBF kernels are improved for large values of $\lambda > C$. These models have not enough support vectors, are thus more dependent on the prior knowledge and require a ratio $\lambda/C > 1$. However, even for large values of λ, they still lead to larger test errors than the models with additional support vectors. The models with polynomial kernels are less sensitive to this issue.

When training the models with only $N = 3$ experimental data points, the method becomes more sensitive to λ and the tuning of this hyperparameter may become critical. As shown in Figure 4.16, a better test error than the one reported in Table 4.4 for $\lambda/C = 1$ could be obtained by the *ODP-model* with RBF kernels for values of λ around $0.1C$. With polynomial kernels, the *DP-model* actually leads to divergent outputs for many values of the couple C, λ. The other models using polynomial kernels are also more sensitive to λ.

These experiments also show that the tuning of C is not critical as the models are mostly insensitive to its value, except for $N = 3$. In general, the polynomial models appear less sensitive to the tuning of C than the RBF models.

However, the models with RBF kernels are more stable with respect to the tuning of λ. Moreover, for different training set sizes N, the *ODP-model* always leads to the best performance, whereas this is not the case with the polynomial models. Thus, for practical reasons, it is preferable to use the RBF kernel, though using a polynomial kernel may lead to a better test error for a particular value of the hyperparameter.

Balance between λ_1 and λ_2. Studying the effect of the balance between the prior knowledge on the ouptut (weighted by λ_1) and the prior knowledge on the derivative (weighted by λ_2) shows that the *ODP-model* is mostly influenced by the information on the derivative for both the RBF and the polynomial kernels. Indeed, for a fixed value of λ_2, the model is rather insensitive to λ_1 as long as λ_1 is not too close to zero. Besides, tests with $N = 15$, 6 and 3 have shown that an almost minimal test error can always be obtained with $\lambda_2 = \lambda_1$. Thus in practice, when mixing two types of prior knowledge without further information on the optimal balance between the two, it is advised to choose $\lambda_2 = \lambda_1$, as this reduces the number of hyperparameters to tune.

Tuning λ by cross validation

The trade-off parameter λ weighting the prior knowledge with respect to the experimental data can be tuned by Leave-One-Out (LOO) cross validation. The LOO procedure allows to compute an estimate of the generalization error, here the generalization root mean square error (RMSE), as follows. The model is trained on $N - 1$ samples from the training set, leaving one sample aside for validation, on which the performance of the model is evaluated. This procedure is repeated N times providing an average performance of the training algorithm for a particular value of the hyperparameter.

This procedure is relevant only for a sufficiently large number of training samples. Thus, only one experiment, for $N = 15$, is reproduced here with, for the training of the *O-*, *OP-*, *D-* and *DP-models*, the optimal value of λ determined by the LOO procedure in the set of 10 values $\{1, 10, 50, 100, 200, 300, 400, 500, 600, 800, 1000, 1500, 2000\}$. These values are to be related to the value of $C = 1000$, which represents the weight of the experimental data in the training. For the *OD-* and *ODP-models*, the relevance of both types of prior knowledge is equivalently balanced and $\lambda_1 = \lambda_2 = \lambda$, where λ is also tuned by the LOO procedure.

Table 4.6: Errors on the residual gas mass fraction obtained by the models using RBF kernels for a training set size of $N = 15$ and λ tuned by cross validation.

N	Model	λ	*RMSE test*	*RMSE total*	*MAE total*
15	1 (experimental model)		1.69	1.54	3.77
	2 (prior model)		5.02	4.93	9.74
	3 (mixed model)		5.07	4.22	7.80
	4 (O-model)	1	2.29	1.49	4.05
	5 (OP-model)	2000	3.70	2.41	5.78
	6 (D-model)	1500	2.93	2.98	8.88
	7 (DP-model)	500	**1.12**	**0.73**	**1.88**
	8 (OD-model)	1500	2.93	2.98	8.88
	9 (ODP-model)	500	**1.12**	**0.73**	**1.88**

Table 4.7: Errors on the residual gas mass fraction obtained by the models using polynomial kernels ($d = 4$) for a training set size of $N = 15$ and λ tuned by cross validation.

N	Model	λ	*RMSE test*	*RMSE total*	*MAE total*
15	1 (experimental model)		2.06	2.17	4.89
	2 (prior model)		4.92	4.89	9.77
	3 (mixed model)		4.97	4.55	9.73
	4 (O-model)	1	3.04	1.98	5.38
	5 (OP-model)	1	3.26	2.12	5.91
	6 (D-model)	400	2.22	1.98	6.22
	7 (DP-model)	10	**0.83**	**0.54**	1.54
	8 (OD-model)	300	2.20	1.83	5.81
	9 (ODP-model)	200	0.84	**0.54**	**1.50**

Tables 4.6 and 4.7 show, respectively for the RBF and polynomial kernels, the optimal value of λ and the corresponding results. The test errors obtained by this tuning procedure are very close to the ones obtained with $\lambda = 1000$ and reported in Tables 4.4 and 4.5. Moreover, an improvement in terms of the *RMSE total* and the *MAE total* can be observed for the two best models using RBF kernels (*DP*- and *ODP-models*). Thus, when no information is available on the optimal balance between the prior knowledge and the data, and when the size of the training set permits it, a cross validation procedure can be used to tune the hyperparameters.

Conclusion

Various methods for the inclusion of knowledge in the form of simulation data have been tested on the application. In this context, real data are available only in a limited number due to the cost of experimental measurements, but additional data can be obtained thanks to a complex physical simulator. The output of the simulator being biased but providing rather good information on the overall shape of the model, prior information on the derivatives, provided by a prior model trained on the simulation data, is the most relevant. Models enhanced by this knowledge thus allow to obtain the best performance.

The additional hyperparameters of the method weight the prior knowledge with respect to the data and can thus be chosen in accordance with the confidence in the prior information. Moreover, the sensitivity of the method with respect to the tuning of these hyperparameters has been experimentally shown, but only on a particular example, to be very low as long as the data are not too few. Besides, the experiments have also shown the importance of adding potential support vectors in the model when using a local kernel, such as the Gaussian RBF kernel, with few training samples.

4.6 Conclusion

This chapter proposed a method for the incorporation of prior knowledge in the LP-SVR learning by adding constraints to the optimization problem and gave an overview of the possibilities offered by this method. Equality and inequality constraints were studied with the corresponding applications illustrated by examples. Many types of prior knowledge can be taken into account by the proposed method such as particular points with known values, prior knowledge on any derivative either provided by a prior model or available only at some points, bounds on the function or a derivative... This extends and regroups the previous works of [184] and [158], which considered particular forms of prior knowledge. Moreover, a new method for the simultaneous approximation of multiple outputs linked by some prior knowledge has been proposed. This method uses the general framework for the incorporation of prior knowledge by the addition of constraints and thus allows consideration of different types of prior knowledge on single outputs while training on multiple outputs. The methods were applied to nonlinear system identification problems and promising results were obtained on real-life data for the estimation of in-cylinder residual gas fraction in spark ignition engines. On this application, the addition of virtual samples with derivative values efficiently improved the model.

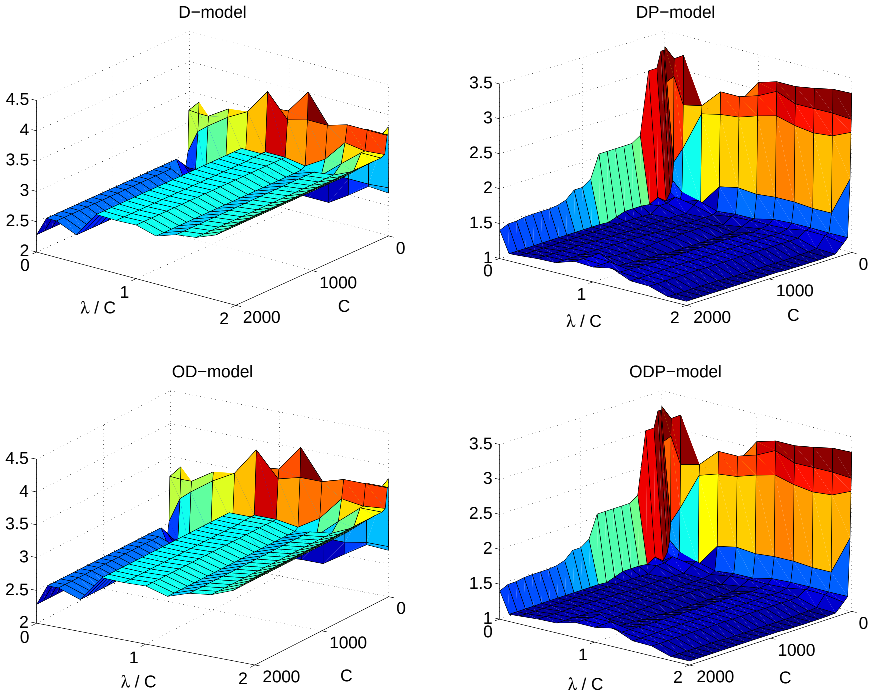


Figure 4.12: *RMSE test* versus C and the ratio C/λ for the *D-*, *DP-*, *OP-* and *ODP-models* trained on $N = 15$ data points with a RBF kernel.

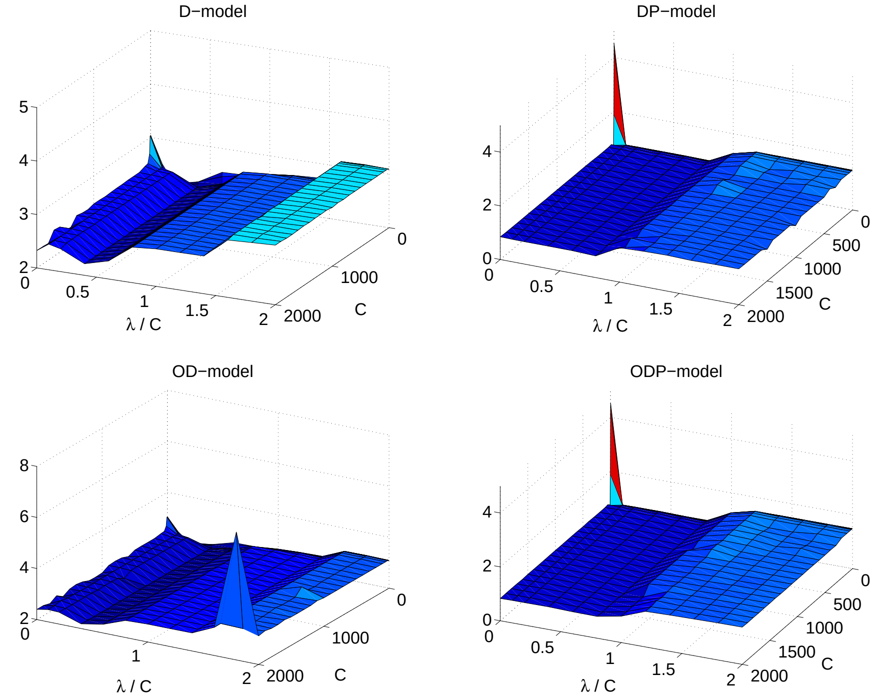


Figure 4.13: *RMSE test* versus C and the ratio C/λ for the *D-*, *DP-*, *OP-* and *ODP-models* trained on $N = 15$ data points with a polynomial kernel ($d = 4$).

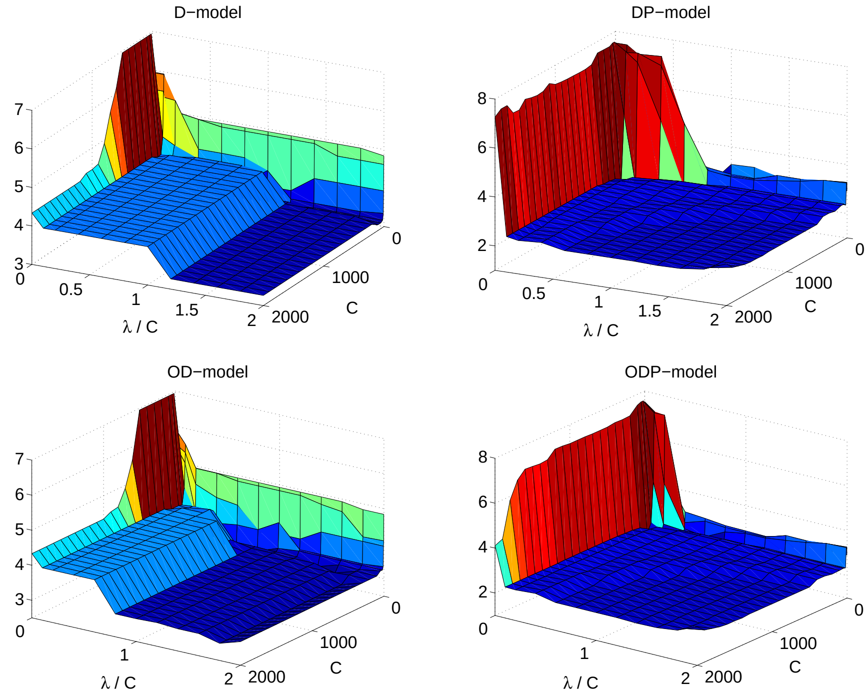


Figure 4.14: *RMSE test* versus C and the ratio C/λ for the *D*-, *DP*-, *OP*- and *ODP-models* trained on $N = 6$ data points with a RBF kernel.

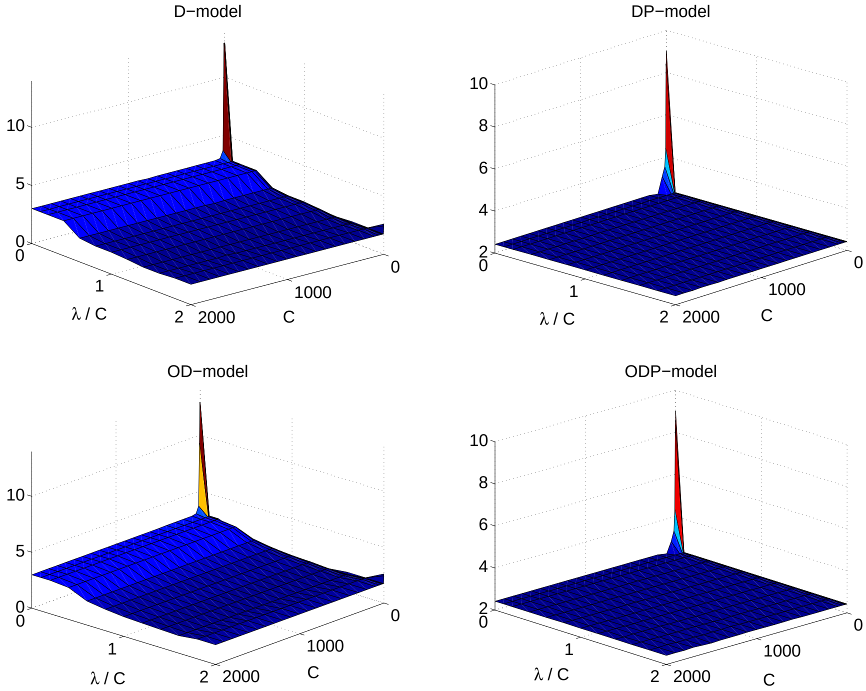


Figure 4.15: *RMSE test* versus C and the ratio C/λ for the *D*-, *DP*-, *OP*- and *ODP-models* trained on $N = 6$ data points with a polynomial kernel ($d = 4$).

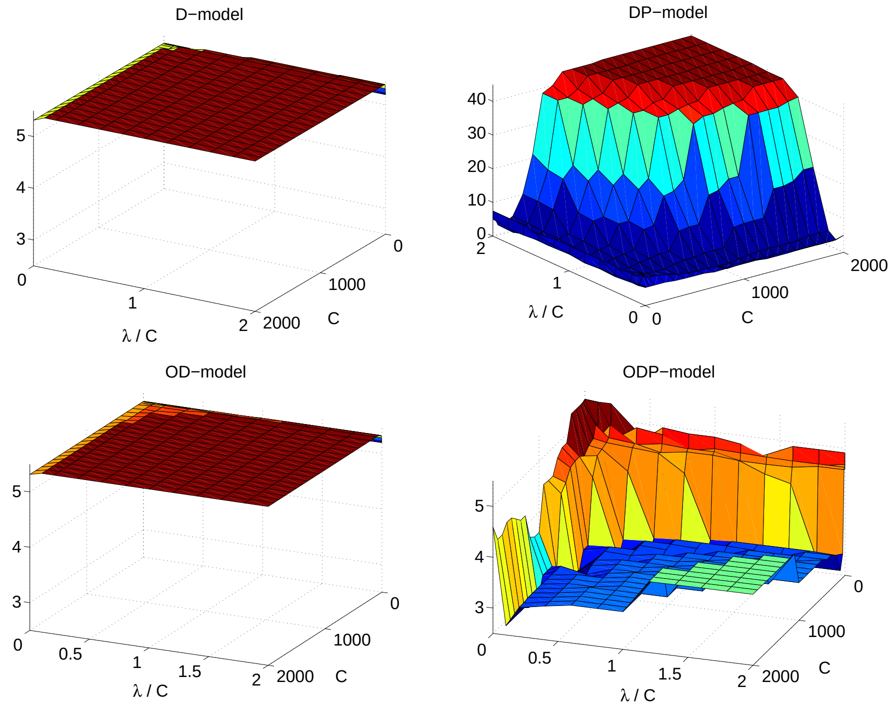


Figure 4.16: *RMSE test* versus C and the ratio C/λ for the *D*-, *DP*-, *OP*- and *ODP-models* trained on $N = 3$ data points with a RBF kernel.

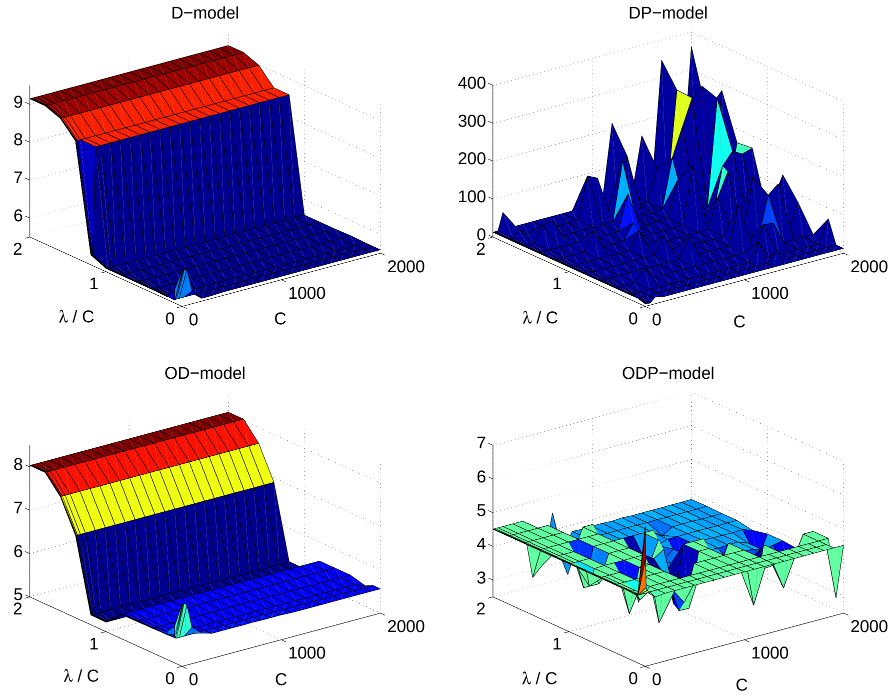


Figure 4.17: *RMSE test* versus C and the ratio C/λ for the *D*-, *DP*-, *OP*- and *ODP-models* trained on $N = 3$ data points with a polynomial kernel ($d = 4$).

Chapter 5

Hybrid system identification

ABSTRACT. *Hybrid system identification is composed of two subproblems: estimate the discrete state or mode for each data point, and estimate the submodel governing the dynamics of the continuous state for each mode. This problem intrinsically involves simultaneous classification and regression. In this chapter, the problem is reformulated as a smooth optimization program by using the ε-insensitive loss function developed for Support Vector Regression (SVR). The resulting algorithm is able to compute both the discrete state and the parameter vectors in a single step, independently of the discrete state sequence that generated the data. In addition to previous works, identification of systems switching between unknown nonlinear dynamics and piecewise systems with nonlinear boundaries between the modes are considered via kernel methods. Other proposed extensions of the method include the estimation of the system order, possibly switching with the system, and the automatic tuning of the main hyperparameter in accordance with the noise level, also possibly switching.*

HYBRID SYSTEMS are more and more popular in the automatic control community. Many models have been proposed to describe them: mixed logical dynamical (MLD) systems, linear complementarity (LC) systems, extended linear complementarity (ELC) systems, piecewise affine (PWA) systems, and max-min-plus-scaling (MMPS) systems. Nonetheless they can all be converted from one to another [116]. In this chapter, two classes of hybrid systems are considered: piecewise systems, for which the discrete state is a function of the continuous state, and arbitrarily switched systems, for which the discrete is independent of the continuous state. Accordingly, these will be identified with two types of models, either piecewise or switched. Hybrid system identification can be very difficult depending on the amount of information available on the system and especially on its discrete part. As seen in section 1.4.3, if the active mode is known for every data point, the problem simply amounts to solving multiple standard identification problems. Therefore the present chapter only studies the case where input-output data are available but the discrete state is unknown.

In this context, hybrid system identification is a natural application and combination of the two problems studied in the previous chapters: *classification* of data into modes and *modeling* of the dynamics of these modes. Note that piecewise models can also be used to approximate general nonlinear systems. This may simplify considerably the control law derivation or system analysis. However, the identification problem becomes much more difficult as it intrinsically involves classification and regression of multiple submodels.

The main contribution of the chapter relies on the reformulation of the hybrid system identification problem as a smooth optimization program (Sect. 5.3) and the introduction of nonlinear kernel models in this framework. This allows us to propose solutions to several problems that have not yet been extensively studied and solved in the literature: identification of hybrid systems switching between unknown nonlinear dynamics (Sect. 5.3.2), identification of hybrid systems with a switching noise level (Sect. 5.3.1), and identification of nonlinearly piecewise systems with nonlinear boundaries between the modes in the regression space (Sect. 5.5.2). An extension to automatically tuning the algorithm according to the noise level of each mode is also provided (Sect. 5.4.3).

Table 5.1: Nomenclature of the hybrid models in ARX form.

ARX model	abbreviation	submodels f_j	discrete state λ_i	domains S_j
PieceWise	PWARX	affine	function of $\boldsymbol{x}_i$	polyhedral
PieceWise Nonlinear	PWNARX	smooth	function of $\boldsymbol{x}_i$	polyhedral
Nonlinearly PieceWise	NPWARX	affine	function of $\boldsymbol{x}_i$	arbitrary
Nonlinearly PieceWise Nonlinear	NPWNARX	smooth	function of $\boldsymbol{x}_i$	arbitrary
Switched	SARX	affine	arbitrary	
Switched Nonlinear	SNARX	nonlinear	arbitrary	

Original presentations of the method can be found in [151] and [152]. In this chapter, the description of the method is unified and extended to systems with varying unknown orders (Sect. 5.4.1). Two frameworks for the interpretation of the algorithms are given in Sect. 5.6, where the proposed method is also interpreted as a bridge between two of the main approaches for hybrid system identification. The chapter ends with a number of numerical examples in section 5.7.

5.1 Hybrid dynamical systems

5.1.1 ARX models of hybrid systems

A hybrid system is usually described by both a continuous state and a discrete state, where the vector field defining the evolution of the continuous state depends on the discrete state. In this framework, a system can be seen as switching between different subsystems, which are usually modeled by linear AutoRegressive with eXogenous inputs (ARX) models in the discrete-time case.

The predicted output $\hat{y}_i$ of a hybrid model in ARX form is given as a function of the p-dimensional regression vector[1] $\boldsymbol{x}_i = [y_{i-1},\ \dots,\ y_{i-n_a},\ u_{i-n_k},\ \dots,\ u_{i-n_k-n_c+1}, 1]^T \in \mathbb{R}^p$, containing the lagged inputs u_{i-k} and outputs y_{i-k}, and the discrete state $\lambda_i \in \{1, 2, \dots, n\}$. Considering n submodels f_j, the hybrid model is written as

$$\hat{y}_i = f(\boldsymbol{x}_i) = f_{\lambda_i}(\boldsymbol{x}_i). \tag{5.1}$$

To model linear hybrid systems, affine submodels

$$f_j(\boldsymbol{x}) = \boldsymbol{w}_j^T \boldsymbol{x}, \tag{5.2}$$

where $\boldsymbol{w}_j$ is the parameter vector of the jth submodel, are used. Hybrid models can be classified with respect to the nature of the submodels f_j and of the evolution of the discrete state λ_i. Table 5.1 defines the nomenclature that will be used in this chapter. SARX and SNARX models assume that the system is arbitrarily switched. On the other hand, PWARX models consider a dependency between the discrete state and the continuous state. They can thus be defined by PieceWise Affine (PWA) maps of the type

$$f(\boldsymbol{x}) = f_j(\boldsymbol{x}),\ \text{if } \boldsymbol{x} \in S_j = \{\boldsymbol{x} \in \mathcal{X} \mid \boldsymbol{H}_j \boldsymbol{x} \le 0\},\ j = 1, \dots, n, \tag{5.3}$$

where f_j are affine functions as in (5.2) and the matrices $\boldsymbol{H}_j$ define a set of hyperplanes partitioning the regression space $\mathcal{X}$ into polyhedral domains S_j. Similarly, PWNARX models can be defined by PieceWise Smooth (PWS) maps, where f_j are smooth nonlinear functions instead of affine functions. In section 5.5.2, extensions of the PWARX and PWNARX models to nonlinearly piecewise models (NPWARX and NPWNARX), where the domains S_j are no more constrained to be polyhedral, will be discussed.

5.1.2 Identification problem

In this thesis, the hybrid system identification problem is considered as follows.

[1] The last component of $\boldsymbol{x}$ is set to 1 in order to account for affine models. This is optional and should be removed if the subsystems are known to be linear.

From N input-output samples $(\boldsymbol{x}_i, y_i)$, find:

1. *the switching sequence $\lambda_i,\ i = 1, \dots, N$,*
2. *the submodels $f_j,\ j = 1, \dots, n$,*
3. *for a piecewise model, the domains S_j or the switching boundaries in the regression space.*

In section 5.4.1, the additional problems of estimating the number of modes n and the orders of the subsystems will be considered.

5.2 Related work

The following survey, though not exhaustive, does not only focus on hybrid dynamical systems, for which good overviews are given in [219] and [204]. The main approaches developed in the last few years for hybrid systems are first recalled with pointers to their most recent updates. Then an overview of the papers discussing the identification of other models related to hybrid systems is given. Note that all the methods below consider linear hybrid system identification or piecewise affine function approximation. On the contrary, the proposed method will be able to deal with arbitrary and unknown nonlinearities in the submodels.

5.2.1 Recent approaches

Recently, six main approaches have been devised for hybrid system identification: the clustering-based approach [81], the mixed integer programming based approach [220], the Bayesian approach [137], the bounded-error approach [15], the adapted weights approach [216] and the algebraic approach [287, 176]. The first five focus on the problem of PieceWise Affine (PWA) system identification, where the discrete state depends on the continuous state. However, both the Bayesian and bounded-error approaches can also be used to identify arbitrarily switched systems with SARX models. The algebraic approach [287] focuses on this latter problem.

These recent approaches also benefit from extensions. The Bayesian approach has been extended to piecewise output-error models [136]. The clustering step at the core of the method described in [81] has been improved in [195]. For the algebraic approach, a recursive algorithm has been proposed [284, 283] and the method has recently been extended to deal with Multiple Input Multiple Output (MIMO) systems [13]. Other methods for the identification of switched MIMO systems exist [280, 12] but require a minimum dwell time in each mode.

Since the proposed method relies mainly on the algebraic and bounded-error approaches, only these two are detailed in the following.

Algebraic approach

Consider a discrete-time linear hybrid system of output y_i given by the set of subsystems in ARX form

$$y_i = \boldsymbol{\theta}_{\lambda_i}^T \boldsymbol{x}_i, \tag{5.4}$$

where the discrete state $\lambda_i \in \{1, 2, \dots, n\}$ determines which mode and thus which parameter vector $\boldsymbol{\theta}_j$, $j = 1, \dots, n$, is active to compute the output y_i.

The algebraic approach proposed in [287] allows to simultaneously estimate the discrete and the continuous part of a SARX system. For noiseless data, it basically amounts to consider that the vector $\boldsymbol{z}_i = [\boldsymbol{x}_i^T \ \ -y_i]^T$ lies at least on one of the hyperplanes defined by $\boldsymbol{b}_j^T \boldsymbol{z}_i = 0$, where $\boldsymbol{b}_j = [\boldsymbol{\theta}_j^T \ \ 1]^T$. Therefore, for all data points $\boldsymbol{z}_i$, the hybrid "decoupling" polynomial constraints can be stated as

$$\prod_{j=1}^{n} \boldsymbol{b}_j^T \boldsymbol{z}_i = 0, \ i = 1, \dots, N. \tag{5.5}$$

Thanks to these constraints, the problem becomes independent of the discrete state sequence that generated the data and can be reformulated as a linear system of equations.

Define the homogeneous polynomial of degree n as

$$p_n(\boldsymbol{z}) = \prod_{j=1}^{n} \boldsymbol{b}_j^T \boldsymbol{z}. \tag{5.6}$$

This polynomial can be written in terms of new variables $\boldsymbol{h}$ built from products of the original parameters $\boldsymbol{\theta}$:

$$p_n(\boldsymbol{z}) = \boldsymbol{h}^T \boldsymbol{\nu}_n(\boldsymbol{z}), \tag{5.7}$$

where $\boldsymbol{\nu}_n(\boldsymbol{z})$ is the Veronese map[2] of degree n. Constraints (5.5) can be written in terms of the polynomials $p_n(\boldsymbol{z}_i)$ and the problem finally amounts to solving a linear system of equations in the variables $\boldsymbol{h}$ given by

$$\boldsymbol{L}_n \boldsymbol{h} = \begin{bmatrix} \boldsymbol{\nu}_n(\boldsymbol{z}_1)^T \\ \vdots \\ \boldsymbol{\nu}_n(\boldsymbol{z}_N)^T \end{bmatrix} \boldsymbol{h} = \boldsymbol{0}. \tag{5.8}$$

A condition on the ranks of the matrices $\boldsymbol{L}_m$, built from Veronese maps of degrees m, can be derived to determine the number n of modes [287]. In the case of noisy data, the system (5.8) can still be solved in a least squares sense. However, the condition on the rank cannot be used to determine n without requiring some heuristics.

Recovering of the original parameters $\boldsymbol{b}_j$ as depicted in [287] requires to solve for the n roots α_j of the single-variable polynomial $q_n(\alpha) = p_n(\boldsymbol{z}_0 + \alpha\boldsymbol{v})$ for specific choices of $\boldsymbol{z}_0$ and $\boldsymbol{v}$ and then to evaluate the derivatives $\partial p_n(\boldsymbol{z})/\partial \boldsymbol{z}$ at the points $\boldsymbol{z}_j = \boldsymbol{z}_0 + \alpha_j \boldsymbol{v}$. For noisy data, $\boldsymbol{z}_j$ can be chosen as points from the training set that minimize some distance to the hyperplanes. This allows to bypass the computation of the roots of $q_n(\alpha)$ [176].

The algebraic approach is a convenient method for the identification of SARX systems that allows the problem to become independent of the discrete state sequence that generated the data and thus bypasses the issue of classifying the data points into modes. However it does not take the noise into account and has been shown experimentally in [135] to be rather sensitive to noise compared to the clustering-based or bounded-error methods. However, it can still be used to initialize an iterative method as advised in [287].

Bounded-error approach

The bounded-error approach developed in [15] aims at finding the minimal number n of submodels that allows the error on all the training samples $(\boldsymbol{x}_i, y_i)$ to be bounded by

$$|y_i - f(\boldsymbol{x}_i)| = |e_i| \leq \delta,\ i = 1, \dots, N. \tag{5.9}$$

In this approach, the problem of estimating both the number of submodels and their parameter vectors is solved by a Minimal Partition Feasible Sets (MIN PFS) procedure including the iteration of a Maximum Feasible Set (MAX FS) algorithm. The basic idea is to look for the maximum number of inequalities (and thus points) in (5.9) that can be satisfied by a single linear submodel. Then, these inequalities are removed and a new MAX FS iteration operates on the set of remaining points, while considering a new submodel (or new parameter vector). The algorithm terminates in a finite number of steps when all the inequalities are satisfied. A number of refinement steps are then proposed: merging submodels with parameter vectors close to each other, reassigning undecidable points based on the partition in the regressor space (assuming a PWARX model), discarding submodels with too few samples assigned to the corresponding mode, and detection of outliers based on the infeasibility of the corresponding bounded-error constraints after reducing the number of submodels by the previous steps.

In [135] this method is considered to be rather good with respect to noise. Moreover, the hyperparameter δ allows to tune the trade-off between model complexity (the number of modes) and accuracy. However, if prior knowledge on n is available or a particular structure prescribed,

[2] For instance, for $\boldsymbol{z}_i = [y_{i-1} \ -y_i]^T$ and $n = 2$, $p_n(\boldsymbol{z}_i) = \boldsymbol{b}_1^T\boldsymbol{z}_i \times \boldsymbol{b}_2^T\boldsymbol{z}_i = (\theta_1 y_{i-1} - y_i)(\theta_2 y_{i-1} - y_i) = \theta_1\theta_2 y_{i-1}^2 + y_i^2 - (\theta_1+\theta_2)y_{i-1}y_i$. In this case, $\boldsymbol{h} = [\theta_1\theta_2,\ 1,\ -(\theta_1+\theta_2)]^T$ and the Veronese map of degree 2 is $\boldsymbol{\nu}_n(\boldsymbol{z}_i) = [y_{i-1}^2,\ y_i^2,\ y_{i-1}y_i]^T$.

the error threshold δ together with the other hyperparameters α (threshold for submodel merging) and c (number of nearest neighbors for undecidable data) may be difficult to tune.

Remark 13. *In a refinement step, the points for which $|y_i - f(\boldsymbol{x}_i)| > \delta$ are discarded as outliers. Though this is reasonable for true outliers, some data points may also violate the bounded-error constraint because of a slight error in the parameter estimates or a level of noise slightly higher than the bound δ. In comparison, the proposed method of section 5.3 will allow to take such points into account, while remaining robust to outliers thanks to the ε-insensitive loss function.*

Remark 14 (Extending the bounded-error method to deal with multiple thresholds δ)**.** *The proposed method will allow for the case where different bounds δ_j are used for each mode j (see Sect. 5.3.1). The bounded-error method [15] could also possibly use different bounds δ_j. In this case, the MIN PFS algorithm should be modified to update δ at each iteration based on a non-decreasing predefined sequence $\delta_1 \leq \delta_2 \leq \cdots \leq \delta_n$.*

5.2.2 Other approaches for static functions

Hybrid system identification, as considered in this thesis and in most of the recent approaches of Sect. 5.2.1, amounts to a switched or piecewise regression problem.

In statistics, the problem of piecewise linear (or affine) function approximation has been considered since the 50's: in [215] as linear systems obeying two separate regimes, in [120] as segmented curves fitting, and then in [187] as piecewise regression. Other authors studied this problem until now [96, 167, 205]. The particular case of fitting to data a convex piecewise function, expressed as the maximum of affine functions, is studied in [178]. In data mining, another model of switching regression, also including piecewise regression, has been proposed in [166] as mixture of "regression-classes". Relying on highly robust estimators, the method identifies regression-classes one by one in an iterative scheme.

In economy, switching regression is considered in [262, 263] and has been studied in the Bayesian framework in [173]. Besides, many models in economic theory give rise to continuous piecewise affine functions. Such functions can be approximated by the algorithm proposed in [101], of which a proof of convergence is available [102]. For this problem, estimators involving the combination of min. and max. operations on affine estimators have recently been proposed in [4] as Riesz estimators.

In addition, other models strongly related to piecewise affine maps have been proposed for nonlinear function approximation, among which Chua's canonical form [52, 51, 134] and hinging hyperplanes [32, 214] can be cited.

5.2.3 Other approaches for dynamical systems

Multiple models

Recursive identification with multiple models in parallel has been proposed in [6] for systems with rapidly changing parameters. This method is similar to using a bank of adaptive filters [309]. In [242], the name "composite models" is used to refer to piecewise linear models built with a similar approach.

Nonlinear systems have been approximated by local models associated to validity regions in the framework of fuzzy systems [114, 307] or operating regime modeling [132, 130, 133]. However, in this context, smooth transitions between the local models (or modes) are considered. A link between this approach and multiple models has been made in [92] and [190].

Threshold models

In the context of time-series forecasting, threshold models were proposed at the end of the 70's, see e.g. [265], and later formulated as hybrid systems in the book [264]. A threshold autoregressive (TAR) model is composed of several submodels. Whether a submodel f_j is active or not is determined by indicator functions $I_j(q_k) = 1$, if $q_k > r_j$, 0 otherwise, for ordered $r_j < r_{j+1}$. In these TAR models, the independent variable q_k is supposed to be known. If, instead of an independent variable, a delayed output y_{k-q} is chosen, then the model becomes a *Self Excited* TAR (SETAR).

In *Self Excited Multivariate* TAR (SEMTAR) [188], the thresholds are hyperplanes and a vector of delayed outputs, with possibly another choice of regressors than for $\boldsymbol{x}$, is used instead of y_{k-q}. In *Smooth Transition Autoregressive* (STAR) models [38, 99], the indicator functions I_j are replaced by smooth functions. It must be noticed that in all these TAR models, more than one submodel can be active at a time.

Probabilistic Models

Hyrid dynamical systems are also studied in the framework of graphical models as Hidden Markov Models (HMM) [270, 67, 93], segmental HMM for speech recognition [224, 202], or Jump Markov Linear Systems (JMLS) (see [53] and references therein). In this framework, transition probabilities are considered for the jumps from one mode to another.

In addition, regime-switching models introduced in [111] for econometric time-series analysis, such as interest rates [100, 8], use Markov chains to model the switching process.

5.3 SVR-based hybrid system identification

The proposed method uses tools and ideas from the fields of machine learning and optimization, more particularly Support Vector Machines (SVM) and nonlinear programming. For piecewise affine system identification, SVM classifiers are already used to estimate the switching boundaries between the modes [81, 15]. As seen in chapter 3, for regression, SVR uses the ε-insensitive loss function (3.4). This loss function, which allows for errors below a predefined threshold, is close in spirit to the bounded-error approach [15]. However, the origin is different. In learning theory, the threshold effect is justified in order to minimize the generalization error of the model, whereas the bounded-error approach was developed to allow the automatic determination of the number of linear submodels required to approximate a nonlinear function with a given accuracy.

The following proposes a bounded-error approach for linear and nonlinear hybrid system identification from noisy data. This method can also be seen as a relaxation of the hybrid decoupling constraint in the algebraic identification framework [287]. Borrowing ideas from the theory of support vector regression, the derivation remains simple and allows the formulation of the problem as a smooth optimization program. The method is able to deal with arbitrarily switched systems by being independent of the discrete state sequence that generated the data. Piecewise systems can also be treated as a special case, in which the additional knowledge on the switching rule (e.g. the modes are linearly separable in the regression space) can be used to refine the model.

This section starts by the presentation of the method for linear hybrid system identification. The proposed algorithm is then extended to deal with nonlinear hybrid systems by introducing kernel functions as in the standard SVM methodology. The section ends with a discussion on the optimization issues involved in the various algorithms.

5.3.1 Linear hybrid system identification

In the following, the model output, $\hat{y}_i = f_{\lambda_i}(\boldsymbol{x}_i)$, is a function of the regression vector $\boldsymbol{x}_i$ and the discrete state $\lambda_i \in \{1, \dots, n\}$, where the n affine submodels are of the form[3]

$$f_j(\boldsymbol{x}) = \boldsymbol{w}_j^T \boldsymbol{x}, \ j = 1, \dots, n, \tag{5.10}$$

with parameter vectors $\boldsymbol{w}_j$. The discrete state λ_i can be either dependent or independent of $\boldsymbol{x}_i$, with respect to the class of systems considered.

The general scheme

Following the SVR approach, the submodels (5.10) are estimated by minimizing the norms of the parameter vectors $\|\boldsymbol{w}_j\|_2$. Minimizing these norms, which are also measures of the model

[3] In the standard SVR framework, the affine model is explicitly written by adding a bias term instead of adding ones in the regression vectors. All the algorithms presented here can be equivalently written with a bias term. The only resulting difference is the regularization of the bias term.

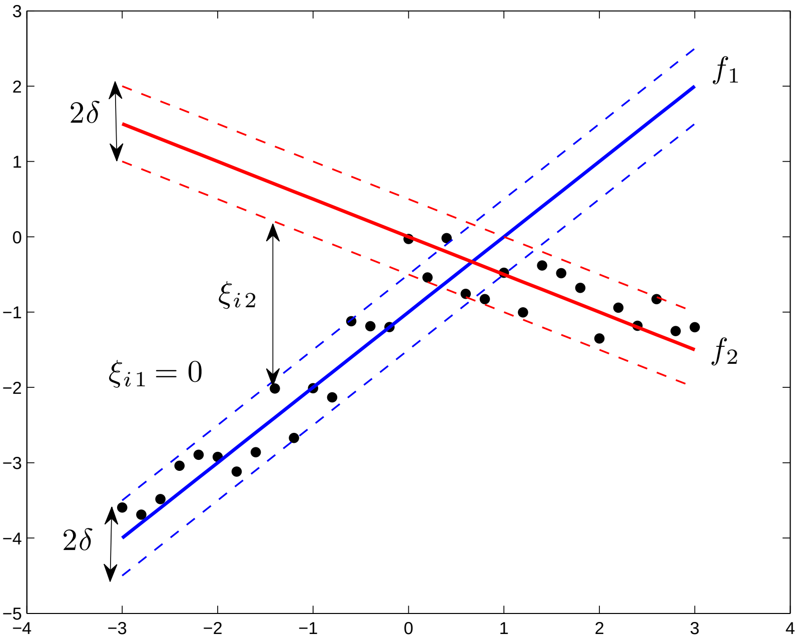


Figure 5.1: Representation of the slack variables ξ_{ij} for linear hybrid system identification.

complexity, is known to yield models with better generalization performance [244]. In this case, the problem of training n models under the bounded-error constraint (5.9) may be written as

$$\min_{\boldsymbol{w}_j} \sum_{j=1}^{n} \|\boldsymbol{w}_j\|_2^2 \tag{5.11}$$

$$-\delta \leq y_i - \boldsymbol{w}_{\lambda_i}^T \boldsymbol{x}_i \leq \delta,\ i = 1, \dots, N, \tag{5.12}$$

where the submodel absolute error $|e_{ij}| = |y_i - \boldsymbol{w}_j^T \boldsymbol{x}_i|$ is constrained to be less than δ only for the submodel f_j corresponding to the unknown discrete state λ_i of the point $\boldsymbol{x}_i$. However, without further information on the discrete state or the regions S_j (5.3) for piecewise systems, the problem is intractable. To circumvent this issue, consider the ε-insensitive loss function $l(e_{ij}) = |e_{ij}|_\varepsilon$, defined in (3.4) for SVR (see Sect. 3.1). This loss function builds a tube of insensitivity, in which the errors are meaningless, and penalizes linearly the errors larger than ε. An approximate solution to problem (5.11) can now be found by solving the following problem, which implements this loss function for $\varepsilon = \delta$ with slack variables $\xi_{ij} \geq l(e_{ij})$, $i = 1, \dots, N$, $j = 1, \dots, n$, as

$$\min_{\boldsymbol{w}_j, \xi_{ij} \geq 0} \sum_{j=1}^{n} \|\boldsymbol{w}_j\|_2^2 \tag{5.13}$$

$$-\xi_{ij} - \delta \leq y_i - \boldsymbol{w}_j^T \boldsymbol{x}_i \leq \delta + \xi_{ij},\ j = 1, \dots, n,\ i = 1, \dots, N \tag{5.14}$$

$$\prod_{j=1}^{n} \xi_{ij} = 0,\ i = 1, \dots, N. \tag{5.15}$$

As shown in Fig. 5.1, the slack variables ξ_{ij} can be seen as the error of the submodel j for the point i above the threshold δ. The last equalities thus stand for the fact that all points must be estimated with accuracy δ by at least one submodel. The global approach to go from (5.11) to (5.13)-(5.15) can be summarized as follows.

1. *Assign all the points, indexed by i, to all the modes, indexed by j.* This amounts to writing the bounded-error constraint (5.12) for $j = 1, \dots, n$ and $i = 1, \dots, N$.

2. *Relax the constraints* by adding slack variables ξ_{ij} as in (5.14).

3. *For every point, force at least one submodel to approximate it correctly.* This amounts to force at least one of the slack variables $\xi_{ij},\ j = 1, \dots, n$ to be zero and is imposed by constraints (5.15).

Nonlinear equalities are not easy to deal with from an optimization point of view. Moreover, imposing hard constraints such as (5.15) is not robust to outliers, as the problem may become infeasible. The equality constraints are thus relaxed and the problem is finally formulated as

$$\begin{aligned} &\min_{\boldsymbol{w}_j, \xi_{ij} \geq 0} \sum_{j=1}^{n} \|\boldsymbol{w}_j\|_2^2 + C \sum_{i=1}^{N} \prod_{j=1}^{n} \xi_{ij} \\ &-\xi_{ij} - \delta \leq y_i - \boldsymbol{w}_j^T \boldsymbol{x}_i \leq \delta + \xi_{ij},\ j = 1, \dots, n,\ i = 1, \dots, N, \end{aligned} \tag{5.16}$$

or, in matrix form,

$$\begin{aligned} &\min_{\boldsymbol{w}_j, \boldsymbol{\xi}_j \geq \mathbf{0}} \sum_{j=1}^{n} \boldsymbol{w}_j^T \boldsymbol{w}_j + C \sum_{i=1}^{N} \prod_{j=1}^{n} \xi_{ij} \\ &-\boldsymbol{\xi}_j - \delta \mathbf{1} \leq \boldsymbol{y} - \boldsymbol{X} \boldsymbol{w}_j \leq \delta \mathbf{1} + \boldsymbol{\xi}_j,\ j = 1, \dots, n, \end{aligned} \tag{5.17}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1\ \boldsymbol{x}_2\ \dots\ \boldsymbol{x}_N]^T$, $\boldsymbol{y} = [y_1\ y_2\ \dots\ y_N]^T$ and $\boldsymbol{\xi}_j = [\xi_{1j}\ \xi_{2j}\ \dots\ \xi_{Nj}]^T$. Solving this problem yields the parameter vectors $\boldsymbol{w}_j$, $j = 1, \dots, n$. Moreover, the discrete state λ_i is readily available for all i from the variables ξ_{ij}. Indeed $\hat{\lambda}_i = j$, when $\xi_{ij} = 0$, i.e. when the submodel f_j approximates $(\boldsymbol{x}_i, y_i)$ with an error less than δ. Undecidable points, for which more than one ξ_{ij} is zero, or points for which the bounded-error constraint is not satisfied, i.e. no ξ_{ij} is zero, can be further discriminated by considering the absolute errors and letting $\hat{\lambda}_i = \arg\min_{j=1,\dots,n} |e_{ij}|$.

Switching noise level

An advantage of the proposed approach is the possibility to deal easily with a different noise level for each mode by using different bounds on the errors. Simply define n ε-insensitive loss functions (3.4) with different thresholds ε set to δ_j for each mode j. This setting is implemented in the following final problem, involving $n(p + N)$ variables,

$$\begin{aligned} &\min_{\boldsymbol{w}_j, \boldsymbol{\xi}_j \geq \mathbf{0}} \sum_{j=1}^{n} \boldsymbol{w}_j^T \boldsymbol{w}_j + C \sum_{i=1}^{N} \prod_{j=1}^{n} \xi_{ij} \\ &-\boldsymbol{\xi}_j - \delta_j \mathbf{1} \leq \boldsymbol{y} - \boldsymbol{X} \boldsymbol{w}_j \leq \delta_j \mathbf{1} + \boldsymbol{\xi}_j,\ j = 1, \dots, n. \end{aligned} \tag{5.18}$$

ℓ_1-norm regularization

Regularization can also be obtained by minimizing the ℓ_1-norms of the parameter vectors $\boldsymbol{w}_j$ instead of $\|\boldsymbol{w}_j\|_2$ in the objective function (5.18). Using a similar trick as for LP-SVR (see Sect. 3.1.2), this may be equivalently implemented by

$$\begin{aligned} &\min_{\boldsymbol{w}_j, \boldsymbol{a}_j \geq \mathbf{0}, \boldsymbol{\xi}_j \geq \mathbf{0}} \sum_{j=1}^{n} \mathbf{1}^T \boldsymbol{a}_j + C \sum_{i=1}^{N} \prod_{j=1}^{n} \xi_{ij} \\ &-\boldsymbol{\xi}_j - \delta_j \mathbf{1} \leq \boldsymbol{y} - \boldsymbol{X} \boldsymbol{w}_j \leq \delta_j \mathbf{1} + \boldsymbol{\xi}_j,\ j = 1, \dots, n, \\ &-\boldsymbol{a}_j \leq \boldsymbol{w}_j \leq \boldsymbol{a}_j,\ j = 1, \dots, n, \end{aligned} \tag{5.19}$$

where vectors $\boldsymbol{a}_j$ of positive variables are introduced to bound $\|\boldsymbol{w}_j\|_1$ by $\mathbf{1}^T \boldsymbol{a}_j$. Problem (5.19) involves $n(2p + N)$ variables. The case when this algorithm is preferable over the standard ℓ_2-norm regularized version (5.18) will be discussed in section 5.4.1.

Table 5.2: Possible choices of regularizer with the corresponding norm and parameter space in which the norm is computed.

Regularizer	parameter space	main goal	program
$\|\boldsymbol{w}_j\|_2^2$	$\mathbb{R}^p$	flatness	(5.18)
$\|\boldsymbol{w}_j\|_1$	$\mathbb{R}^p$	feature selection	(5.19)
$\|\boldsymbol{\alpha}_j\|_1$	$\mathbb{R}^N$	sparsity of SVs	(5.21)
$\|\boldsymbol{\alpha}_j\|_2^2$	$\mathbb{R}^N$	smoothness	(5.22)

5.3.2 Nonlinear hybrid system identification with kernels

Following the LP-SVR approach of Sect. 3.1.2, submodels in kernel expansion form

$$f_j(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_{ij} k_j(\boldsymbol{x}, \boldsymbol{x}_i) + b_j = \boldsymbol{K}_j(\boldsymbol{x}, \boldsymbol{X}^T)\boldsymbol{\alpha}_j + b_j, \tag{5.20}$$

of parameters $\boldsymbol{\alpha}_j = [\alpha_{1j}\ \alpha_{2j}\ \dots\ \alpha_{Nj}]^T$, can be used to estimate nonlinear hybrid models. As indicated by the subscript j, different kernel functions k_j can be associated to the different models f_j, leading to kernel matrices $\boldsymbol{K}_j = \boldsymbol{K}_j(\boldsymbol{X}^T, \boldsymbol{X}^T)$. Thus it is possible to take prior information into account such as the number of modes governed by linear dynamics or knowledge on the type of a particular nonlinearity. Using the ℓ_1-norm regularization as in LP-SVR and a similar derivation as for linear submodels, the problem of training n nonlinear submodels may be written as

$$\begin{aligned} \min_{\boldsymbol{\alpha}_j, b_j, \boldsymbol{a}_j, \boldsymbol{\xi}_j \geq \boldsymbol{0}} \quad & \sum_{j=1}^{n} \boldsymbol{1}^T \boldsymbol{a}_j + C \sum_{i=1}^{N} \prod_{j=1}^{n} \xi_{ij} \\ & -\boldsymbol{\xi}_j - \delta_j \boldsymbol{1} \leq \boldsymbol{y} - \boldsymbol{K}_j \boldsymbol{\alpha}_j - b_j \boldsymbol{1} \leq \delta_j \boldsymbol{1} + \boldsymbol{\xi}_j,\ j = 1, \dots, n, \\ & -\boldsymbol{a}_j \leq \boldsymbol{\alpha}_j \leq \boldsymbol{a}_j,\ j = 1, \dots, n, \end{aligned} \tag{5.21}$$

which involves $n(3N+1)$ variables.

Another possible formulation with $n \times N$ less variables and constraints involves the minimization of the squares of the parameters α_{ij}, which also penalizes non-smooth functions f_j. This leads to

$$\begin{aligned} \min_{\boldsymbol{\alpha}_j, b_j, \boldsymbol{\xi}_j \geq \boldsymbol{0}} \quad & \sum_{j=1}^{n} \boldsymbol{\alpha}_j^T \boldsymbol{\alpha}_j + C \sum_{i=1}^{N} \prod_{j=1}^{n} \xi_{ij} \\ & -\boldsymbol{\xi}_j - \delta_j \boldsymbol{1} \leq \boldsymbol{y} - \boldsymbol{K}_j \boldsymbol{\alpha}_j - b_j \boldsymbol{1} \leq \delta_j \boldsymbol{1} + \boldsymbol{\xi}_j,\ j = 1, \dots, n, \end{aligned} \tag{5.22}$$

with $n(2N+1)$ variables. The solution of this problem is not as sparse as the one of (5.21) but can usually be computed in less time.

The different algorithms use different regularizers, either the ℓ_1 or ℓ_2 norm, computed in different parameter spaces for different purposes. Table 5.2 summarizes the four alternatives with their respective features.

Remark 15. *In the case of a linear kernel $k_j(\boldsymbol{x}, \boldsymbol{x}_i) = \boldsymbol{x}^T \boldsymbol{x}_i$, the corresponding linear submodel, $f_j(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{X}^T \boldsymbol{\alpha}_j + b_j$, can be simply rewritten as $f_j(\boldsymbol{x}) = \boldsymbol{w}_j^T \boldsymbol{x} + b_j$, with parameters explicitly given by $\boldsymbol{w}_j = \boldsymbol{X}^T \boldsymbol{\alpha}_j$.*

A note on the bias term b

For nonlinear models, the role of the bias term b is not the intercept as for affine models. It is used to extend the class of admissible functions f, explicitly including constant functions, and cannot be simply replaced by setting the last component of $\boldsymbol{x}$ to 1. The effect of adding a constant to the regression vector depends on the kernel function. This component, for RBF kernels based on the norm $\|\boldsymbol{x} - \boldsymbol{x}_i\|$, is simply ignored. Using a homogeneous polynomial kernel $k(\boldsymbol{x}, \boldsymbol{x}_i) = (\boldsymbol{x}^T \boldsymbol{x}_i)^d$ is then equivalent to using an in-homogeneous polynomial kernel $k(\boldsymbol{x}, \boldsymbol{x}_i) = (\boldsymbol{x}^T \boldsymbol{x}_i + 1)^d$ without the ones in the regression vectors $\boldsymbol{x}$. For the linear kernel, however, the constant term in $\boldsymbol{x}$ can replace the explicit bias b, which is in this case given by $b = \sum_{i=1}^N \alpha_i$.

5.3.3 MIMO hybrid system identification

The proposed method can be modified to take multiple outputs into account. In this context, each measured output y^m, $m = 1, \dots, M$, is estimated by a hybrid model f^m, with submodel parameter vectors $\boldsymbol{w}_j^m$, $j = 1, \dots, m$. After modification, the algorithm (5.18) becomes

$$\min_{\boldsymbol{w}_j^m, \boldsymbol{\xi}_j \geq \boldsymbol{0}} \sum_{m=1}^M \sum_{j=1}^n \|\boldsymbol{w}_j^m\|_2^2 + C \sum_{i=1}^N \prod_{j=1}^n \xi_{ij} \tag{5.23}$$

$$-\boldsymbol{\xi}_j - \delta_j \mathbf{1} \leq \boldsymbol{y}^1 - \boldsymbol{X}\boldsymbol{w}_j^1 \leq \delta_j \mathbf{1} + \boldsymbol{\xi}_j, \ j = 1, \dots, n,$$

$$\vdots$$

$$-\boldsymbol{\xi}_j - \delta_j \mathbf{1} \leq \boldsymbol{y}^M - \boldsymbol{X}\boldsymbol{w}_j^M \leq \delta_j \mathbf{1} + \boldsymbol{\xi}_j, \ j = 1. \dots, n.$$

Note that a unique slack vector $\boldsymbol{\xi}_j$ is used for all the M outputs to ensure that the M models are in the same mode at the same time. Thus no additional variable is required beside the parameters of the submodels and the number of variables is $n(Mp + N)$. Extending this algorithm to consider different values of δ_j for each output is straightforward.

5.3.4 Optimization issues

Problems (5.18) and (5.19) are linearly constrained nonlinear optimization programs involving respectively $n(p + N)$ and $n(2p + N)$ variables, which may become a difficulty for a large number N of data. However, the proposed method benefits from the good properties of Support Vector Regression and is thus well adapted for the cases where few data are available. Problems minimizing a nonlinear objective function over a convex set, such as (5.18) and (5.19), can be solved by general purpose solvers such as the function *fmincon* found in MATLAB, but with no guarantee that the solution is a global minimum. The non-convexity of the objective functions is mainly due to inherent symmetries of the problems: permutation of two submodels has no effect on the solution. These symmetries are embedded in the non-convex product terms $\prod_{j=1}^n \xi_{ij}$, which have however a global minimum equal to zero (due to $\xi_{ij} \geq 0$). For values of δ_j so that the bounded-error constraints can be satisfied, this optimum is feasible. Local optimization algorithms, such as sequential quadratic programming used by *fmincon*, are thus expected to converge towards a non-unique but global minimum and can be used to solve the problems.

In practice, it has been observed that the optimization rarely leads to a local minimum corresponding to a bad solution of the initial system identification problem. Moreover, such bad solutions are easily detected, in which case the optimization can be restarted from another initial point and with a smaller tolerance on the termination criteria. For instance, in Algorithm 1, used in the experiments of section 5.7, bad solutions are detected from the resulting maximum absolute error, MAE $= \max\limits_{i \in \{1,\dots,N\}} |y_i - f(x_i)|$. The optimization is restarted with a smaller tolerance Tol, used for the three termination tolerances (change in the variables, change in the objective function and constraint violation), until the MAE is lower than the maximal error allowed and specified by $\overline{\delta} \geq \max_{j=1,\dots,n} \delta_j$. However, when this overall bound is too small compared to the noise level, this condition cannot be satisfied and the algorithm is stopped when Tol reaches a predefined minimal value m.

Remark 16. *Though dealing with nonlinear submodels, problems (5.21) and (5.22) remain linearly constrained nonlinear programs of structure similar to that of (5.19) and (5.18). However, as for choosing different bounds δ_j, care must be taken when using different kernels for different submodels, in which case permuting submodels is no longer without effect and may lead to local minima.*

Algorithm 1 Compute a solution to problem (5.18) or (5.19) that is acceptable with respect to the maximal absolute error (MAE).

```
Require: Data $(\boldsymbol{X}, \boldsymbol{y})$, hyperparameters $\delta_j$, and internal parameters $\overline{\delta}$, TolInit, $m$.
Ensure: The parameter vectors $\boldsymbol{w}_j$ satisfying MAE $\leq \overline{\delta}$.
Tol $\leftarrow$ TolInit
repeat
    $\boldsymbol{u}_0 \leftarrow$ random values $\in [0,1]^{n\times(p+N)}$ {or $[0,1]^{n\times(2p+N)}$ for problem (5.19)}
    call fmincon to solve (5.18) or (5.19) starting from $\boldsymbol{u}_0$
    Tol $\leftarrow 0.1\times$ Tol
until MAE $\leq \overline{\delta}$ or Tol $\leq m$
if MAE $> \overline{\delta}$ then
    print "No solution with MAE $< \overline{\delta}$ can be found, maybe the bound $\overline{\delta}$ is too tight for this data set"
end if
return $\boldsymbol{w}_j,\ j = 1, \dots, n$
```

5.3.5 Refining the parameters

The proposed algorithms can be considered as the initialization step of a more complete estimation procedure. Thanks to these algorithms the main difficulty is overcome and the data can be labeled with their corresponding mode λ_i. Then it is always possible to refine the parameter estimates by any other estimation, approximation or identification algorithm suited for the particular task at hand. Thus obtaining optimality results, such as asymptotic efficiency of the estimator, is not the issue here. For instance, for data corrupted by Gaussian noise, the least squares estimator can be applied to the data of each mode to yield asymptotically optimal parameters.

5.4 Hyperparameters

We now turn to the tuning of the hyperparameters involved in the proposed algorithms: the subsystems' orders, the number of modes n, the bounds on the error δ_j, the kernel types, the internal parameters (only for nonlinear submodels), and the trade-off parameter C.

5.4.1 Estimation of the subsystem orders

Since each linear submodel f_j is parametrized by a different parameter vector $\boldsymbol{w}_j$, the proposed method can be used to model hybrid systems with varying orders, i.e. systems switching between subsystems with different orders. In this case, a different regression vector has to be defined for each mode. Here, estimating the subsystem orders can be seen as selecting variables among the regression vector with an overestimated system order. In this context, feature selection can be performed by shrinkage of the parameters or ℓ_1-norm regularization as in (5.19).

Starting with regression vectors $\boldsymbol{x}_i = [y_{i-1}\ \dots\ y_{i-\bar{n}_a},\ u_{i-\bar{n}_k}\ \dots\ u_{i-\bar{n}_k-\bar{n}_c+1},\ 1]^T$, containing an overestimated number p of regressors, the procedure is as follows.

1. Train a hybrid model by solving (5.19) with the oversized regression matrix $\boldsymbol{X}$.
2. For $i = 1, \dots, N$, estimate the discrete state by $\hat{\lambda}_i = \arg\min_{j=1,\dots,n} |y_i - \boldsymbol{w}_j^T \boldsymbol{x}_i|$.
3. For $j = 1, \dots, n$,
   (a) Build the regression matrix $\boldsymbol{X}_j$ and the target vector $\boldsymbol{y}_j$ by selecting the ith rows of $\boldsymbol{X}$ and $\boldsymbol{y}$ for which $\hat{\lambda}_i = j$.
   (b) For $k = 1, \dots, p$, if $|w_{jk}| < t$ then remove the kth column of $\boldsymbol{X}_j$ corresponding to the kth regressor.
   (c) For $j = 1, \dots, n$, refine the parameter vector $\boldsymbol{w}_j$ by using a suitable standard regression method on the data $(\boldsymbol{X}_j, \boldsymbol{y}_j)$, as in Sect. 5.3.5.

In this scheme, a predefined threshold t is used to determine whether a parameter w_{jk} is zero or not.

Remark 17. *Though the nonlinear identification problems (5.21) and (5.22) could also deal with a different regression vector for each mode, parameter shrinkage cannot be applied in a straightforward manner. For more details on the problem of feature selection for nonlinear predictors, see for instance the reviews [105, 106] and the references therein.*

5.4.2 Estimating the number of modes

The proposed method is well adapted when some basic prior knowledge on the system is available such as the number n of modes. However, due to the universal approximation capability of kernel models, the tuning of n is less critical when using nonlinear submodels than when using linear or affine ones. Indeed for piecewise maps, a good fit can be obtained with an underestimated number of nonlinear submodels.

For linear hybrid systems, a procedure similar to the one used for estimating the subsystem orders can be used to estimate n. In this case, an overestimated number of modes is used and some matrices $\boldsymbol{X}_j$ are expected to vanish, i.e. all the parameters corresponding to mode j are shrank to zero. However in practice, this procedure is not sufficient to detect irrelevant submodels.

The proposed procedure starts by solving (5.19) with an overestimated number of modes and then acts in three steps:

1. *Remove the submodels to which too few data points assigned.* The threshold on the number of data points could be set to the number of submodel parameters.

2. *Prune the submodels w.r.t. the training error.* Submodels that can be removed without increasing the training error above a given threshold can be discarded. In a bounded-error framework, the threshold δ on the training error can be used.

3. *Discard similar submodels.* If multiple submodels have parameter vectors close to each other, i.e. satisfying $\|\boldsymbol{w}_j - \boldsymbol{w}_k\|_2 < c$, only one should be kept in the final model. The threshold c can be set for instance to 0.01.

Remark 18. *For PWARX models, two separate modes can share the same parameter vector in different regions of the regression space. For instance, a mechanical system with backlash typically shows this type of behavior. In this case, the step 3 in the procedure above should be omitted.*

5.4.3 Automatic tuning of the bounds δ_j

Using the ν-Support Vector Regression (ν-SVR) formulation [233] described in section 3.1.3, one can tune automatically the width ε of an ε-insensitive loss function. Applying this trick to the algorithm (5.18) leads to

$$\begin{aligned} \min_{\boldsymbol{w}_j, \boldsymbol{\xi}_j \geq \boldsymbol{0}, \delta_j \geq 0} \ & \sum_{j=1}^{n} \boldsymbol{w}_j^T \boldsymbol{w}_j + C \sum_{i=1}^{N} \prod_{j=1}^{n} \xi_{ij} + \nu N C \sum_{j=1}^{n} \delta_j^2 \\ & -\boldsymbol{\xi}_j - \delta_j \mathbf{1} \leq \boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}_j \leq \delta_j \mathbf{1} + \boldsymbol{\xi}_j, \ j = 1, \dots, n. \end{aligned} \tag{5.24}$$

Notice that, in comparison to [233], the *squares* of the δ_j are minimized instead of the δ_j directly. Minimizing the δ_j directly would produce a shrinkage effect due to an influence function equal to 1. This is not desirable when dealing with noisy data, since the method relies on the fact that some ξ_{ij}, which satisfy $\xi_{ij} \geq \max(0, \ |e_{ij}| - \delta_j)$, are zeros and thus that $e_{ij} \leq \delta_j$ for a certain number of points. On the contrary, in the minimization of a quadratic criterion, the influence function is known to decrease around zero, which leads to solutions with non-zero δ_j.

As one hyperparameter ν has been added, the n hyperparameters δ_j are now automatically tuned. The optimization program (5.24) thus involves $n(p + 2 + N)$ variables.

This algorithm can be further improved by considering adaptive δ_j as above but bounded by $\overline{\delta}$, simply by introducing the constraint $\overline{\delta} \geq \delta_j$, for $j = 1, \dots, n$, in (5.24). Doing so, we impose a bound $\overline{\delta}$ on the error as in the standard approach [15], but also allow the algorithm to shrink this bound in order to produce better solutions.

Note that this trick can be similarly applied to the nonlinear problems (5.21) and (5.22).

5.4.4 Other hyperparameters

The remaining hyperparameters are the trade-off parameter C, the kernel types k_j and their internal parameters (for nonlinear submodels). Their optimal values cannot be simply determined. The common approach to this problem, i.e. tuning the hyperparameters on the basis of a performance index measured on an independent validation set, has to be used. When too few data are available, cross-validation techniques can be used.

5.5 Piecewise system identification

When the system is known to be piecewise, the identification must also return the boundaries between the regions in the regressor space corresponding to the different modes. Such boundaries can then be used to determine the discrete state λ_i. As will be seen, this additional assumption may also be used to enhance the quality of the overall estimation. The following subsections also discuss nonlinearly piecewise systems with boundaries given by nonlinear functions in the regression space.

5.5.1 Partitioning the regression space

Estimating the regions S_j, or equivalently the hyperplanes separating these regions, on the basis of labeled data points is a pattern recognition (or classification) problem (see chapter 2). However, in hybrid system identification, the labeling of the points is not given *a priori* and has to be estimated. In practice, the labels take the values of the estimated discrete state $\hat{\lambda}_i$. To avoid misleading the classifier, the undecidable points (for which more than one ξ_{ij} is zero) can be removed from the training set, or included with small weights if using a method similar to the Weighted SVM [306], described in Sect. 2.4.1. The procedure is as follows.

1. Train a hybrid model on the input-output data $(\boldsymbol{x}_i, y_i)$, $i = 1, \dots, N$, by solving e.g. (5.18) or (5.22).

2. Estimate the discrete state sequence by $\hat{\lambda}_i = \arg\min_{j=1,\dots,n} |e_{ij}|$, $i = 1, \dots, N$.

3. Identify the set U of indices i of undecidable points satisfying $|e_{ij}| < \delta_j$ for more than one submodel f_j.

4. Train a classifier h on the decidable labeled data $(\boldsymbol{x}_i, \hat{\lambda}_i)$, $i \in \{1, \dots, N\} \setminus U$, yielding the mode estimates $h(\boldsymbol{x}) \in \{1, \dots, n\}$.

5. For all $i \in U$, if $|y_i - f_{h(\boldsymbol{x}_i)}(\boldsymbol{x}_i)| < \delta_{h(\boldsymbol{x}_i)}$, reassign the point by $\hat{\lambda}_i = h(\boldsymbol{x}_i)$.

Additionally, in a refinement step similar to the one of Sect. 5.3.5, the submodel f_j can be retrained on the data assigned to mode j by the classifier, for $j = 1, \dots, n$. However, as classification errors may occur, a robust estimator should be used to refine the submodels f_j on the basis of data possibly containing outliers.

5.5.2 Nonlinear boundaries

A direct extension of the PWARX and PWNARX models is obtained by introducing nonlinear boundaries or arbitrary regions. In these "nonlinearly piecewise" models (denoted by NPWARX and NPWNARX, see Table 5.1 in the chapter introduction), the discrete state is still a function of the regressors, but the separating surfaces are no longer restricted to be hyperplanes. This can lead to a decrease of the number of submodels if the true system corresponds to this description. On the contrary, forcing the linear separability of the modes would require building multiple identical submodels for different regions of the regression space that are however governed by the same dynamics. Moreover, regrouping the data available in several regions of the regression space into one mode may help to better estimate the submodels assigned to regions with few samples.

Nonlinear classifiers have to be used in this case and are readily available thanks to the kernel trick as seen in chapter 2. In particular, for the binary case (only 2 modes), the nonlinear separating surface $\mathcal{S}$ of a SVM classifier is given by

$$h(\boldsymbol{x}) = \sum_{i=1}^{N} \beta_i k_c(\boldsymbol{x}_i, \boldsymbol{x}) + b_c = 0, \tag{5.25}$$

where $k_c(.,.)$ is a kernel function and the β_i, b_c are the trainable parameters of the classifier. Simply taking the sign of the function h yields the class of a pattern $\boldsymbol{x}$, i.e. $+1$ if $h(\boldsymbol{x}) \geq 0$ and -1 otherwise.

The method proposed in Sect. 5.3 can deal with nonlinearly piecewise maps without any modification and provides the labeling of the data, required to train a classifier, through $\hat{\lambda}_i$. Indeed, any method (including the bounded-error, Bayesian or algebraic approaches) that estimates the discrete state without dependency on the regressors, and thus without any assumption regarding the linear separability of the data in the regression space, can deal with nonlinearly piecewise maps.

An illustrative example of this procedure is given in Sect. 5.7.3, where the method is applied to estimate a Nonlinearly PieceWise Affine (NPWA) map.

5.6 Interpretations

In this section, three interpretations of the method are proposed. The first one defines a min-min error estimator, while the second one considers a maximum likelihood framework. Then the method is interpreted as a bridge between two hybrid system identification methods: the algebraic [287] and the bounded-error [15] approaches.

5.6.1 Min-min error estimators

Here, we define a min-min error (MME) estimator as an estimator minimizing the error of the particular submodel f_j that leads to the minimal error. In other words, the estimator assigns a sample $(\boldsymbol{x}_i, y_i)$ to the submodel giving the best estimate $f_j(\boldsymbol{x}_i)$ of y_i. A MME estimator is of the form

$$\min_{f_1,\dots,f_n} J^{MME} \propto \sum_{i=1}^{N} \left(\min_{j=1,\dots,n} l(y_i - f_j(\boldsymbol{x}_i)) \right), \tag{5.26}$$

where l is an appropriate loss function. Direct optimization of such an estimator is difficult, since it involves a non-smooth and non-differentiable objective function J^{MME}. However, for admissible loss functions l satisfying $e = 0 \;\Rightarrow\; l(e) = 0$, these estimators can be approximated by Product of Error (PE) estimators given by

$$\min_{f_1,\dots,f_n} J^{PE} \propto \sum_{i=1}^{N} \prod_{j=1}^{n} l(y_i - f_j(\boldsymbol{x}_i)). \tag{5.27}$$

In this framework, the problem (5.18) is a regularized version of the PE estimator using the ε-insensitive loss function (3.4), and thus an approximation of the MME estimator for this choice of l.

5.6.2 Maximum likelihood of the most likely

The maximum likelihood (ML) approach consists in finding the function f that most likely generated the data. Thus, the aim is to maximize the functional $p(\boldsymbol{x}, y|f)$ with respect to f. Consider f as a model switching between n submodels f_j, where the active submodel for a training sample $(\boldsymbol{x}_i, y_i)$ is entirely defined by the set of submodels f_j. Then f only depends on the submodels f_j

and the likelihood can be written as $p(\boldsymbol{x}_i, y_i|f_1, \dots, f_n) = p(y_i|\boldsymbol{x}_i, f_1, \dots, f_n)p(\boldsymbol{x}_i)$ for a particular point. Thus for N i.i.d. samples, the ML estimator may be written as

$$\begin{aligned}\max_{f_1,\dots,f_n} J &= p(\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}, \{y_1, \dots, y_N\}|f_1, \dots, f_n)\\ &= \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, f_1, \dots, f_n)p(\boldsymbol{x}_i).\end{aligned}$$

Maximizing this likelihood is equivalent to minimizing the log-likelihood as

$$\min_{f_1,\dots,f_n} \sum_{i=1}^{N} - \ln p(y_i|\boldsymbol{x}_i, f_1, \dots, f_n). \tag{5.28}$$

In this framework, a Maximum Likelihood of the Most Likely (MLML) estimator considers that for all training samples $(\boldsymbol{x}_i, y_i)$, the active submodel f_j is the most likely, i.e. the one with maximal likelihood of the sample w.r.t. f_j. This leads to

$$f(\boldsymbol{x}_i) = f_j(\boldsymbol{x}_i), \text{ for } j = \arg\max_{k=1,\dots,n} p(y_i|\boldsymbol{x}_i, f_k). \tag{5.29}$$

Thus, the likelihood of a particular sample w.r.t. the model f, involved in the criterion of (5.28), can be written as

$$p(y_i|\boldsymbol{x}_i, f_1, \dots, f_n) = p(y_i|\boldsymbol{x}_i, f_j), \tag{5.30}$$

where j is defined by (5.29). For all training samples, this leads to[4]

$$p(y_i|\boldsymbol{x}_i, f_1, \dots, f_n) \propto \max_{j=1,\dots,n} p(y_i|\boldsymbol{x}_i, f_j). \tag{5.31}$$

Replacing the maximization in (5.31) by the minimization of the log-likelihood allows to write the right-hand side as

$$\max_{j=1,\dots,n} p(y_i|\boldsymbol{x}_i, f_j) = \exp\left(- \min_{j=1,\dots,n} - \ln p(y_i|\boldsymbol{x}_i, f_j)\right).$$

Plugging this result into (5.28) leads to the maximum likelihood of the most likely (MLML) estimator given as the solution of

$$\min_{f_1,\dots,f_n} J^{MLML} = \sum_{i=1}^{N} \left(\min_{j=1,\dots,n} - \ln p(y_i|\boldsymbol{x}_i, f_j)\right). \tag{5.32}$$

Here again, the problem amounts to a min-min optimization program not easily solvable. However, there exists an equivalence between MLML and MME estimators. Choosing a particular loss function l for a MME estimator of the form (5.26) corresponds to a particular choice of noise probability density function (p.d.f.), $p(y_i|\boldsymbol{x}_i, f_j)$, for a MLML estimator (5.32). For the ε-insensitive loss function, this p.d.f. is given by

$$p(y_i|\boldsymbol{x}_i, f_j) = \frac{1}{2(1+\varepsilon)} \exp\left(-l\left(y_i - f_j(\boldsymbol{x}_i)\right)\right). \tag{5.33}$$

where l is given by (3.4) for $\varepsilon = \delta_j$.

Thus the problem (5.18) is a regularized approximation of the MLML estimator using the p.d.f. above.

[4] This relation is not a strict equality as the likelihood has to be normalized in order to integrate to 1.

5.6.3 Algebraic method with bounded error

The proposed method can be interpreted as a bridge between the bounded-error approach [15] and the algebraic procedure [287], while providing nonlinear extensions to these. More precisely, it amounts to a bounded-error relaxation of the hybrid decoupling constraint used in the algebraic procedure, as follows.

The hybrid decoupling constraint (5.5) of the algebraic procedure can be expressed as a function of the submodel errors, $e_{ij} = y_i - f_j(\boldsymbol{x}_i)$, by

$$\prod_{j=1}^{n} e_{ij} = 0,\ i = 1, \dots, N. \tag{5.34}$$

These constraints account for the fact that there must be at least one of the submodels f_j that can estimate the ith point with zero error. In the case of noisy data, these constraints cannot be satisfied for all the N points. On the other hand, the bounded-error constraints (5.9) are less restrictive on the estimation error (with a threshold δ), while acting similarly to (5.34). Combining these two approaches results in constraints of the form

$$\prod_{j=1}^{n} [|e_{ij}| \geq \delta] = 0,\ i = 1, \dots, N, \tag{5.35}$$

where $[\cdot] = 1$, if the bracketed expression is true, and 0 otherwise. Using these constraints, the absolute value of the error $|y_i - f_j(\boldsymbol{x}_i)| = \min_{j=1,\dots,n} |e_{ij}|$ (assuming that $\hat{\lambda}_i = \arg\min_{j=1,\dots,n} |e_{ij}|$) of the hybrid model is bounded by the threshold δ. Approximating $[|e_{ij}| \geq \delta]$ for all j by an ε-insensitive loss function and minimizing their products leads to the algorithms (5.18), (5.19), (5.21) or (5.22) depending on the chosen regularizer.

5.7 Numerical examples

In this section, the performance of the proposed approach is evaluated on six different examples and one case study. The first example shows the automatic tuning of the bounds on the error for the estimation of a PWA map with a switching noise level. The second example considers SARX system identification with fixed and varying orders. The next two examples consider nonlinearly piecewise affine models, respectively for a regression problem and a hybrid dynamical system identification problem. The last two examples show the simultaneous estimation of two functions, one linear and one nonlinear, and the identification of a SNARX system switching between linear and nonlinear dynamics. In these two examples the discrete state is arbitrarily switched and the type of nonlinearity is unknown. Finally, the identification of a hybrid tank system is considered as a case study in different settings including the issue of unbalanced data and the estimation of the number of modes.

In the experiments, the linear or affine dynamical subsystems obey the equation $y_i = \boldsymbol{\theta}_{\lambda}^T \boldsymbol{x}_i$, where $\boldsymbol{\theta}_j = [\theta_{j1}\ \theta_{j2}\ \dots\ \theta_{jp}]^T$, $j = 1, \dots, n$, are the true parameter vectors and λ_i is the discrete state at time i. Static functions will be written as $y(\boldsymbol{x}) = \boldsymbol{\theta}_{\lambda}^T \boldsymbol{x}$, where λ is the true mode for point $\boldsymbol{x}$.

The functions f and f_j will respectively refer to the hybrid model and its submodels. Similarly, for linear (or affine) submodels, $\boldsymbol{w}_j$ refers to the estimated parameters. For switched models (e.g. SARX and SNARX), the discrete state or mode is estimated by $\hat{\lambda}_i = \arg\min_{j=1,\dots,n} |y_i - f_j(\boldsymbol{x}_i)|$. For piecewise models (e.g. PWARX or PWNARX), these estimates are used to train a classifier, yielding the final mode estimates $\hat{\lambda}_i$.

All the optimization programs are solved by the Matlab function *fmincon* and Algorithm 1 is used to detect local minima.

5.7.1 Switching noise level and automatic tuning

The first example shows the approximation of a one-dimensional PWA map with two modes given by

$$y(x) = \begin{cases} \boldsymbol{\theta}_1^T \boldsymbol{x} + u = \begin{bmatrix} 1 & -0.5 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} + u = x - 0.5 + u, & \text{if } x < -2 \\ \boldsymbol{\theta}_2^T \boldsymbol{x} + v = \begin{bmatrix} -0.5 & 0 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} + v = -0.5x + v, & \text{otherwise,} \end{cases} \tag{5.36}$$

where u and v are two noise terms uniformly distributed in the intervals $[-\beta_1, \beta_1]$ and $[-\beta_2, \beta_2]$ respectively. This example also emphasizes the need for multiple bounds δ_j on the errors in case of a switching noise level, and shows the automatic tuning of these bounds to the noise levels.

30 points in the interval $-5 \leq x < -2$ are generated for mode 1 and 41 points in the interval $-2 \leq x \leq 2$ for mode 2. In the first set of experiments, β_1 and β_2 are set to 0.8 and 0.2 respectively. All the algorithms use $C = 100$. The left-hand side of Figure 5.2 shows the approximation of the data by algorithm (5.18) for fixed bounds $\delta_1 = \delta_2 = 0.6$. When using the same bound on the error for the two submodels, the data are correctly approximated within the bounds, but the bound is clearly too large for the data corrupted by low level noise, leading to $\boldsymbol{w}_2 = [-0.344 \ \ -0.141]^T$. In this case, using different bounds for each model, in accordance to the noise levels, may lead to a better approximation.

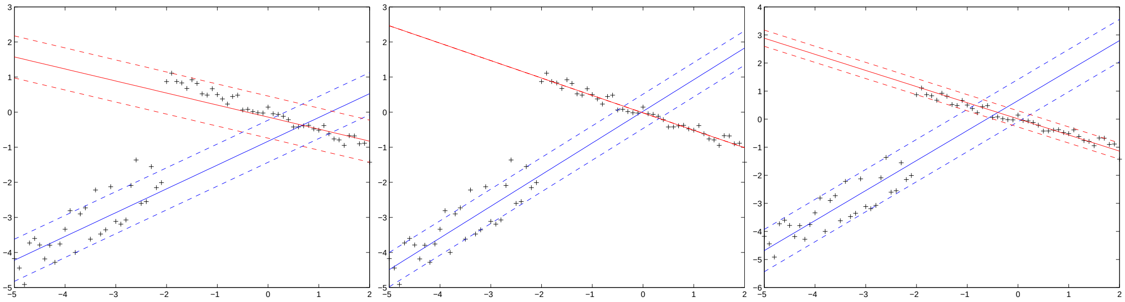


Figure 5.2: Simultaneous estimation of 2 affine functions with different noise levels. *Left:* using the same fixed bound on the error for the 2 submodels, the bound is clearly too large for $x \geq -2$. *Middle:* automatic tuning of the bounds with $\nu = 1$ leads to a better fit. *Right:* the best fit is obtained for $\nu = 0.1$, which also allows to estimate the noise level by the bounds δ_j.

Algorithm (5.24), which automatically tunes the bounds δ_j, is then used with an overall bound $\overline{\delta} = 1$ and $\nu = 1$ to estimate both the linear submodels and the noise levels. As shown in the middle of Figure 5.2, this leads to a better fit for $x \geq 2$ with $\boldsymbol{w}_2 = [-0.494 \ \ -0.016]^T$. However, the final bounds on the errors, $\delta_1 = 0.45$ and $\delta_2 = 0.00$, do not correspond to the noise levels. A better estimation of the noise levels can be obtained for $\nu = 0.1$ as shown at the right of Fig. 5.2. Table 5.3 summarizes the results w.r.t. the parameter estimates and the bounds δ_j. For the three settings, a bias can be observed for the offset parameter θ_{12}, due to a high noise level $\beta_1 = 0.8$ and a low number of data (30 points generated with the parameters $\boldsymbol{\theta}_1$).

Automatic tuning of the thresholds δ_j is now evaluated w.r.t. the noise level. The noise level for mode 1 ($x < -2$) is set to $\beta_1 = 0.5$, while the one for mode 2 ($x \geq 2$) varies from $\beta_2 = 0.1$ to $\beta_2 = 1$. Figure 5.3 shows the mean and standard deviations over 100 runs of the the estimated parameters. During the experiments, δ_1 and δ_2 given by the algorithm (5.24) evolve in accordance with the noise levels β_1 and β_2. Thanks to this automatic tuning, the averages of the parameter estimates are mostly insensitive to the noise level, while their standard deviations increase only slightly for high levels of noise.

Table 5.3: Parameter estimates of a PWA map with switching noise level.

		δ_1	δ_2				
true parameters				θ_{11}	θ_{12}	θ_{21}	θ_{22}
				1	*0.5*	*-0.5*	*0*
estimated parameters				w_{11}	w_{12}	w_{21}	w_{22}
with fixed bounds		0.6	0.6	0.678	-0.830	-0.343	-0.141
with automatic tuning	$\nu = 1$	0.45	0.00	0.828	-0.287	-0.494	-0.016
	$\nu = 0.1$	0.75	0.29	1.069	0.662	-0.575	0.011

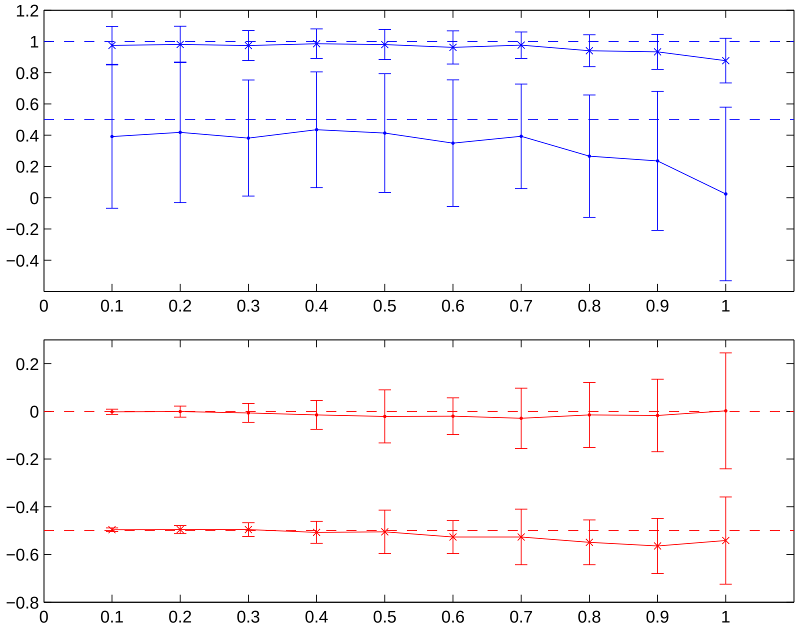


Figure 5.3: Average of the estimated parameters $\boldsymbol{w}_1$ (top) and $\boldsymbol{w}_2$ (bottom) over 100 runs with their standard deviations versus the noise level β_2. The true parameters are also represented (dash lines).

5.7.2 Switched linear system identification

Consider the hybrid system switching between two modes as

$$y_i = \begin{cases} \boldsymbol{\theta}_1^T \boldsymbol{x}_i + e_i = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} y_{i-1} \\ u_{i-1} \end{bmatrix} + e_i = y_{i-1} + u_{i-1} + e_i, & \text{if } \lambda_i = 1 \\ \boldsymbol{\theta}_2^T \boldsymbol{x}_i + e_i = \begin{bmatrix} -0.905 & 0.048 \end{bmatrix} \begin{bmatrix} y_{i-1} \\ u_{i-1} \end{bmatrix} + e_i = -0.905 y_{i-1} + 0.048 u_{i-1} + e_i, & \text{if } \lambda_i = 2. \end{cases}$$

where e_i is a zero-mean Gaussian noise of standard deviation 0.1. An output trajectory of 100 points of this system is generated with a random initial condition y_0, a random input sequence $u_i \sim \mathcal{N}(0,1)$ and random mode switches. A training set is built from this trajectory by considering the regression vector $\boldsymbol{x}_i = [y_{i-1} \ u_{i-1}]^T$. Then problem (5.18) is solved with $C = 100$ and $\delta_1 = \delta_2 = 0.1$. The trajectory of the resulting SARX model used in simulation, $\hat{y}_i = f_{\hat{\lambda}_i}(\hat{y}_{i-1}, u_{i-1})$, is shown Figure 5.4. The estimated parameters are

$$\boldsymbol{w}_1 = \begin{bmatrix} 0.989 \\ 1.02 \end{bmatrix}, \text{ and } \boldsymbol{w}_2 = \begin{bmatrix} -0.893 \\ 0.043 \end{bmatrix}. \tag{5.37}$$

As shown in Fig. 5.4, the mode is correctly estimated by $\hat{\lambda}_i$ without requiring a dwelling time and only 5 classification errors occur on the whole trajectory, which corresponds to 5 % classification error. Moreover, the effect of these classification errors is limited and their origin can be explained. They occur on ambiguous points for which $f_1(\boldsymbol{x}_i) = f_2(\boldsymbol{x}_i) \pm (\delta_1 + \delta_2)$. In the case of a PWA system, these ambiguities could be removed by classifying the points w.r.t. a separating boundary in the regression space.

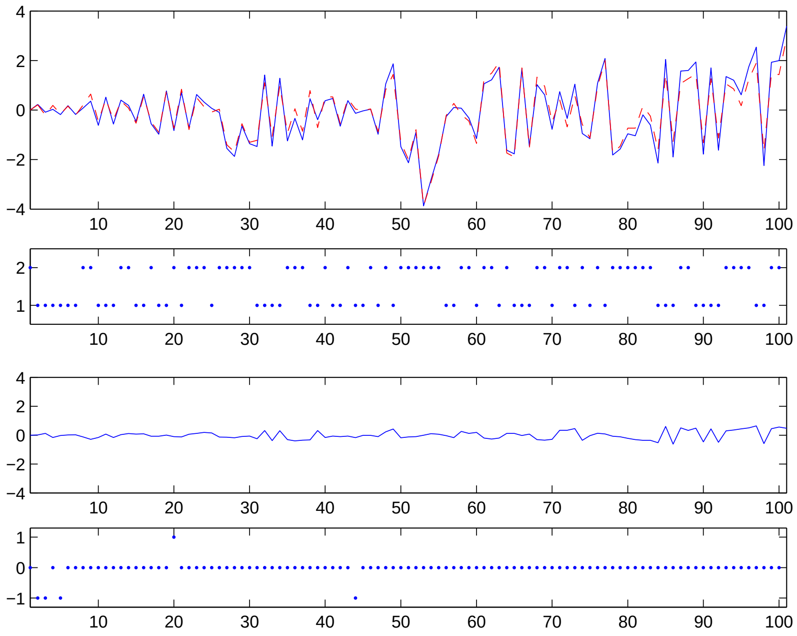


Figure 5.4: *From top to bottom*: (a) Trajectory of the system (plain line) and of the model (dash line) in simulation (only the initial condition and the input are given to the model); (b) Estimated discrete state $\hat{\lambda}_i$; (c) Output error $y_i - \hat{y}_i$; (d) Classification error $\lambda_i - \hat{\lambda}_i$.

Varying orders

Consider the hybrid system switching between two subsystems of different orders as

$$y_i = \begin{cases} \boldsymbol{\theta}_1^T \boldsymbol{x}_i = \begin{bmatrix} 0.2 & 0.4 & 0.2 \end{bmatrix} \begin{bmatrix} y_{i-1} \\ y_{i-2} \\ u_{i-1} \end{bmatrix} = 0.2 y_{i-1} + 0.4 y_{i-2} + 0.2 u_{i-1}, & \text{if } \lambda_i = 1 \\ \boldsymbol{\theta}_2^T \boldsymbol{x}_i = \begin{bmatrix} 0.5 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_{i-1} \\ y_{i-2} \\ u_{i-1} \end{bmatrix} = 0.5 y_{i-1} + u_{i-1}, & \text{if } \lambda_i = 2. \end{cases} \tag{5.38}$$

A simulation of this system with the initial condition $y_0 = y_{-1} = 0$, a uniformly distributed random input sequence u_i and random mode switches is used to build a training set of 50 points. The aim of this example is to show that the method can deal with varying orders, automatically estimated by the training. The parameters estimated by solving (5.19) for $C = 100$ and $\delta_1 = \delta_2 = 0.01$ are

$$\boldsymbol{w}_1 = \begin{bmatrix} 0.1951 \\ 0.3975 \\ 0.1950 \end{bmatrix}, \text{ and } \boldsymbol{w}_2 = \begin{bmatrix} 0.4800 \\ 0.0000 \\ 1.0102 \end{bmatrix}. \tag{5.39}$$

The parameter w_{22} is zero (up to 10^{-33}), thus indicating that the corresponding regressor y_{i-2} is irrelevant for submodel f_2. This shows that the ℓ_1-norm regularization can be used to estimate the subsystem's orders. On the contrary, using the ℓ_2-norm regularization and solving (5.18) leads to the parameters

$$\boldsymbol{w}_1 = \begin{bmatrix}0.2112\\0.3806\\0.2113\end{bmatrix}, \text{ and } \boldsymbol{w}_2 = \begin{bmatrix}0.6489\\-0.4687\\1.1478\end{bmatrix}, \tag{5.40}$$

which do not allow to select the relevant regressors.

5.7.3 Nonlinearly PWA map: reducing the number of submodels

In this illustrative example, the problem is to estimate a Nonlinearly PieceWise Affine (NPWA) map defined as

$$y(x) = \begin{cases} x + 0.5 + e, & \text{if } x \in \,]-\infty, -1] \bigcup [1, \infty[\\ -0.5x - 1 + e, & \text{if } x \in \,]-1, 1[, \end{cases} \tag{5.41}$$

where e is a zero-mean Gaussian noise with standard deviation 0.1. This problem could be solved by considering a PWA map with 3 modes linearly separable w.r.t. the x variable. Here, we propose to consider only 2 modes separated by a nonlinear boundary. Two submodels with linear kernels are trained on $N = 60$ data points by solving (5.21) for $\delta_1 = \delta_2 = 0.1$ and $C = 100$. The parameters of the resulting submodels, shown on Fig. 5.5, are $w_1 = \boldsymbol{X}^T\boldsymbol{\alpha}_1 = 1.01$, $b_1 = 0.53$, $w_2 = \boldsymbol{X}^T\boldsymbol{\alpha}_2 = -0.48$, and $b_2 = -1.02$. All the points are associated to the correct model except for one point, $(x_i, y_i) = (-0.9, -0.31)$, close to the mode boundary. Training a QP-SVM classifier, with a polynomial kernel $k_c(x_i, x) = (xx_i + 1)^3$, on the data labeled by $\hat{\lambda}_i$, $i = 1, \dots, N$, yields a nonlinear boundary $\mathcal{S}$ between the modes given by 2 support vectors $x_1 = -3$ and $x_{60} = 2.9$. As the data x_i are in $\mathbb{R}$, this nonlinear separating surface is a set of points defined as $\mathcal{S} = \{x \,:\, h(x) = -0.24(-3x+1)^3 - 0.22(2.9x+1)^3 + 10.3 = 0\}$, as shown on the right hand side of Fig. 5.5. The resulting switching rule is

$$f(x) = \begin{cases} 1.01x + 0.53, & \text{if } x \in \,]-\infty, -0.86] \bigcup [0.95, 10.7[\\ -0.48x - 1.02, & \text{if } x \in \,]-0.86, 0.95[\bigcup [10.7, \infty[. \end{cases} \tag{5.42}$$

The classifier yields no classification error w.r.t. the target labels $\hat{\lambda}_i$ and approximates rather well the true boundary given in (5.41). Note that due to the polynomial kernel k_c of degree 3, an additional switch occurs at $x = 10.7$. However, this is far beyond the range of the training data, where the correctness of the model cannot be guaranteed in any case.

5.7.4 Nonlinearly piecewise dynamical system

Consider the nonlinearly piecewise dynamical system (NPWARX) with regression vector $\boldsymbol{x}_i = [y_{i-1}\ y_{i-2}\ u_{i-1}]^T$ and a switching boundary between two modes, $h(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x} - 0.5$, corresponding to the centered sphere of radius $\sqrt{2}/2$:

$$y_i = \begin{cases} \boldsymbol{\theta}_1^T\boldsymbol{x}_i + e_i = \begin{bmatrix}0.2 & -0.1 & 0.2\end{bmatrix}\begin{bmatrix}y_{i-1}\\y_{i-2}\\u_{i-1}\end{bmatrix} + e_i = 0.2y_{i-1} - 0.1y_{i-2} + 0.2u_{i-1} + e_i, & \text{if } h(\boldsymbol{x}_i) \geq 0 \\ \boldsymbol{\theta}_2^T\boldsymbol{x}_i + e_i = \begin{bmatrix}0.8 & -0.2 & 0.8\end{bmatrix}\begin{bmatrix}y_{i-1}\\y_{i-2}\\u_{i-1}\end{bmatrix} + e_i = 0.8y_{i-1} - 0.2y_{i-2} + 0.8u_{i-1} + e_i, & \text{otherwise.} \end{cases} \tag{5.43}$$

The training set is composed of $N = 100$ samples generated by (5.43) with zero-mean Gaussian noise e_i of standard deviation 0.1 and an input sequence uniformly distributed in the range $u_i \in [-1, 1]$. The results on a test set generated for a different input sequence $\{u_i\}$ are shown in Fig. 5.6. The parameters estimated by (5.24) with automatic tuning are close to the true values: $\boldsymbol{w}_1 = [0.2508\ -0.1648\ 0.1925]^T$ and $\boldsymbol{w}_2 = [0.8387\ -0.1934\ 0.7448]^T$. The classification error rate on the test data obtained by the QP-SVM classifier is 8%.

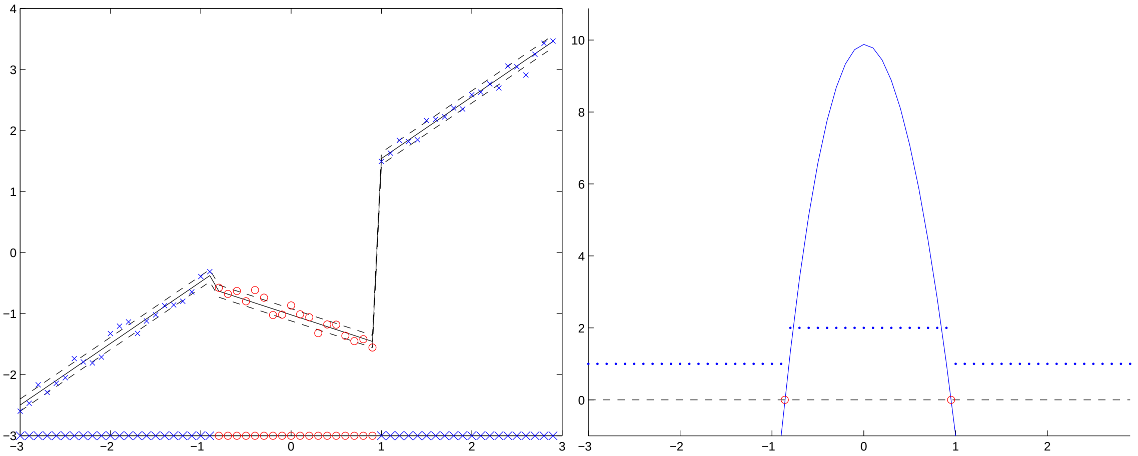
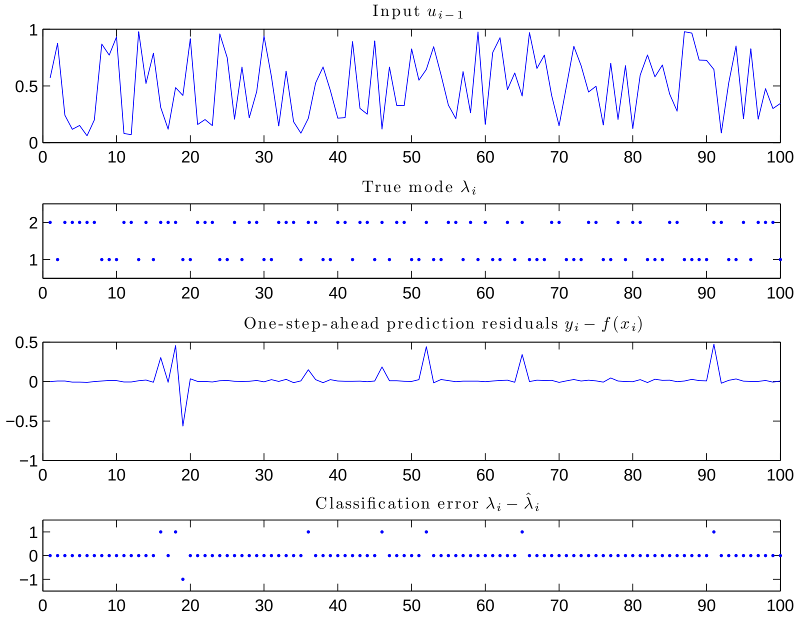


Figure 5.5: Nonlinear boundary between 2 modes. *Left*: Hybrid model (—) with its insensitivity tube (- -) approximating the data points represented by '×' for $\hat{\lambda}_i = 1$ and '○' for $\hat{\lambda}_i = 2$. The estimated mode $\hat{\lambda}_i$ also appears on the x-axis as '×' for mode 1 and '○' for mode 2 to highlight the partition of the input space $\mathcal{X}$. *Right*: Class labels $\hat{\lambda}_i$ (.) for each data point x_i used to learn the nonlinear boundary $\mathcal{S}$ (○), defined as the zeros of $h(x)$ (—).


Input u_{i-1}

True mode λ_i

One-step-ahead prediction residuals $y_i - f(x_i)$

Classification error $\lambda_i - \hat{\lambda}_i$

Figure 5.6: Test data and estimation for a nonlinearly piecewise linear system.

5.7.5 Switching function with unknown nonlinearity

In this one-dimensional example, the true function $y(x)$ arbitrarily switches between an affine and a nonlinear function as

$$y(x) = \begin{cases} 2x - 1 + e, & \text{if } \lambda = 1 \\ 0.5x^2 + e, & \text{if } \lambda = 2. \end{cases} \tag{5.44}$$

The training data are generated with a zero-mean Gaussian noise e of variance 0.25. Two data points, one for $\lambda_i = 1$ and one for $\lambda_{i+30} = 2$, are generated for 30 values of x in the interval $]-5, 1[$. Beside these 60 samples, the only prior knowledge is that one submodel is linear and the other is nonlinear. The aim of this example is to show that the proposed method can discriminate between the two submodels and correctly approximate each one without further knowledge on the type of nonlinearity. Figure 5.7 shows the data and the resulting submodels obtained by solving (5.21) for $\delta_1 = \delta_2 = 0.5$, $C = 100$, a linear kernel k_1 and a Gaussian RBF kernel k_2 with $\sigma = 2$. For the linear submodel f_1, the estimated parameters are $w_1 = 1.997$ and $b_1 = -0.863$. A test error is computed w.r.t. the true function (for $e = 0$) on 60 points, resulting in $\text{MSE}[f_1] = 0.0204$ for the linear submodel and $\text{MSE}[f_2] = 0.0646$ for the nonlinear submodel. The overall test error for the two modes, computed on 120 points, assuming that λ is known, is $\text{MSE}[f] = 0.0425$, to be compared to the noise variance of 0.25.

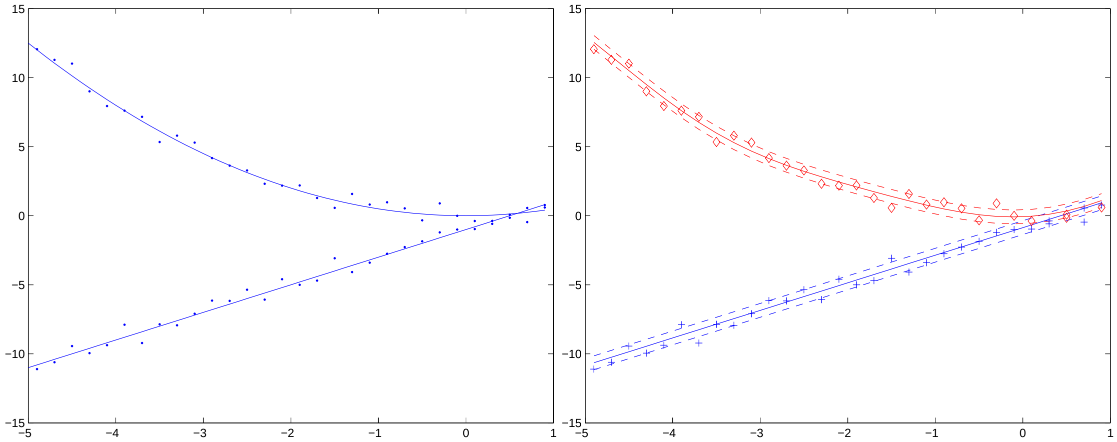


Figure 5.7: *Left:* true function switching between a linear and a nonlinear subfunction (–) shown with the noisy training samples. *Right:* resulting hybrid model (–) with its insensitivity tubes (- -). Points associated to the linear and RBF models are respectively represented by crosses (+) and diamonds (◇).

5.7.6 Switched nonlinear dynamical system

Consider the arbitrarily switched system

$$y_i = \begin{cases} g(\boldsymbol{x}_i) + e_i = -0.4y_{i-1}^2 + 0.5u_{i-1} + e_i, \text{ if } \lambda_i = 1 \\ \boldsymbol{\theta}_2^T \boldsymbol{x}_i + e_i = \begin{bmatrix} -0.905 & 0.9 \end{bmatrix} \begin{bmatrix} y_{i-1} \\ u_{i-1} \end{bmatrix} + e_i = -0.905y_{i-1} + 0.9u_{i-1} + e_i, \text{ if } \lambda_i = 2, \end{cases} \tag{5.45}$$

where e_i is a noise term. An output trajectory of $N = 100$ points of this system is generated with a random initial condition y_0, a random input sequence u_i uniformly distributed in the interval $[0, 1]$ and a mode switch from mode 1 to mode 2 at iteration $i = 41$. For these training data, the noise e_i is a zero-mean Gaussian noise of standard deviation 0.1. The signal-to-noise ratio of this trajectory is 12 dB corresponding to a variance of the noise free trajectory of 0.20 and a noise variance of 0.012. These data are then used to train a SNARX model by solving (5.21) with $C = 1000$, $\delta_1 = \delta_2 = 0.1$, a RBF kernel k_1 with $\sigma = 1$ and a linear kernel k_2. Thus, the only prior knowledge is that one submodel is nonlinear and the other is linear. The training data are shown in Figure 5.8. The estimated parameters of the linear submodel are $\boldsymbol{w}_2 = [-0.929\ 0.960]^T$, to be compared to $\boldsymbol{\theta}_2 = [-0.905\ 0.9]^T$. As shown on Fig. 5.8, 22 classification errors occur for the estimation of the discrete state over the whole training trajectory. Most of them occur on ambiguous points for which $f_1(\boldsymbol{x}_i) = f_2(\boldsymbol{x}_i) \pm (\delta_1 + \delta_2)$, which explains why there are more errors than in the switched linear system identification example of Sect. 5.7.2. Indeed, nonlinear submodels usually have more crossing points than linear submodels. Only 8 support vectors with nonzero α_{ij} are selected from the 100 training samples to build the kernel expansion f_1.

The model is tested on an independent test set generated by (5.45) for $e_i = 0$, another input sequence, uniformly distributed in the interval $[0, 1]$, and another discrete state sequence, given to

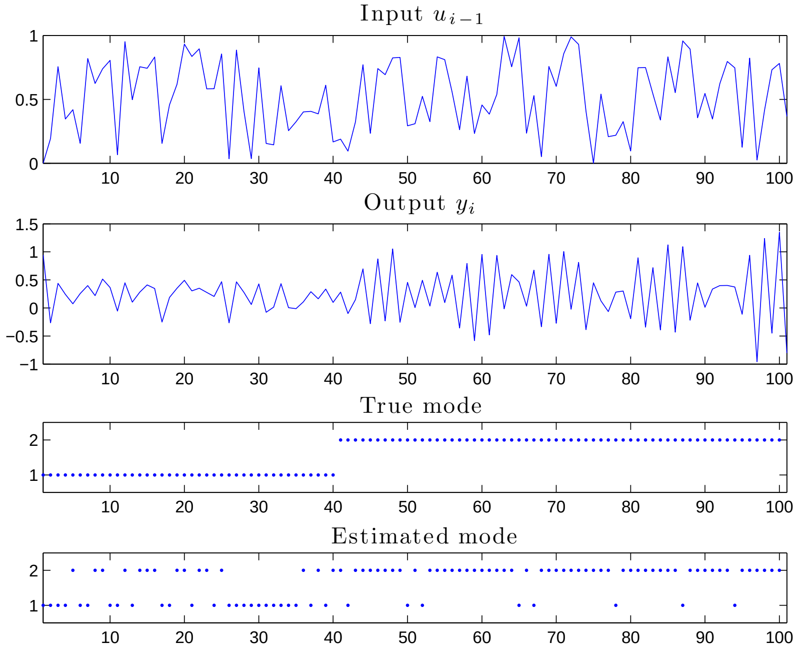


Figure 5.8: Training data and estimated discrete state $\hat{\lambda}_i$ for the switched nonlinear system identification.

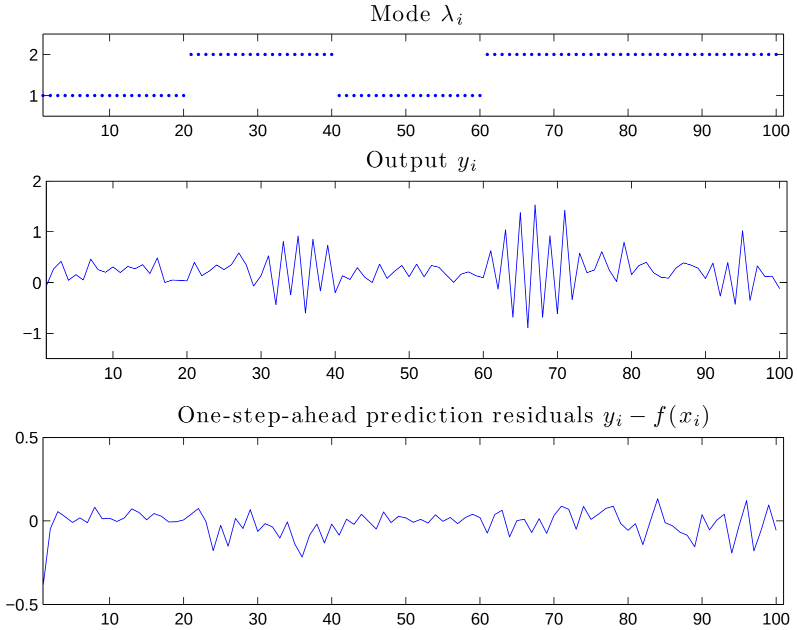


Figure 5.9: Test data and one-step-ahead prediction error for the switched nonlinear system identification.

the model. Figure 5.9 shows the test data and the resulting one-step-ahead prediction residuals $y_i - f_{\lambda_i}(\boldsymbol{x}_i)$. In this setting the model gives a high prediction accuracy both in short term, MSE $= 0.0063$, and in long term, $\text{MSE}_{100} = 0.0201$.

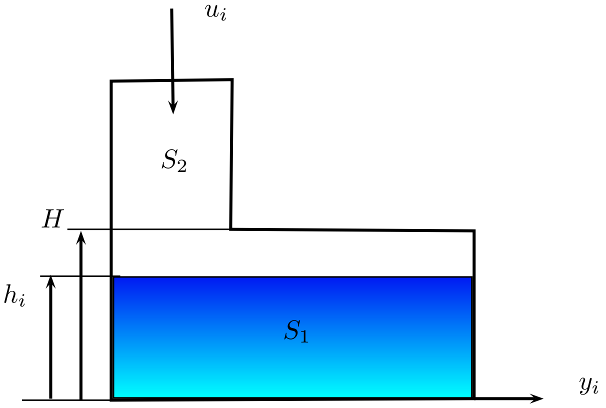


Figure 5.10: Tank system with two different sections S_1 and S_2.

5.7.7 Case study: hybrid tank system

Consider the tank system[5] drawn in Fig. 5.10, where u_i is the input flowrate and y_i the output flowrate depending linearly on the height h_i of the liquid at instant i. The tank has two different sections: S_1 for $h_i \leq H$ and S_2 for $h_i > H$, leading to the hybrid behavior of a piecewise linear system of equations

$$\begin{cases} h_{i+1} = h_i + \frac{u_i - y_i}{S_i} \\ y_i = K_{out} h_i, \end{cases} \tag{5.46}$$

where the instantaneous section S_i switches between S_1 and S_2, and K_{out} is a constant modeling the linear dependency between the height and the output flowrate. The system can be expressed in ARX form as

$$y_i = \left(1 - \frac{K_{out}}{S_{i-1}}\right) y_{i-1} + \frac{K_{out}}{S_{i-1}} u_{i-1}. \tag{5.47}$$

As a function of the output y_i, the switching rule is given by

$$S_i = \begin{cases} S_1, & \text{if } y_i \leq HK_{out} \\ S_2, & \text{otherwise.} \end{cases} \tag{5.48}$$

In the following experiments, the constants are set to $K_{out} = 0.6$, $S_1 = 10$, $S_2 = 3$ and $H = 2$. The goal is thus to recover the true parameters

$$\boldsymbol{\theta}_1 = \begin{bmatrix} \left(1 - \frac{K_{out}}{S_1}\right) \\ \frac{K_{out}}{S_1} \end{bmatrix} = \begin{bmatrix} 0.94 \\ 0.06 \end{bmatrix}, \text{ and } \boldsymbol{\theta}_2 = \begin{bmatrix} \left(1 - \frac{K_{out}}{S_2}\right) \\ \frac{K_{out}}{S_2} \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix}, \tag{5.49}$$

together with the threshold $Y = H \times K_{out} = 1.2$, defining the switching rule w.r.t. y_i.

Unbalanced data

The system (5.46) is simulated for 80 iterations with the initial condition $h_0 = 0$ and a random input uniformly distributed in the interval $u_i \in [0, 4]$. The resulting data are shown on Fig. 5.11. A training set is built from these data by considering regression vectors $\boldsymbol{x}_i = [y_{i-1} \ u_{i-1}]^T$ and targets y_i, for $i = 1, \dots, 80$. In this data set, less than 23% of the samples belong to mode 1, corresponding to $S_{i-1} = S_1$.

The parameters estimated by algorithm (5.24) with $C = 100$ and $\nu = 0.5$ are very close to the true parameters of the system:

$$\boldsymbol{w}_1 = \begin{bmatrix} 0.9397 \\ 0.0601 \end{bmatrix}, \text{ and } \boldsymbol{w}_2 = \begin{bmatrix} 0.8000 \\ 0.2000 \end{bmatrix}. \tag{5.50}$$

[5] A similar system is considered in [216].

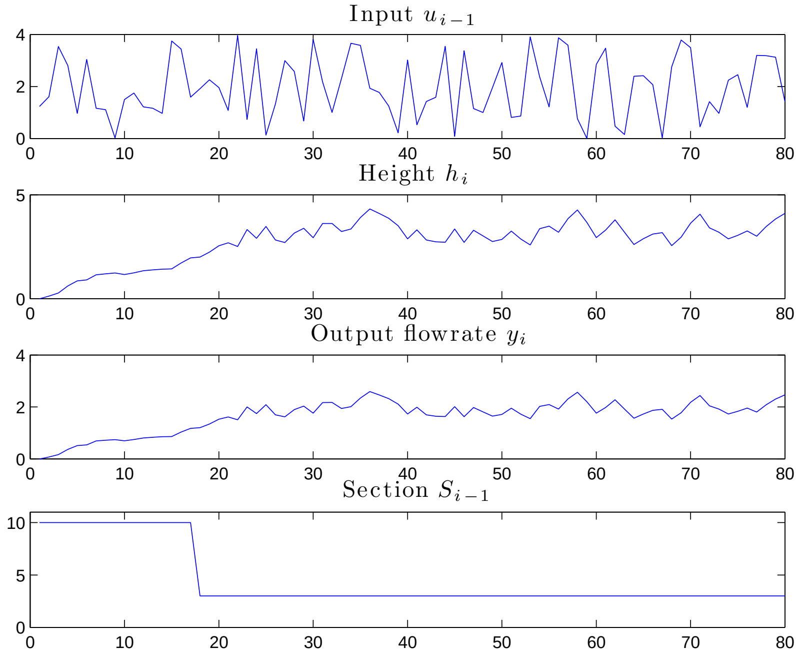


Figure 5.11: Training data from a simulation of the tank system.

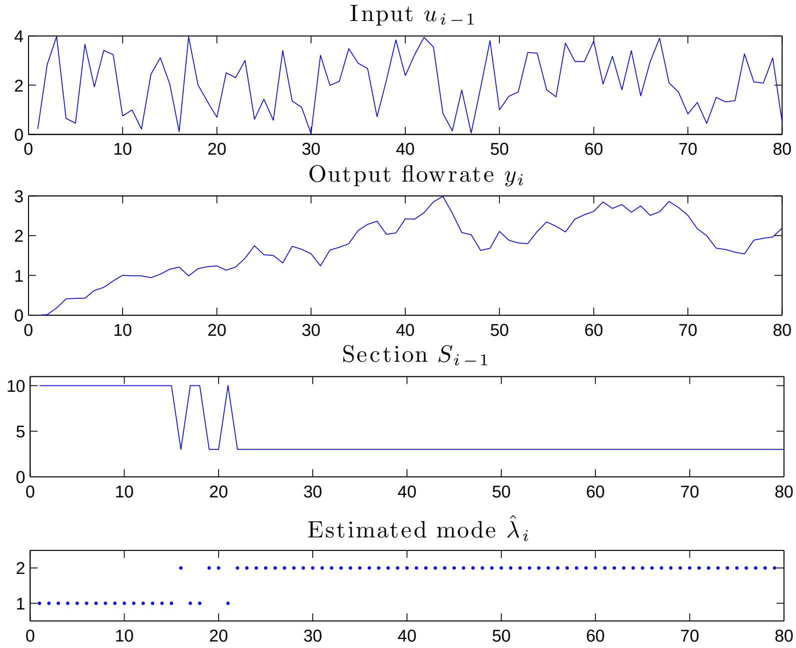


Figure 5.12: Test data and estimated mode for the tank system. The predictions (not plotted) fit exactly y_k.

After training a QP-SVM classifier with a linear kernel and $C = 10$, the estimated threshold is $\hat{Y} = 1.1798$. Figure 5.12 shows a simulation on test data for another random input sequence. The mode is correctly estimated for all points and the model achieves a very high accuracy both in short-term ($\text{MSE} = 1.2 \times 10^{-8}$) and in long-term ($\text{MSE}_{80} = 1.5 \times 10^{-7}$) prediction.

Table 5.4: Number of modes n and parameter vectors estimated with an overestimated number of submodels $\overline{n}$.

$\overline{n}$	n	$\boldsymbol{w}_1$	$\boldsymbol{w}_2$	$\boldsymbol{w}_3$
3	2	$\begin{bmatrix}0.9397\\0.0628\end{bmatrix}$	$\begin{bmatrix}0.7968\\0.1997\end{bmatrix}$	
4	2	$\begin{bmatrix}0.9353\\0.0585\end{bmatrix}$	$\begin{bmatrix}0.7972\\0.1992\end{bmatrix}$	
5	2	$\begin{bmatrix}0.9286\\0.0613\end{bmatrix}$	$\begin{bmatrix}0.7976\\0.1998\end{bmatrix}$	
6	3	$\begin{bmatrix}0.9217\\0.0628\end{bmatrix}$	$\begin{bmatrix}0.7976\\0.1988\end{bmatrix}$	$\begin{bmatrix}0.8887\\0.0756\end{bmatrix}$

Estimating the number of modes

Consider identifying the tank system of Fig. 5.10 without knowing the number of sections. The procedure given in Sect. 5.4.2 is used with the suggested threshold parameters and for different maximum number of modes. The initialization of this procedure is obtained by solving (5.19) for $C = 50$. Table 5.4 gives the estimated number of modes n and the resulting parameter vectors $\boldsymbol{w}_j$. Note that these parameter estimates are given by the first solution of (5.19) without retraining or refinement.

Tank with three modes

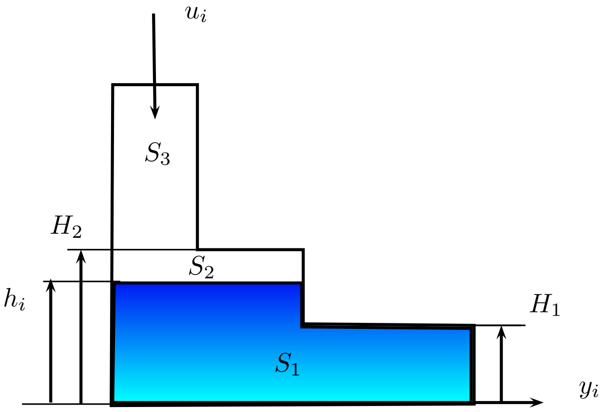


Figure 5.13: Tank system with 3 modes.

Consider now a tank with 3 sections ($S_1 = 10$, $S_2 = 5$, $S_3 = 2$), shown in Fig. 5.13, obeying the equation (5.47) with the switching rule

$$S_i = \begin{cases} S_1, & \text{if } y_i \leq H_1 K_{out} \\ S_2, & \text{if } H_1 K_{out} \leq y_i \leq H_2 K_{out} \\ S_3, & \text{if } y_i \geq H_2 K_{out}, \end{cases} \tag{5.51}$$

and the thresholds $H_1 = 2$, $H_2 = 3$. The true parameters of this system are

$$\boldsymbol{\theta}_1 = \begin{bmatrix}0.94\\0.06\end{bmatrix}, \ \boldsymbol{\theta}_2 = \begin{bmatrix}0.88\\0.12\end{bmatrix}, \text{ and } \boldsymbol{\theta}_3 = \begin{bmatrix}0.7\\0.3\end{bmatrix}. \tag{5.52}$$

The parameters estimated by algorithm (5.24) with $C = 100$ and $\nu = 0.5$ are

$$\boldsymbol{w}_1 = \begin{bmatrix}0.9397\\0.0601\end{bmatrix}, \ \boldsymbol{w}_2 = \begin{bmatrix}0.8799\\0.1200\end{bmatrix}, \ \text{and } \boldsymbol{w}_3 = \begin{bmatrix}0.7000\\0.3000\end{bmatrix}. \tag{5.53}$$

A QP-SVM classifier is trained on the labeled samples $(\boldsymbol{x}_i, \hat{\lambda}_i)$ from the training set for $C = 10$ and a linear kernel. The results on a test trajectory for another random input sequence are plotted in Fig. 5.14. The classification error rate is 7.6% and the one-step-ahead prediction error remains low: MSE $= 2.8 \times 10^{-3}$.

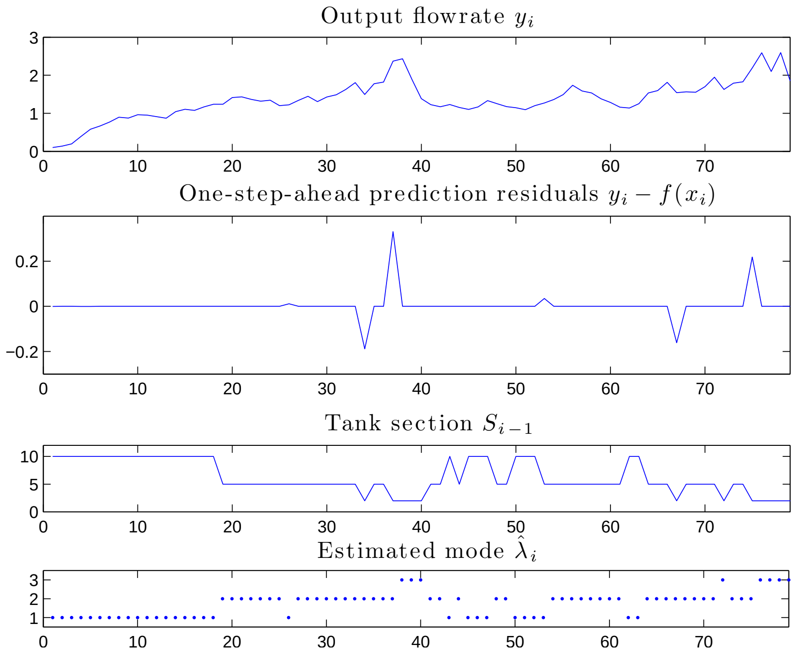


Figure 5.14: Test data for the tank system with 3 modes.

5.8 Conclusion

In this chapter, a new method for hybrid system identification has been developed by using the ε-insensitive loss function proposed in the machine learning community for Support Vector Regression. Since no assumption on the discrete sequence that generate the data is required, Switched ARX system can be treated as well as PWA maps.

In comparison to previous works from the literature, the proposed method has some advantages and allows a number of extensions.

- *Smooth optimization.* The proposed method relies on a smooth optimization program that is less demanding than methods based on Mixed-Integer Programming (MIP) [220] or on Minimal Partition Feasible Sets (MIN PFS) problems [15]. Though remaining a non-convex problem, the complexity is reduced, because finding an exact global optimum is not required. Approximate solutions close to the global minimum are most of the time sufficient. Moreover, bad solutions can be easily detected on the basis of the model error on the training data.

- *Nonlinear hybrid systems.* The proposed method is the first of the kind that can deal with arbitrary and unknown nonlinearities. Previous works are limited to nonlinearities known in a parametrized form.

- *Nonlinearly piecewise systems.* The proposed method allows to estimate nonlinearly piecewise maps. This feature comes from the fact that the problem solved by the algorithm is independent of the switching sequence. Therefore other approaches, such as the algebraic [287] or bounded-error [15] methods, can deal with nonlinearly piecewise systems. However, the present thesis and [152] are the first work discussing this issue.

- *Switching noise level.* The proposed method applies a bounded-error approach in which the bound δ on the error can be set to a different value δ_j for each mode. To the best of our knowledge, this is the first work on hybrid system identification considering that the noise may also switch with the system.
- *Automatic tuning of the bounds δ_j.* In comparison to the bounded-error method described in [15], the proposed algorithm can be automatically adapted to the noise level(s) of the data.

5.9 Further reading

"Hybrid systems: computation and control"

This label is the name of one of the most famous conference in the field of hybrid systems and emphasizes the fact that hybrid systems are at the junction of two communities and research areas: computer science and automatic control. In the thesis, the control point of view is taken and hybrid systems are modeled by recurrent equations (for the discrete-time case). Another approach, often studied by researchers from computer science, is to model hybrid systems by discrete-event systems such as finite-state automata.

Applications

Beside the obvious application to numerous control problems, hybrid systems, and hybrid system identification in particular, are also used in other fields of research.

Computer vision. In computer vision, hybrid system identification can be used as a video segmentation method, see e.g. [286] and references therein. In particular, temporal segmentation, where the aim is to detect switches in the video sequence such as a change of camera, has been tackled by the Generalized Principal Component Analysis (GPCA) [285], related to the algebraic procedure described in Sect. 5.2.1. A similar approach is also used in [282] for spatial-temporal segmentation, i.e. segmentation of video sequences including multiple objects in motion.

Systems biology. Hybrid system identification has recently gained in interest for biological system modeling [210, 72, 54, 58], while automatic control methods in general are more and more applied in a biological context [142]. In particular, gene regulatory networks are modeled using piecewise affine systems and hybrid system identification techniques.

Chapter 6

Conclusions and perspectives

In this thesis, the problem of obtaining a model from data has been treated from a machine learning point of view. More precisely, the so-called Support Vector Machines (SVMs) have been applied to three modeling problems: classification, regression and hybrid system identification.

Focusing on machine learning methods for solving problems emerging from different fields of research, the author worked in favor of an interdisciplinary approach. The hope is that the readers from various domains have developed an interest in machine learning techniques. In addition, the thesis promotes the use of prior knowledge while training black box models.

Classification

This thesis introduced the SVMs and some of their key features in the context of classification. With SVMs being a well established state-of-the-art method in the field of pattern recognition, the next step is to enhance the learning algorithms with prior knowledge. In this respect, chapter 2 discussed the main forms of prior knowledge often encountered in pattern recognition and provided a review of the corresponding SVM methods. An example of application to handwriting recognition has been given to show the use of the virtual sample approach and emphasize the need for the inclusion of prior knowledge.

Open issues

- Building classifiers with multiple invariances. *The review provided in chapter 2 shows that many methods are available for the incorporation of transformation-invariance. However, most of these approaches are suitable for one or few transformations only. Including invariances to multiple transformations in a computationally efficient manner remains a challenging task.*
- Combining different forms of prior knowledge. *As briefly discussed in section 2.5.2, most of the methods can be simply combined in practice. Consider the example of character recognition, for which we have seen that generating virtual samples by elastic distortions improves the results [240] by incorporating invariance to small random variations of the image. However the elastic distortion may sometimes yield very distorted images that become unrecognizable [153] and can mislead the classifier. This drawback could be diminished by combining elastic distortions with a Weighted-SVM on the basis of the prior knowledge that less confidence should be assigned to the virtual samples than to the original training ones. Despite the fact that such combinations could possibly lead to significant performance improvements, the experimental evidence supporting this approach, the formalization of the interaction between the forms of knowledge, and the reduction of the computational load implied by the combinations, remain open issues.*

Regression and nonlinear modeling

In chapter 3, Support Vector Regression has been presented in its three most popular forms (QP-SVR, LP-SVR and LS-SVM) together with some of their extensions (ν-QPSVR, μ-LPSVR and Robust LS-SVM). The theoretical properties of these algorithms have been illustrated in numerical experiments, including a nonlinear system identification example. In particular, the experiments showed that the SVR method is particularly well suited for difficult cases, i.e. when few data are available or when the measurements are corrupted by non-Gaussian noise. In addition, the limits of the asymptotic properties have been highlighted and the robustness to outliers either provided intrinsically by the ε-insensitive loss function or recovered by the Robust LS-SVM procedure has been evaluated.

Open issues

- System identification. *One of the main open issues concerns the application of SVR to system identification. In particular, training recurrent (or output-error) models, where the regression vector includes the predicted outputs instead of the measurements, remains an open problem. Though this problem has been studied in the LS-SVM framework [256], most of the original SVR good features are lost in the process.*

Prior knowledge

For difficult applications, where few data are available, physical knowledge should always be taken into account in the modeling process. A general method has been proposed in chapter 4 to incorporate a wide range of information. In particular, the problem of how to best use simulation data, possibly not very accurate, when learning a model from very few experimental data has been addressed on a real- application: the estimation of the in-cylinder gas fraction. In this case, prior knowledge on the derivatives, provided by a prior model trained on the simulation data, led to the best results. Besides, this example showed the importance of adding the simulation data as potential support vectors when using Gaussian RBF kernels.

Open issues

- Quadratic programming SVR. *In the proposed method, the prior knowledge is incorporated as additional constraints in the LP-SVR framework. Similar results can be obtained for QP-SVR and LS-SVM in a straightforward manner. However, the computational benefits of specialized optimization algorithms such as SMO would be lost in the process. For QP-SVR, the constraints can be added to the dual problem, which can then be solved by standard quadratic programming optimizers. The effect of adding the constraints to the primal problem remains an open issue. Indeed this could change the structure of the resulting model, as it would modify the Lagrangian.*
- Hyperparameters. *The proposed method involves an additional hyperparameter for each set of constraints corresponding to a particular form of prior knowledge. In the numerical examples, at most two types of prior knowledge have been considered simultaneously. For the particular application to the estimation of the in-cylinder gas fraction, the tuning of the hyperparameters was not critical as the optimal values lie in rather flat regions of the test error. However, the sensitivity of the method to the additional hyperparameters when more than two types of knowledge are mixed remains to be explored. In addition, tuning these hyperparameters when no information on the relative confidence of the priors is available, remains an open issue.*
- Simulated experiment design and active prior learning. *When prior knowledge is available through a prior model such as a simulator, simulation data can be generated on demand. The issue of how to generate these samples should be investigated as this may significantly affect the model. This question is strongly related to two fields of research: experiment design and active learning. However, instead of choosing the location of experimental data, here, simulation data representing the prior knowledge must be chosen.*

Hybrid system identification

Searching for a model with a predetermined structure usually reduces the complexity of the problem, but not always. Building hybrid (or piecewise) models is more difficult than finding a single nonlinear model for all the data. However, in automatic control, the structure may be determined by the purpose of the model. For instance, the controller may require a piecewise affine model of the system. Thus taking the prior knowledge into account may improve the performance of the overall control design rather than the quality of the model itself.

In comparison to previous works from the literature, the method proposed in chapter 5 allows a number of extensions, listed in Sect. 5.8. However, a number of open issues remain to be solved.

Open issues

- Online identification. *Another approach to the problem is to consider online (or recursive) hybrid system identification, as studied in [283]. In this setting, the model evolves in time and is typically updated at each time step with the presentation of a new sample. An advantage of online identification is that the structure of the model can also be updated: previously unseen modes can be detected and new submodels created. This is for instance discussed in [12] and also relates to novelty detection [234] and fault diagnosis [14, 146].*
- Computational complexity. *The proposed method suffers from a complexity issue due to nonconvex optimization. From a practical point of view, only Sequential Quadratic Programming (SQP) has been used in all the experiments. Other optimizers might perform better and reduce the computational time (though not reducing the complexity of the problem). In particular, stochastic gradient descent algorithms can be very fast and have been used successfully to solve nonconvex optimization problems for training large neural networks on large datasets. Besides, the complexity of the method comes from the number of slack variables and the number of constraints. A promising perspective could be to reformulate the problem as an unconstrained optimization problem involving only the parameters of the models as variables.*
- Nonconvex optimization. *The proposed method relies on nonconvex optimization, involving local minima. For the training of neural networks, breaking the symmetries appears as a key issue to reducing the number of local minima [159]. As symmetries are inherently present in the hybrid system identification problem, the algorithm could be improved, e.g. by constraining or preventing the permutations of submodels. In addition, though the experiments show that local optimizers can be used, better results could be obtained by global optimization techniques [200].*
- Control of nonlinear hybrid systems. *A number of works are now available for optimal control of piecewise affine and linear hybrid systems, see e.g. [17, 64, 28, 156]. Identifying nonlinear hybrid systems in the form of piecewise or switched kernel expansions implies the issue of kernel-model-based control of nonlinear hybrid systems. In this respect, Model Predictive Control (MPC) with instantaneous linearized models as in [55] may be considered.*

Appendix A

Derivatives of a Gaussian RBF kernel expansion

Considering the RBF kernel $k(\boldsymbol{x}, \boldsymbol{x}_i) = \exp\left(-\|\boldsymbol{x} - \boldsymbol{x}_i\|^2/2\sigma^2\right)$, gives the following derivative of f w.r.t. x^j, the jth component of $\boldsymbol{x}$,

$$\begin{aligned} \frac{\partial f(\boldsymbol{x})}{\partial x^j} &= \frac{1}{\sigma^2}\sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i)(x_i^j - x^j) \\ &= \frac{1}{\sigma^2}(\boldsymbol{X}_j - \boldsymbol{1}x^j)^T \boldsymbol{D}_{\boldsymbol{K}(\boldsymbol{x},\boldsymbol{X}^T)}\ \boldsymbol{\alpha} = \boldsymbol{r}_1(\boldsymbol{x})^T\boldsymbol{\alpha}, \end{aligned} \tag{A.1}$$

where $\boldsymbol{D}_{\boldsymbol{K}(\boldsymbol{x},\boldsymbol{X}^T)} = \operatorname{diag}\left(\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\right)$ and $\boldsymbol{X}_j$ represents the jth column of $\boldsymbol{X}$.

Looking at the second order derivatives

$$\begin{aligned} \frac{\partial^2 f(\boldsymbol{x})}{\partial {x^j}^2} &= \frac{1}{\sigma^2}\sum_{i=1}^{N} \alpha_i \left[\frac{k(\boldsymbol{x}, \boldsymbol{x}_i)(x_i^j - x^j)^2}{\sigma^2} - k(\boldsymbol{x}, \boldsymbol{x}_i)\right] \\ &= \frac{1}{\sigma^2}\sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i)\left(\frac{(x_i^j - x^j)^2}{\sigma^2} - 1\right), \end{aligned} \tag{A.2}$$

gives

$$\nabla^2 f(\boldsymbol{x}) = \frac{1}{\sigma^2}\sum_{j=1}^{p}\sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i)\left(\frac{(x_i^j - x^j)^2}{\sigma^2} - 1\right), \tag{A.3}$$

which becomes in matrix form

$$\begin{aligned} \nabla^2 f(\boldsymbol{x}) &= \frac{1}{\sigma^4}\sum_{j=1}^{p}\left[((\boldsymbol{X}_j - \boldsymbol{1}_N x^j)^2)^T \boldsymbol{D}_{\boldsymbol{K}(\boldsymbol{x},\boldsymbol{X}^T)}\,\boldsymbol{\alpha}\right] - \frac{p}{\sigma^2}\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\boldsymbol{\alpha} \\ &= \left[\frac{1}{\sigma^4}\sum_{j=1}^{p}\left[((\boldsymbol{X}_j - \boldsymbol{1}_N x^j)^2)^T \boldsymbol{D}_{\boldsymbol{K}(\boldsymbol{x},\boldsymbol{X}^T)}\right] - \frac{p}{\sigma^2}\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T)\right]\boldsymbol{\alpha} \\ &= \boldsymbol{r}_2(\boldsymbol{x})^T\boldsymbol{\alpha}, \end{aligned} \tag{A.4}$$

where $(\boldsymbol{X}_j - \boldsymbol{1}x^j)^2$ is a vector with all components equals to the squares of the components of $(\boldsymbol{X}_j - \boldsymbol{1}x^j)$.

To summarize the results, the following parameters can be used to constrain the first order derivative w.r.t. x^j and the Laplacian

$$\boldsymbol{r}_1(\boldsymbol{x})^T \quad = \quad \frac{1}{\sigma^2}(\boldsymbol{X}_j - \boldsymbol{1}x^j)^T \boldsymbol{D}_{\boldsymbol{K}(\boldsymbol{x},\boldsymbol{X}^T)} \tag{A.5}$$

$$\boldsymbol{r}_2(\boldsymbol{x})^T \quad = \quad \frac{1}{\sigma^4}\sum_{j=1}^{p}\left[((\boldsymbol{X}_j - \boldsymbol{1}x^j)^2)^T \boldsymbol{D}_{\boldsymbol{K}(\boldsymbol{x},\boldsymbol{X}^T)}\right] - \frac{p}{\sigma^2}\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{X}^T). \tag{A.6}$$

Bibliography

[1] J. Abonyi. Incorporating prior knowledge in fuzzy model identification. *International Journal of Systems Science*, 31(5):657–667, 2000.

[2] Y. S. Abu-Mostafa. Learning from hints in neural networks. *Journal of Complexity*, 6(2):192–198, 1990.

[3] Y. S. Abu-Mostafa. Learning from hints. *Journal of Complexity*, 10(1):165–178, 1994.

[4] C. D. Aliprantis, D. Harris, and R. Tourky. Riesz estimators. *Journal of Econometrics*, 136(2):431–456, 2007.

[5] J. L. An, Z. O. Wang, Q. X. Yang, Z. P. Ma, and C. J. Gao. Study on method of on-line identification for complex nonlinear dynamic system based on SVM. In *Proc. of the Int. Conf. on Machine Learning and Cybernetics (ICMLC), Guangzhou, China*, volume 3, pages 1654–1659, 2005.

[6] P. Andersson. Adaptive forgetting in recursive identification through multiple models. *International Journal of Control*, 42(5):1175–1193, 1985.

[7] R. Andrews and S. Geva. On the effects of initializing a neural network with prior knowledge. In *Proc. of the Int. Conf. on Neural Information Processing (ICONIP), Perth, Australia*, pages 251–256, 1999.

[8] A. Ang and G. Bakaert. Regime switches in interest rates. *Journal of Business & Economic Statistics*, 20(2):163–182, 2002.

[9] F. R. Bach and M. I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3(1):1–48, 2003.

[10] E. W. Bai. A blind approach to the Hammerstein–Wiener model identification. *Automatica*, 38(6):967–979, 2002.

[11] E. W. Bai and S. Sastry. Discrete-time adaptive control utilizing prior information. *IEEE Trans. on Automatic Control*, 31(8):779–782, 1986.

[12] L. Bako, G. Mercère, and S. Lecoeuche. On-line structured identification of switching systems with possibly varying orders. In *European Control Conference (ECC), Kos, Greece*, 2007.

[13] L. Bako and R. Vidal. Identification of switched MIMO ARX models. In *Proc. of the 11th Int. Conf. on Hybrid Systems: Computation and Control (HSCC), St. Louis, MO, USA*, volume 4981 of *LNCS*, pages 43–57, 2008.

[14] M. Basseville and I. V. Nikiforov. *Detection of abrupt changes: theory and application.* Prentice Hall, Englewood Cliffs, NJ, USA, 1993.

[15] A. Bemporad, A. Garulli, S. Paoletti, and A. Vicino. A bounded-error approach to piecewise affine system identification. *IEEE Trans. on Automatic Control*, 50(10):1567–1580, 2005.

[16] A. Bemporad and N. Giorgetti. Website of the IEEE CSS technical committee on hybrid systems. http://www.dii.unisi.it/hybrid/ieee/.

[17] A. Bemporad, W. Heemels, and B. De Schutter. On hybrid systems and closed-loop MPC systems. *IEEE Trans. Automatic Control*, 47(5):863–869, 2002.

[18] Y. Bengio. On the challenge of learning complex functions. *Progress in Brain Research*, 165:521–534, 2007.

[19] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and Q. Montreal. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems (NIPS 2006)*, volume 19, pages 153–160, 2007.

[20] Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*. MIT Press, Cambridge, MA, USA, 2007.

[21] K. P. Bennett. Combining support vector and mathematical programming methods for classification. In B. Schölkopf, C. J.C. Burges, and A. J. Smola, editors, *Advances in kernel methods: support vector learning*, pages 307–326. MIT Press, Cambridge, MA, USA, 1999.

[22] K. P. Bennett and O. L. Mangasarian. Multicategory discrimination via linear programming. *Optimization Methods and Software*, 3(1):27–39, 1994.

[23] J. Bi and T. Zhang. Support vector classification with input data uncertainty. In *Advances in Neural Information Processing Systems (NIPS 2004)*, volume 17, pages 161–168, 2005.

[24] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[25] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[26] G. Bloch, F. Lauer, G. Colin, and Y. Chamaillard. Combining experimental data and physical simulation models in support vector learning. In *Proc. of the 10th Int. Conf. on Engineering Applications of Neural Networks (EANN), Thessaloniki, Greece*, volume 284 of *CEUS Workshop Proceedings*, pages 284–295, 2007.

[27] G. Bloch, F. Lauer, G. Colin, and Y. Chamaillard. Support vector regression from simulation data and few experimental samples. *Information Sciences*, 178(20):3813–3827, 2008.

[28] F. Borrelli, M. Baotić, A. Bemporad, and M. Morari. Dynamic programming for constrained optimal control of discrete-time linear hybrid systems. *Automatica*, 41(10):1709–1721, 2005.

[29] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proc. of the 5th Annual ACM Workshop on Computational Learning Theory (COLT), Pittsburgh, PA, USA*, pages 144–152, 1992.

[30] L. Bottou and C.-J. Lin. Support vector machine solvers. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007.

[31] E. Bredensteiner and K. Bennett. Multicategory classification by support vector machines. *Computational Optimization and Applications*, 12(1-3):35–46, 1999.

[32] L. Breiman. Hinging hyperplanes for regression, classification, and function approximation. *IEEE Trans. on Information Theory*, 39(3):999–1013, 1993.

[33] M. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. W. Sugnet, T. S. Furey, M. Ares Jr, and D. Haussler. Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proc. of the National Academy of Sciences*, 97(1):262–267, 2000.

[34] M. Brown and C. Harris. *Neurofuzzy Adaptive Modelling and Control*. Prentice Hall International, Hertfordshire, UK, 1994.

[35] S. Canu, Y. Grandvalet, V. Guigue, and A. Rakotomamonjy. SVM and kernel methods matlab toolbox. Perception Systèmes et Information, INSA de Rouen, Rouen, France, 2005.

[36] A. Chalimourda, B. Schölkopf, and A. J. Smola. Experimentally optimal ν in support vector regression for different noise models and parameter settings. *Neural Networks*, 17(1):127–141, 2004.

[37] A. B Chan and N. Vasconcelos. Probabilistic kernels for the classification of auto-regressive visual processes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), San Diego, CA, USA*, pages 846–851, 2005.

[38] K. S. Chan and H. Tong. On estimating thresholds in autoregressive models. *Journal of Time Series Analysis*, 7(3):179–194, 1986.

[39] W. C. Chan, C. W. Chan, K. C. Cheung, and C. J. Harris. Modelling of nonlinear dynamical systems using support vector neural networks. *Engineering Applications of Artificial Intelligence*, 14:105–113, 2001.

[40] C. Chang and C. Lin. LibSVM: a library for support vector machines, 2001. http://www.csie.ntu.edu.tw/∼cjlin/libsvm/.

[41] O. Chapelle. Training a support vector machine in the primal. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines.* MIT Press, 2007.

[42] O. Chapelle, P. Haffner, and V. N. Vapnik. Support vector machines for histogram-based image classification. *IEEE Trans. on Neural Networks*, 10(5):1055–1064, 1999.

[43] O. Chapelle and B. Schölkopf. Incorporating invariances in non-linear support vector machines. In *Advances in Neural Information Processing Systems (NIPS 2001)*, volume 14, pages 609–616, 2001.

[44] O. Chapelle, V. N. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002.

[45] S. Chen and S. A. Billings. Neural networks for nonlinear dynamic system modelling and identification. *International Journal of Control*, 56(2):319–346, 1992.

[46] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Trans. on Neural Networks*, 2(2):302–309, 1991.

[47] S. Chen, X. Hong, C. J. Harris, and X. Wang. Identification of nonlinear systems using generalized kernel models. *IEEE Trans. on Control Systems Technology*, 13(3):401–411, 2005.

[48] V. Cherkassky and X. Shao. Signal estimation and denoising using VC-theory. *Neural Networks*, 14(1):37–52, 2001.

[49] T. L. Chia, P. C. Chow, and H. J. Chizeck. Recursive parameter identification of constrained systems: an application to electrically stimulated muscle. *IEEE Trans. on Biomedical Engineering*, 38(5):429–442, 1991.

[50] W. Chu, S. S. Keerthi, and C. J. Ong. Bayesian support vector regression using a unified loss function. *IEEE Trans. on Neural Networks*, 15(1):29–44, 2004.

[51] L. O. Chua and A. C. Deng. Canonical piecewise-linear representation. *IEEE Trans. Circuits and Systems*, 35(1):101–111, 1988.

[52] L. O. Chua and S. M. Kang. Section-wise piecewise-linear functions: Canonical representation, properties and applications. *Proceedings of the IEEE*, 65(6):915–929, 1977.

[53] E. Cinquemani, R. Porreca, G. Ferrari-Trecate, and J. Lygeros. A general framework for the identification of jump Markov linear systems. *Proc. of the 46th IEEE Conf. on Decision and Control (CDC), New Orleans, LA, USA*, pages 5737–5742, 2007.

[54] E. Cinquemani, R. Porreca, G. Ferrari-Trecate, and J. Lygeros. Subtilin production by bacillus subtilis: Stochastic hybrid models and parameter identification. *IEEE Trans. on Automatic Control*, 53(1):38–50, 2008.

[55] G. Colin, Y. Chamaillard, G. Bloch, and G. Corde. Neural control of fast nonlinear systems – Application to a turbocharged SI engine with VCT. *IEEE Trans. on Neural Networks*, 18(4):1101–1114, 2007.

[56] C. Cortes, P. Haffner, and M. Mohri. Positive definite rational kernels. In *Proc. of the 16th Annual Conf. on Computational Learning Theory (COLT), Washington, DC, USA*, volume 2777 of *LNCS*, pages 41–56, 2003.

[57] C. Cortes, P. Haffner, and M. Mohri. Weighted automata kernels – general framework and algorithms. In *Proc. of the 9th European Conf. on Speech Communication and Technology (Eurospeech 03), Special Session Advanced Machine Learning Algorithms for Speech and Language Processing, Geneva, Switzerland*, 2003.

[58] C. Cosentino, W. Curatola, M. Bansal, D. Di Bernardo, and F. Amato. Piecewise affine approach to inferring cell cycle regulatory network in fission yeast. *Biomedical Signal Processing and Control*, 2(3):208–216, 2007.

[59] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Trans. on Electronic Computers*, 14(3):326–334, 1965.

[60] D. Cox and F. O'Sullivan. Asymptotic analysis of penalized likelihood and related estimators. *The Annals of Statistics*, 18(4):1676–1695, 1990.

[61] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2(2):265–292, 2001.

[62] N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[63] S. Dasgupta, B. D. O. Anderson, and R. J. Kaye. Output-error identification methods for partially known systems. *International Journal of Control*, 43(1):177–192, 1986.

[64] B. De Schutter and T. J. J. Van Den Boom. MPC for continuous piecewise-affine systems. *Systems & Control Letters*, 52(3-4):179–192, 2004.

[65] D. DeCoste and M. Burl. Distortion-invariant recognition via jittered queries. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Hilton Head Island, SC, USA*, pages 732–737, 2000.

[66] D. Decoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190, 2002.

[67] Z. Ding and L. Hong. An interactive multiple model algorithm with a switching Markov chain. *Mathematical and Computer Modelling*, 25(1):1–9, 1997.

[68] J. X. Dong, A. Krzyzak, and C. Y. Suen. A fast parallel optimization for training support vector machines. In *Proc. of the 3rd Int. Conf. on Machine Learning and Data Mining in Pattern Recognition, Leipzig, Germany*, volume 2734 of *LNCS*, pages 96–105, 2003.

[69] J. X. Dong, A. Krzyzak, and C. Y. Suen. Fast SVM training algorithm with decomposition on very large data sets. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(4):603–618, 2005.

[70] G. Dreyfus. *Neural Networks: Methodology And Applications*. Springer, 2005.

[71] P. M. L. Drezet and R. F. Harrison. Support vector machines for system identification. *Proc. of the UKACC Int. Conf. on Control, Swansea, UK*, 1:688–692, 1998.

[72] S. Drulhe, G. Ferrari-Trecate, and H. de Jong. The switching threshold reconstruction problem for piecewise-affine models of genetic regulatory networks. *IEEE Trans. on Automatic Control*, 53(1):153–165, 2008.

[73] K. Duan, S. S. Keerthi, and A. N. Poo. Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing*, 51(4):41–59, 2003.

[74] K. B. Duan and S. S. Keerthi. Which is the best multiclass SVM method? an empirical study. In *Proc. of the 6th Int. Workshop on Multiple Classifier Systems, Seaside, CA, USA*, volume 3541 of *LNCS*, pages 278–285, 2005.

[75] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley-Interscience, 2000.

[76] E. Eskinat, S. H. Johnson, and W. L. Luyben. Use of Hammerstein models in identification of nonlinear systems. *AIChE Journal*, 37(2):255–268, 1991.

[77] M. Espinoza, J. A. K. Suykens, and B. De Moor. Least squares support vector machines and primal space estimation. *Proc. of the 42nd Conf. on Decision and Control (CDC), Maui, Hawaii, USA*, 4:3451–3456, 2003.

[78] M. Espinoza, J. A. K. Suykens, and B. De Moor. Fixed-size least squares support vector machines: A large scale application in electrical load forecasting. *Computational Management Science*, 3(2):113–129, 2006.

[79] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13:1–50, 2000.

[80] R. E. Fan, P. H. Chen, and C. J. Lin. Working set selection using the second order information for training SVM. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[81] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39(2):205–217, 2003.

[82] A. W. Fitzgibbon and A. Zisserman. Joint manifold distance: a new approach to appearance based clustering. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), , Madison, WI, USA*, volume 1, pages 26–33, 2003.

[83] J. W. Fox, W. K. Cheng, and J. B. Heywood. A model for predicting residual gas fraction in spark-ignition engines. *SAE Technical Papers*, (931025), 1993.

[84] F. Friedrichs and C. Igel. Evolutionary tuning of multiple SVM parameters. In *Proc. of the 12th Europ. Symp. on Artificial Neural Networks (ESANN)*, pages 519–524, Bruges, Belgium, 2004.

[85] G. Fung, O. L. Mangasarian, and J. W. Shavlik. Knowledge-based nonlinear kernel classifiers. In *Proc. of the 16th Annual Conf. on Learning Theory (COLT), Washington, DC, USA*, volume 2777 of *LNCS*, pages 102–113, 2003.

[86] G. Fung, O. L. Mangasarian, and J. W. Shavlik. Knowledge-based support vector machine classifiers. In *Advances in Neural Information Processing Systems (NIPS 2002)*, volume 15, pages 521–528, 2003.

[87] J. Gao and D. Shi. Sparse kernel regression modelling based on l1 significant vector learning. In *Proc. of the Int. Conf. on Neural Networks and Brain*, volume 3, pages 1925–1930, 2005.

[88] J. B. Gao, S. R. Gunn, C. J. Harris, and M. Brown. A probabilistic framework for SVM regression and error bar estimation. *Machine Learning*, 46(1):71–89, 2002.

[89] H. Garnier, M. Mensler, and A. Richard. Continuous-time model identification from sampled data: implementation issues and performance evaluation. *International Journal of Control*, 76(13):1337–1357, 2003.

[90] H. Garnier and L. Wang, editors. *Identification of Continuous-Time Models from Sampled Data.* Advances in Industrial Control. Springer Verlag, 2008.

[91] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proc. of the 16th Annual Conf. on Computational Learning Theory (COLT), Washington, DC, USA*, volume 2777 of *LNCS*, pages 129–143, 2003.

[92] K. Gasso, G. Mourot, and J. Ragot. Structure identification in multiple model representation: elimination and merging of local models. In *Proc. of the 40th IEEE Conf. on Decision and Control (CDC), Orlando, FL, USA*, volume 3, pages 2992–2997, 2001.

[93] Z. Ghahramani and G.E. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):831–864, 2000.

[94] P. Giansetti, G. Colin, P. Higelin, and Y. Chamaillard. Residual gas fraction measurement and computation. *International Journal of Engine Research*, 8(4):347–364, 2007.

[95] F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

[96] M. T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. *Discrete and Computational Geometry*, 14(1):445–462, 1995.

[97] G. C. Goodwin and R. L. Payne. *Dynamic system identification: experiment design and data analysis.* Academic Press, 1977.

[98] T. Graepel and R. Herbrich. Invariant pattern recognition by semi-definite programming machines. In *Advances in Neural Information Processing Systems (NIPS 2003)*, volume 16, pages 33–40, 2004.

[99] C. W. J. Granger and T. Teräsvirta. *Modelling Nonlinear Economic Relationships.* Oxford University Press, USA, 1993.

[100] S. F. Gray. Modeling the conditional distribution of interest rates as a regime-switching process. *Journal of Financial Economics*, 42(1):27–62, 1996.

[101] R. E. Groff. *Piecewise Linear Homeomorphisms for Approximation of Invertible Maps.* PhD thesis, The University of Michigan, 2003.

[102] R. E. Groff, D. E. Koditschek, and P. P. Khargonekar. A local convergence proof for the minvar algorithm for computing continuous piecewise linear approximations. *SIAM Journal on Numerical Analysis*, 41(3):983–1007, 2003.

[103] Y. Guermeur, A. Elisseeff, and H. Paugam-Moisy. A new multi-class SVM based on a uniform convergence result. In *Proc. of the Int. Joint Conf. on Neural Networks (IJCNN), Como, Italy*, volume 4, pages 183–188, 2000.

[104] S. R. Gunn. Support vector machines for classification and regression. Technical report, Information: Signals, Images, Systems (ISIS) Research Group, University of Southampton, 1998.

[105] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[106] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature Extraction: Foundations and Applications*, volume 207 of *Studies in Fuzziness and Soft Computing.* Springer, 2006.

[107] B. Haasdonk and H. Burkhardt. Invariant kernels for pattern analysis and machine learning. Technical Report 3/05, IIF-LMB, Computer Science Department, University of Freiburg, 2005.

[108] B. Haasdonk and H. Burkhardt. Invariant kernel functions for pattern analysis and machine learning. *Machine Learning*, 68(1):35–61, 2007.

[109] B. Haasdonk and D. Keysers. Tangent distance kernels for support vector machines. In *Proc. of the 16th Int. Conf. on Pattern Recognition (ICPR), Québec, QC, Canada*, volume 2, pages 864–868, 2002.

[110] B. Haasdonk, A. Vossen, and H. Burkhardt. Invariance in kernel methods by Haar-integration kernels. In *Scandinavian Conf. on Image Analysis*, volume 3540 of *LNCS*, pages 841–851, 2005.

[111] J. D. Hamilton. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica*, 57(2):357–384, 1989.

[112] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415, 2004.

[113] T. Hastie, R. Tibshirani, J. Friedman, et al. *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2001.

[114] R. J. Hathaway and J. C. Bezdek. Switching regression models and fuzzy clustering. *IEEE Trans. on Fuzzy Systems*, 1(3):195–204, 1993.

[115] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 1999.

[116] W. Heemels, B. De Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, 2001.

[117] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[118] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[119] P.J. Huber. *Robust Statistics.* Wiley-Interscience, 2004.

[120] D. J. Hudson. Fitting segmented curves whose join points have to be estimated. *Journal of the American Statistical Association*, 61(316):1097–1129, 1966.

[121] Imagine. Amesim web site. www.amesim.com, 2006.

[122] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems (NIPS 1998)*, volume 11, pages 487–493, 1999.

[123] M. Jankovic, S. Magner, S. Hsieh, and J. Konesol. Transient effects and torque control of engines with variable cam timing. In *Proc. of the American Control Conference (ACC), San Francisco, CA*, volume 1, pages 50–54, 2000.

[124] L. Jaulin and E. Walter. Nonlinear bounded-error parameter estimation using interval computation. In Pedrycz W., editor, *Granular computing: an emerging paradigm*, pages 58–71. Physica-Verlag, 2001.

[125] A. Jayadeva, R. Khemchandani, and S. Chandra. Regularized least squares support vector regression for the simultaneous learning of a function and its derivatives. *Information Sciences (in press)*, 2008.

[126] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proc. of the 10th Eur. Conf. on Machine Learning (ECML), Chemnitz, Germany*, volume 1398 of *LNCS*, pages 137–142, 1998.

[127] T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in kernel methods: support vector learning*, pages 169–184. MIT Press, Cambridge, MA, USA, 1999.

[128] T. Joachims. Transductive inference for text classification using support vector machines. In *Proc. of the 6th Int Conf. on Machine Learning (ICML), Bled, Slovenia*, pages 200–209, 1999.

[129] T. Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms.* Kluwer Academic Publishers, 2002.

[130] T. A. Johansen. *Operating Regime based Process Modeling and Identification.* PhD thesis, University of Trondheim, 1994.

[131] T. A. Johansen. Identification of non-linear systems using empirical data and prior knowledge-an optimization approach. *Automatica*, 32(3):337–356, 1996.

[132] T. A. Johansen and B. A. Foss. Constructing narmax models using armax models. *International Journal of Control*, 58(5):1125–1153, 1993.

[133] T. A. Johansen and B. A. Foss. Identification of non-linear system structure and parameters using regime decomposition. *Automatica*, 31(2):321–326, 1995.

[134] P. Julian, A. Desages, and O. Agamennoni. High-level canonical piecewise linear representation using asimplicial partition. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 46(4):463–480, 1999.

[135] A. L. Juloski, W. Heemels, G. Ferrari-Trecate, R. Vidal, S. Paoletti, and J. H. G. Niessen. Comparison of four procedures for the identification of hybrid systems. In *Proc. of the 8th Int. Conf. on Hybrid Systems: Computation and Control (HSCC), Zurich, Switzerland*, volume 3414 of *LNCS*, pages 354–369, 2005.

[136] A. L. Juloski and S. Weiland. A Bayesian approach to the identification of piecewise linear output error models. In *Proc. of the 14th IFAC Symp. on System Identification (SYSID), Newcastle, Australia*, pages 374–379, 2006.

[137] A. L. Juloski, S. Weiland, and W. Heemels. A Bayesian approach to identification of hybrid systems. *IEEE Trans. on Automatic Control*, 50(10):1520–1533, 2005.

[138] S. Kambar. Generating synthetic data by morphing transformation for handwritten numeral recognition (with ν-svm). Master's thesis, Concordia University, Montréal, QC, Canada, 2005.

[139] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proc. of the 20th Int. Conf. on Machine Learning (ICML), Washington, DC, USA*, pages 321–328, 2003.

[140] L. Kaufman. Solving the quadratic programming problem arising in support vector classification. In *Advances in kernel methods: support vector learning*, pages 147–167. MIT Press, Cambridge, MA, USA, 1999.

[141] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637–649, 2001.

[142] M. Khammash, C. J. Tomlin, and M. Vidyasagar. Guest editorial special issue on systems biology. *IEEE Trans. on Automatic Control*, 53(1):4–7, 2008.

[143] G. S. Kimeldorf and G. Wahba. A correspondence between bayesian estimation of stochastic processes and smoothing by splines. *Annals of Mathematical Statistics*, 41(2):495–502, 1971.

[144] R. I. Kondor and T. Jebara. A kernel between sets of vectors. In *Proc. of the 20th Int. Conf. on Machine Learning (ICML), Washington, DC, USA*, pages 361–368, 2003.

[145] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proc. of the 19th Int. Conf. on Machine Learning (ICML), Sydney, Australia*, 2002.

[146] J. Korbicz, J. M. Kościelny, Z. Kowalczuk, and W. Cholewa, editors. *Fault Diagnosis: Models, Artificial Intelligence, Applications.* Springer, 2004.

[147] F. Lauer, M. Bentoumi, G. Bloch, G. Millerioux, and P. Aknin. Ho-Kashyap with early stopping versus soft margin SVM for linear classifiers - an application. In *Advances in Neural Networks - ISNN 2004, Proc. of the Int. Symp. on Neural Networks, Dalian, China*, volume 3173 of *LNCS*, pages 524–530, 2004.

[148] F. Lauer and G. Bloch. Méthodes SVM pour l'identification. In *Journées Identification et Modélisation Expérimentale (JIME), Poitiers, France*, 2006.

[149] F. Lauer and G. Bloch. Incorporating prior knowledge in support vector machines for classification: a review. *Neurocomputing*, 71(7-9):1578–1594, 2008.

[150] F. Lauer and G. Bloch. Incorporating prior knowledge in support vector regression. *Machine Learning*, 70(1):89–118, 2008.

[151] F. Lauer and G. Bloch. A new hybrid system identification algorithm with automatic tuning. In *Proc. of the 17th IFAC World Congress, Seoul, Korea*, pages 10207–10212, 2008.

[152] F. Lauer and G. Bloch. Switched and piecewise nonlinear hybrid system identification. In *Proc. of the 11th Int. Conf. on Hybrid Systems: Computation and Control (HSCC), St. Louis, MO, USA*, volume 4981 of *LNCS*, pages 330–343, 2008.

[153] F. Lauer, C. Y. Suen, and G. Bloch. A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40:1816–1824, 2007.

[154] M. H. Law and J. T. Kwok. Bayesian support vector regression. *Proc. of the 8th Int. Workshop on Artificial Intelligence and Statistics (AISTATS), Key West, FL, USA*, pages 239–244, 2001.

[155] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems.* Classics in Applied Mathematics. SIAM, 1995.

[156] M. Lazar, W. Heemels, S. Weiland, and A. Bemporad. Stabilizing model predictive control of hybrid systems. *IEEE Trans. on Automatic Control*, 51(11):1813–1818, 2006.

[157] M. Lázaro, F. Pérez-Cruz, and A. Artés-Rodriguez. Learning a function and its derivative forcing the support vector expansion. *IEEE Signal Processing Letters*, 12:194–197, 2005.

[158] M. Lázaro, I. Santamaria, F. Pérez-Cruz, and A. Artés-Rodriguez. Support vector regression for the simultaneous learning of a multivariate function and its derivatives. *Neurocomputing*, 69:42–61, 2005.

[159] Y. LeCun. Who is afraid of non-convex loss functions? Invited Talk, NIPS Workshop on Efficient Learning, Vancouver, Canada, 2007. Available at http://videolectures.net/eml07_lecun_wia/.

[160] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[161] Y. LeCun and C. Cortes. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/index.html.

[162] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In *Proc. of the Int. Conf. on Artificial Neural Networks (ICANN), Paris, France*, pages 53–60, 1995.

[163] K. L. Lee. Time series prediction using support vector machines, the orthogonal and the regularized orthogonal least-squares algorithms. *Int. Journal of Systems Science*, 33(10):811–821, 2002.

[164] Y.-J. Lee and O. L. Mangasarian. SSVM: A smooth support vector machine. *Computational Optimization and Applications*, 20(1):5–22, 2001.

[165] T. K. Leen. From data distributions to regularization in invariant learning. *Neural Computation*, 7(5):974–981, 1995.

[166] Y. Leung, J. H. Ma, and W. X. Zhang. A new method for mining regression classes in large data sets. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(1):5–21, 2001.

[167] H. L. Li and J. R. Yu. A piecewise regression analysis with automatic change-point detection. *Intelligent Data Analysis*, 3(1):75–85, 1999.

[168] P. Lindskog. *Methods, Algotithms and Tools for System Identification Based on Prior Knowledge.* PhD thesis, Department of Electrical Engineering, Linköping University, 1996.

[169] L. Ljung. *System identifcation: Theory for the user.* Prentice-Hall Inc., 2nd edition edition, 1999.

[170] G. Loosli, S. Canu, S. V. N. Vishwanathan, and A. J. Smola. Invariances in classification: an efficient SVM implementation. In *Int. Symp. on Applied Stochastic Models and Data Analysis (ASMDA)*, 2005.

[171] G. Loosli, S. Canu, S. V. N. Vishwanathan, A. J. Smola, and M. Chattopadhyay. Boîte à outils SVM simple et rapide. *Revue d'intelligence artificielle*, 19(4-5):741–767, 2005.

[172] G. Loosli, G. Gasso, and S. Canu. Regularization paths for nu-SVM and nu-SVR. In *Advances in Neural Networks - ISNN, Proc. of the 4th Int. Symp. on Neural Networks, Nanjing, China*, volume 4493 of *LNCS*, pages 486–496, 2007.

[173] M. Lubrano. Bayesian analysis of switching regression models. *Journal of Econometrics*, 29(1-2):69–95, 1985.

[174] M. Lucea. *Modélisation dynamique par réseaux de neurones et machines à vecteurs supports : contribution à la maîtrise des émissions polluantes de véhicules automobiles.* PhD thesis, Université Paris 6, 2006.

[175] J. Ma, J. Theiler, and S. Perkins. Accurate on-line support vector regression. *Neural Computation*, 15(11):2683–2703, 2003.

[176] Y. Ma and R. Vidal. Identification of deterministic switched ARX systems via identification of algebraic varieties. In *Proc. of the 8th Int. Conf. on Hybrid Systems: Computation and Control (HSCC), Zurich, Switzerland*, volume 3414 of *LNCS*, pages 449–465, 2005.

[177] R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proc. of the 20th National Conf. on Artificial Intelligence, Pittsburgh, PA, US*, pages 819–824, 2005.

[178] A. Magnani and S. Boyd. Convex piecewise-linear fitting. *Optimization and Engineering*, to appear, 2008.

[179] O. L. Mangasarian. Generalized support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 135–146. MIT Press, Cambridge, MA, USA, 2000.

[180] O. L. Mangasarian and D. R. Musicant. Lagrangian support vector machines. *Journal of Machine Learning Research*, 1:161–177, 2001.

[181] O. L. Mangasarian and D. R. Musicant. Large scale kernel regression via linear programming. *Machine Learning*, 46(1-3):255–269, 2002.

[182] O. L. Mangasarian, Jude W. Shavlik, and Edward W. Wild. Knowledge-based kernel approximation. *Journal of Machine Learning Research*, 5:1127–1141, 2004.

[183] O. L. Mangasarian and E. W. Wild. Nonlinear knowledge in kernel approximation. Technical Report 05-05, Data Mining Institute, University of Wisconsin, 2005.

[184] O. L. Mangasarian and E. W. Wild. Nonlinear knowledge in kernel approximation. *IEEE Trans. on Neural Networks*, 18(1):300–306, 2007.

[185] M. Martin. On-line support vector machine regression. In *Proc. of the 13th European Conf. on Machine Learning (ECML), Helsinki, Finland*, volume 2430 of *LNCS*, pages 282–294, 2002.

[186] D. Mattera and S. Haykin. Support vector machines for dynamic reconstruction of a chaotic system. In B. Schölkopf, C. J.C. Burges, and A. J. Smola, editors, *Advances in kernel methods: support vector learning*, pages 211–241. MIT Press, Cambridge, MA, USA, 1999.

[187] V. E. McGee and W. T. Carleton. Piecewise regression. *Journal of the American Statistical Association*, 65(331):1109–1124, 1970.

[188] M. C. Medeiros, A. Veiga, and M. G. C. Resende. A combinatorial approach to piecewise linear time series analysis. *Journal of Computational and Graphical Statistics*, 11(1):236–258, 2002.

[189] C.A. Micchelli and F.I. Utreras. Smoothing and interpolation in a convex subset of a hilbert space. *SIAM Journal on Scientific and Statistical Computing*, 9:728, 1988.

[190] L. Mihaylova, V. Lampaert, H. Bruyninckx, and J. Swevers. Identification of hysteresis functions using a multiple model approach. In *Proc. of the IEEE Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems*, pages 153–158, 2001.

[191] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In *Proc. of the IEEE Conf. on Neural Networks for Signal Processing IX*, pages 41–48, 1999.

[192] K. R. Müller, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Trans. on Neural Networks*, 12(2):181–201, 2001.

[193] K. R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *Proc. of the Int. Conf. on Artificial Neural Networks (ICANN), Lausanne, Switzerland*, volume 1327 of *LNCS*, pages 999–1004, 1997.

[194] K. R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Using support vector machines for time series prediction. In B. Schölkopf, C. J.C. Burges, and A. J. Smola, editors, *Advances in kernel methods: support vector learning*, pages 243–253. MIT Press, Cambridge, MA, USA, 1999.

[195] H. Nakada, K. Takaba, and T. Katayama. Identification of piecewise affine systems based on statistical clustering technique. *Automatica*, 41(5):905–913, 2005.

[196] K. Narendra and P. Gallman. An iterative method for the identification of nonlinear systems using a Hammerstein model. *IEEE Trans. on Automatic Control*, 11(3):546–550, 1966.

[197] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. on Neural Networks*, 1(1):4–27, 1990.

[198] O. Nelles. *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models.* Springer-Verlag, Berlin, Germany, 2001.

[199] E. G. Nepomuceno, R. H. C. Takahashi, G. F. V. Amaral, and L. A. Aguirre. Nonlinear identification using prior knowledge of fixed points: a multiobjective approach. *International Journal of Bifurcation and Chaos*, 13(5):1229–1246, 2003.

[200] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica*, 13:271–369, 2004.

[201] P. Niyogi, F. Girosi, and T. Poggio. Incorporating prior information in machine learning by creating virtual examples. *Proceedings of the IEEE*, 86(11):2196–2209, 1998.

[202] M. Ostendorf, V. V. Digalakis, and O. Kimball. From HMM's to segment models: a unified view of stochastic modeling for speech recognition. *IEEE Trans. on Speech and Audio Processing*, 4(5):360–378, 1996.

[203] E. Osuna, R. Freund, and F Girosi. An improved training algorithm for support vector machines. In *Proc. of the IEEE Workshop on Neural Networks for Signal Processing, Amelia Island, FL, USA*, pages 276–285, 1997.

[204] S. Paoletti, A. L. Juloski, G. Ferrari-Trecate, and R. Vidal. Identification of hybrid systems: a tutorial. *European Journal of Control*, 13(2-3):242–262, 2007.

[205] J. Pittman and C. A. Murthy. Fitting optimal piecewise linear functions using genetic algorithms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(7):701–718, 2000.

[206] J. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, Cambridge, MA, USA, 1999.

[207] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J.C. Burges, and A. J. Smola, editors, *Advances in kernel methods: support vector learning*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[208] T. Poggio and T. Vetter. Recognition and structure from one 2D model view: Observations on prototypes, object classes and symmetries. Technical Report AIM-1347, Massachusetts Institute of Technology, Cambridge, MA, USA, 1992.

[209] M. Pontil and A. Verri. Support vector machines for 3D object recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(6):637–646, 1998.

[210] R. Porreca, G. Ferrari-Trecate, D. Chieppi, L. Magni, and O. Bernard. Switch detection in genetic regulatory networks. In *Proc. of the 10th Int. Conf. on Hybrid Systems: Computation and Control (HSCC), Pisa, Italy*, volume 4416 of *LNCS*, pages 754–757, 2007.

[211] A. Pozdnoukhov and S. Bengio. Tangent vector kernels for invariant image classification with SVMs. In *Proc. of the 17th Int. Conf. on Pattern Recognition (ICPR), Cambridge, UK*, pages 486–489, 2004.

[212] A. Pozdnoukhov and S. Bengio. Invariances in kernel methods: From samples to objects. *Pattern Recognition Letters*, 27(10):1087–1097, 2006.

[213] D. C. Psichogios and L. H. Ungar. A hybrid neural network-first principles approach to process modeling. *AIChE Journal*, 38(10):1499–1511, 1992.

[214] P. Pucar and J. Sjöberg. On the hinge-finding algorithm for hingeing hyperplanes. *IEEE Trans. on Information Theory*, 44(3):1310–1319, 1998.

[215] R. E. Quandt. The estimation of the parameters of a linear regression system obeying two separate regimes. *Journal of the American Statistical Association*, 53(284):873–880, 1958.

[216] J. Ragot, G. Mourot, and D. Maquin. Parameter estimation of switching piecewise linear system. In *Proc. of the 42nd IEEE Conf. on Decision and Control (CDC), Maui, Hawaï, USA*, volume 6, pages 5783–5788, 2003.

[217] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. *Advances in Neural Information Processing Systems (NIPS 2006)*, 17:1137–1144, 2007.

[218] G. P. Rao and H. Unbehauen. Identification of continuous-time systems. *IEE Proceedings – Control Theory and Applications*, 153(2):185–220, 2006.

[219] J. Roll. *Local and piecewise affine approaches to system identification.* PhD thesis, Department of Electrical Engineering, Linköping University, Sweden, 2003.

[220] J. Roll, A. Bemporad, and L. Ljung. Identification of piecewise affine systems via mixed-integer programming. *Automatica*, 40(1):37–50, 2004.

[221] O. Ronneberger. LibsvmTL: a support vector machine template library (2004), 2004. http://lmb.informatik.uni-freiburg.de/lmbsoft/libsvmtl/index.en.html.

[222] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection.* Wiley-Interscience, 2003.

[223] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[224] M. Russell. A segmental HMM for speech pattern matching. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), Minneapolis, MN, USA*, pages 499–502, 1993.

[225] M. Sánchez-Fernández, M. De Prado-Cumplido, J. Arenas-García, and F. Pérez-Cruz. SVM multiregression for nonlinear channel estimation in multiple-input multiple-output systems. *IEEE Trans. on Signal Processing*, 52(8):2298–2307, 2004.

[226] C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. *Proc. of the 15th Int. Conf. on Machine Learning (ICML), Madison, WI, USA*, pages 515–521, 1998.

[227] B. Schölkopf. *Support Vector Learning.* PhD thesis, TU Berlin, R. Oldenbourg Verlag, München, 1997.

[228] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In *Proc. of the Int. Conf. on Artificial Neural Networks (ICANN), Bochum, Germany*, volume 1112 of *LNCS*, pages 47–52, 1996.

[229] B. Schölkopf, R. Herbrich, and A.J. Smola. A generalized representer theorem. In *Proc. of the Annual Conf. on Computational Learning Theory (COLT), Amsterdam, The Netherlands*, pages 416–426. Springer, 2001.

[230] B. Schölkopf, P. Simard, A. J. Smola, and V. Vapnik. Prior knowledge in support vector kernels. In *Advances in Neural Information Processing Systems (NIPS 1997)*, volume 10, pages 640–646, 1998.

[231] B. Schölkopf, A. Smola, and K.R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

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

[233] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

[234] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. In *Advances in Neural Information Processing Systems (NIPS 1999)*, volume 12, pages 582–588, 2000.

[235] H. Schulz-Mirbach. Constructing invariant features by averaging techniques. In *Proc. of the 12th Int. Conf on Pattern Recognition (ICPR), Jerusalem, Israel*, volume 2, pages 387–390, 1994.

[236] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[237] P. K. Shivaswamy, C. Bhattacharyya, and A. J. Smola. Second order cone programming approaches for handling missing and uncertain data. *Journal of Machine Learning Research*, 7:1283–1314, 2006.

[238] P. K. Shivaswamy and T. Jebara. Permutation invariant SVMs. In *Proc. of the 23th Int. Conf. on Machine Learning (ICML), Pittsburgh, PA, USA*, 2006.

[239] P. Simard, Y. LeCun, J. Denker, and B. Victorri. Transformation invariance in pattern recognition, tangent distance and tangent propagation. In *Neural Networks: Tricks of the trade*, volume 1524 of *LNCS*, pages 239–274, 1998.

[240] P. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proc. of the 7th Int. Conf. on Document Analysis and Recognition (ICDAR), Edinburgh, Scotland, UK*, volume 2, pages 958–962, 2003.

[241] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.Y. Glorennec, H. Hjalmarsson, and A. Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12):1691–1724, 1995.

[242] A. Skeppstedt, L. Ljung, and M. Millnert. Construction of composite models from observed data. *International Journal of Control*, 55(1):141–152, 1992.

[243] A. J. Smola, T. Friess, and B. Schölkopf. Semiparametric support vector and linear programming machines. In *Advances in Neural Information Processing Systems (NIPS 1998)*, volume 11, pages 585–591. MIT Press, 1999.

[244] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[245] A. J. Smola, B. Schölkopf, and K. R. Müller. The connection between regularization operators and support vector kernels. *Neural Networks*, 11(4):637–649, 1998.

[246] A. J. Smola, B. Schölkopf, and G. Rätsch. Linear programs for automatic accuracy control in regression. In *Proc. of the 9th Int. Conf. on Artificial Neural Networks (ICANN), Edinburgh, UK*, volume 2, pages 575–580, 1999.

[247] M. O. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins, and J. Weston. Support vector regression with ANOVA decomposition kernels. In B. Schölkopf, C. J.C. Burges, and A. J. Smola, editors, *Advances in kernel methods: Support vector learning*, pages 285–291. MIT Press, Cambridge, MA, USA, 1999.

[248] C. Y. Suen and J. Tan. Analysis of errors of handwritten digits made by a multitude of classifiers. *Pattern Recognition Letters*, 26(3):369–379, 2005.

[249] J. A. K. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle. Weighted least squares support vector machines: robustness and sparse approximation. *Neurocomputing*, 48(1-4):85–105, 2002.

[250] J. A. K. Suykens, L. Lukas, and J. Vandewalle. Sparse approximation using least squares support vector machines. In *Proc. of the IEEE Int. Symp. on Circuits and Systems (ISCAS), Geneva*, volume 2, pages 757–760, 2000.

[251] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, River Edge, NJ, USA, 2002.

[252] J. A. K. Suykens and J. Vandewalle, editors. *Nonlinear Modeling: Advanced Black-Box Techniques*. Kluwer Academic Publishers, 1998.

[253] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[254] J. A. K. Suykens and J. Vandewalle. Multiclass least squares support vector machines. *Proc. of the Int. Joint Conf. on Neural Networks (IJCNN), Washington, DC, USA*, 2:900–903, 1999.

[255] J. A. K. Suykens and J. Vandewalle. Recurrent least squares support vector machines. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 47(7):1109–1114, 2000.

[256] J. A. K. Suykens, J. Vandewalle, and B. De Moor. Optimal control by least squares support vector machines. *Neural Networks*, 14(1):23–35, 2001.

[257] F. E. H. Tay and L. J. Cao. Modified support vector machines in financial time series forecasting. *Neurocomputing*, 48(1):847–861, 2002.

[258] M. L. Thompson and M. A. Kramer. Modeling chemical processes using prior knowledge and neural networks. *AIChE Journal*, 40(8):1328–1340, 1994.

[259] R. Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

[260] W. D. Timmons, H. J. Chizeck, and P. G. Katona. Parameter-constrained adaptive control. *Industrial & Engineering Chemical Research*, 36(11):4894–4905, 1997.

[261] M. Tipping. The relevance vector machine. In *Advances in Neural Information Processing Systems (NIPS 1999)*, volume 12, pages 652–658, 2000.

[262] A. Tishler and I. Zang. A switching regression method using inequality conditions. *Journal of Econometrics*, 11(2):259–274, 1979.

[263] A. Tishler and I. Zang. A new maximum likelihood algorithm for piecewise regression. *Journal of the American Statistical Association*, 76(376):980–987, 1981.

[264] H. Tong. *Non-Linear Time Series: A Dynamical System Approach.* Oxford University Press, 1993.

[265] H. Tong and K. S. Lim. Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 42(3):245–292, 1980.

[266] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2):119–165, 1994.

[267] T. B. Trafalis. Support vector regression with noisy data: a second order cone programming approach. *International Journal of General Systems*, 36(2):237–250, 2007.

[268] T. B. Trafalis and R. C. Gilbert. Robust classification and regression using support vector machines. *European Journal of Operational Research*, 173(3):893–909, 2006.

[269] I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[270] J. Tugnait. Adaptive estimation and identification for discrete systems with markov jump parameters. *IEEE Trans. on Automatic Control*, 27(5):1054–1065, 1982.

[271] H. J. A. F. Tulleken. Grey-box modelling and identification using physical knowledge and bayesian techniques. *Automatica*, 29(2):285–308, 1993.

[272] T. Van Gestel, J. A. K. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle. Benchmarking least squares support vector machine classifiers. *Machine Learning*, 54(1):5–32, 2004.

[273] T. Van Gestel, J. A. K. Suykens, G. Lanckriet, A. Lambrechts, B. De Moor, and J. Vandewalle. Bayesian framework for least-squares support vector machine classifiers, Gaussian processes, and kernel Fisher discriminant analysis. *Neural Computation*, 14(5):1115–1147, 2002.

[274] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[275] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, NY, USA, 1995.

[276] V. N. Vapnik. *Statistical Learning Theory.* John Wiley, New York, NY, USA, 1998.

[277] V. N. Vapnik. Three remarks on the support vector method of function estimation. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning.* MIT Press, Cambridge, MA, USA, 1999.

[278] E. Vazquez and E. Walter. Multi-output support vector regression. In *Proc. of the 13th IFAC Symp. on System Identification (SYSID), Rotterdam, The Netherlands*, pages 1820–1825, 2003.

[279] A. Vedaldi, P. Favaro, and E. Grisan. Boosting invariance and efficiency in supervised learning. In *Proc. of the IEEE 11th Int. Conf. on Computer Vision (ICCV), Rio de Janeiro, Brazil*, pages 1–8, 2007.

[280] V. Verdult and M. Verhaegen. Subspace identification of piecewise linear systems. *Proc. of the 43rd IEEE Conf. on Decision and Control (CDC), Atlantis, Paradise Island, Bahamas*, 4:3838–3843, 2004.

[281] K. Veropoulos, C. Campbell, and N. Cristianini. Controlling the sensitivity of support vector machines. In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, pages 55–60, 1999.

[282] R. Vidal. Identification of spatial-temporal switched ARX systems. *Proc. of the 46th IEEE Conf. on Decision and Control (CDC), New Orleans, LA, USA*, pages 4675–4680, 2007.

[283] R. Vidal. Recursive identification of switched ARX systems. *Automatica*, 44(9):2274–2287, 2008.

[284] R. Vidal and B. D. O. Anderson. Recursive identification of switched ARX hybrid models: exponential convergence and persistence of excitation. In *Proc. of the 43rd IEEE Conf. on Decision and Control (CDC), Atlantis, Paradise Island, Bahamas*, volume 1, pages 32–37, 2004.

[285] R. Vidal, Y. Ma, and S. Sastry. Generalized principal component analysis (GPCA). *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(12):1945–1959, 2005.

[286] R. Vidal, S. Soatto, and A. Chiuso. Applications of hybrid system identification in computer vision. In *Proc. of the European Control Conference (ECC)*, 2007.

[287] R. Vidal, S. Soatto, Y. Ma, and S. Sastry. An algebraic geometric approach to the identification of a class of linear hybrid systems. In *Proc. of the 42nd IEEE Conf. on Decision and Control (CDC), Maui, Hawaï, USA*, pages 167–172, 2003.

[288] M. Villalobos and G. Wahba. Inequality-constrained multivariate smoothing splines with application to the estimation of posterior probabilities. *Journal of the American Statistical Association*, 82(397):239–248, 1987.

[289] S. V. N. Vishwanathan, A. J. Smola, and M. Narasimha Murty. SimpleSVM. In *Proc. of the 20th Int. Conf. on Machine Learning (ICML), Washington, DC, USA*, pages 760–767, 2003.

[290] S. V. N. Vishwanathan, A. J. Smola, and R. Vidal. Binet-Cauchy kernels on dynamical systems and its application to the analysis of dynamic scenes. *International Journal of Computer Vision*, 73(1):95–119, 2007.

[291] E. Walter and H. Piet-Lahanier. Estimation of parameter bounds from bounded-error data: a survey. *Mathematics and Computers in Simulation*, 32(5-6):449–468, 1990.

[292] H. Wang, D. Pi, and Y. Sun. Online SVM regression algorithm-based adaptive inverse control. *Neurocomputing*, 70(4-6):952–959, 2007.

[293] L. Wang, Y. Gao, K. L. Chan, P. Xue, and W.-Y. Yau. Retrieval with knowledge-driven kernel design: An approach to improving SVM-based CBIR with relevance feedback. In *Proc. of the Int. Conf. on Computer Vision (ICCV), Beijing, China*, pages 1355–1362. IEEE Computer Society, 2005.

[294] L. Wang, P. Xue, and K. L. Chan. Incorporating prior knowledge into SVM for image retrieval. In *Proc. of the 17th Int. Conf. on Pattern Recognition (ICPR), Cambridge, UK*, volume 2, pages 981–984, 2004.

[295] X. D. Wang and M. Y. Ye. Nonlinear dynamic system identification using least squares support vector machine regression. In *Proc. of the IEEE Int. Conf. on Machine Learning and Cybernetics (ICMLC), Shanghai, China*, volume 2, pages 941–945, 2004.

[296] X. X. Wang, S. Chen, D. Lowe, and C. J. Harris. Sparse support vector regression based on orthogonal forward selection for the generalised kernel model. *Neurocomputing*, 70(1-3):462–474, 2006.

[297] J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. N. Vapnik. Kernel dependency estimation. *Advances in Neural Information Processing Systems (NIPS 2002)*, 15:873–880, 2003.

[298] J. Weston, A. Elisseeff, G. Bakir, and F. Sinz. Spider: a matlab machine learning library. http://www.kyb.tuebingen.mpg.de/bs/people/spider/.

[299] J. Weston, A. Gammerman, M. O. Stitson, V. N. Vapnik, V. Vovk, and C. Watkins. Support vector density estimation. In B. Schölkopf, C. J.C. Burges, and A. J. Smola, editors, *Advances in kernel methods: support vector learning*, pages 293–305. MIT Press, Cambridge, MA, USA, 1999.

[300] J. Weston and C. Watkins. Multiclass support vector machines. Technical Report CSD-TR-98-04, Royal Holloway, University of London, 1998.

[301] D. Westwick and M. Verhaegen. Identifying MIMO Wiener systems using subspace model identification methods. *Signal Processing*, 52(2):235–258, 1996.

[302] H. White. Connectionist nonparametric regression: multilayer feedforward networks can learn arbitrary mappings. *Neural Networks*, 3(5):535–549, 1990.

[303] T. Wigren. Recursive prediction error identification using the nonlinear Wiener model. *Automatica*, 29(4):1011–1025, 1993.

[304] L. Wolf and A. Shashua. Learning over sets using kernel principal angles. *Journal of Machine Learning Research*, 4(10):913–931, 2003.

[305] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation*, 1(1):67–82, 1997.

[306] X. Wu and R. Srihari. Incorporating prior knowledge with weighted margin support vector machines. In *Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, Seattle, WA, USA*, pages 326–333, 2004.

[307] J. R. Yu, G. H. Tzeng, and H. L. Li. General fuzzy piecewise regression analysis with automatic change-point detection. *Fuzzy Sets and Systems*, 119(2):247–257, 2001.

[308] L. Zhang and Y. Xi. Nonlinear system identification based on an improved support vector regression estimator. In *Advances in Neural Networks, Proc. of the Int. Symp. on Neural Networks (ISNN), Dalian, China*, volume 3173 of *LNCS*, pages 586–591, 2004.

[309] Y. Zheng and Z. Lin. Recursive adaptive algorithms for fast and rapidly time-varying systems. *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, 50(9):602–614, 2003.

[310] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm support vector machines. In *Advances in Neural Information Processing Systems (NIPS 2003)*, volume 16, 2004.

Index