KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kasteelpark Arenberg 10, 3001 Leuven (Heverlee)

Kernel Models for Large Scale Applications

Promotor:
Prof. dr. ir. B. De Moor

Proefschrift voorgedragen tot het behalen van het doctoraat in de toegepaste wetenschappen door

Bart Hamers

June 2004

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT TOEGEPASTE WETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kasteelpark Arenberg 10, 3001 Leuven (Heverlee)

Kernel Models for Large Scale Applications

Jury:
Prof. dr. ir. L. Froyen, voorzitter
Prof. dr. ir. B. De Moor, promotor
Prof. dr. ir. J. Suykens
Prof. dr. ir. M. Verleysen (UCL)
Prof. dr. ir. H. Blockeel
Prof. dr. ir. J. Vandewalle
Prof. dr. ir. S. Vandewalle

Proefschrift voorgedragen tot het behalen van het doctoraat in de toegepaste wetenschappen door

Bart Hamers

U.D.C. 681.3*H2 June 2004

ISBN 90-5682-509-7
UDC 681.3*H2
D/2004/7515/44

Voorwoord

Dit document dient zeker te beginnen met een uitgebreid woord van dank aan alle mensen die bewust en onbewust hebben bijgedragen aan het tot stand komen ervan. Dit werk zou er niet gekomen zijn zonder jullie uitvoerige hulp.

Allereerst dank ik prof. Bart De Moor voor het aanbieden van een plaats in zijn excellente onderzoeksgroep.

Vervolgens wil ik zeker prof. Johan Suykens bedanken die me van begin onder zijn vleugels heeft genomen. Bedankt voor de dagelijkse begeleiding en goede samenwerking in dit boeiend onderzoeksdomein.

Ook dank aan het IWT en aan de KUL onderzoeksfondsen die zorgden voor de nodige financiële ondersteuning.

Verder dank ik de leden van het leescomite en jury. Jullie advies en commentaar hebben zeker bijgedragen kwaliteit van dit werk.

Speciale dank naar mijn bureaugenoten. Jos bedankt voor de vele uren uitleg in de misterieuze wereld van de statistiek. Luc, Kristiaan voor mijn eindeloze vragen over latex. Maar zeker ook voor de fijne sfeer waarin we hebben gewerkt.

Natuurlijk gaat er ook veel dank uit naar alle andere collega's en vrienden binnen sista: Geert, Gert, Pat, Frank, Kristof, Bert, Stein, Yves, Kathleen, Jannick, Mustak, Maarten, Steven,... en de vele die ik nog vergeten ben.

Pat, bedankt dat ik de frustraties van het onderzoeksleven in de wekelijkse aikibudo-training heb mogen kanaliseren. (lees op u uitwerken...)

Een zeer hartelijk woord van dank gaat natuurlijk ook uit naar mijn ouders. Bedankt voor mij al deze jaren te steunen om deze en vele andere dromen waar te maken.

Een laatste woordje gaat zeker naar Mips. Schat, ook ne dikke bedankt naar u. Voor alle steun en geduld dat je deze tijd hebt moeten bieden. Alle uitgestelde beloftes worden nu zeker waargemaakt...

Korte Inhoud

Als een gevolg van de alsmaar toenemende invloed van de Informatie Technologie in zowel het onderzoek als de bedrijven worden er dagelijks terabytes gegevens verwerkt en opgeslagen. De interesse om deze bron van informatie te gebruiken, groeit gestaag. Het is daar dat gegevensontginning en machineleer-technieken ons een handje helpen. Een recent ontwikkelde techniek in de wereld van gegevensontginning zijn de kernfunctie modellen. Deze modellen blinken uit in diverse probleemsituaties zoals classificatie, regressie en tijdsreeksvoorspelling en dit met betrekking tot mogelijkheid tot veralgemening getest op nieuwe data. Het nadeel van deze methoden is dat hun computationele vereisten voor training kwadratisch scaleren met de grootte van de verzameling van trainingspunten. In dit werk zullen we aantonen hoe dit scaleerbaarheidsprobleem kan worden overwonnen door gebruik te maken van methoden van zowel numerische als leertheorie origine.

De oplossingen, waarmee rekening moet gehouden worden indien men kernfunctie modellen wil gebruiken voor grootschalige toepassingen, worden voorgesteld op basis van vijf kern-ideeën. Deze vijf kern-ideeën zijn: de modelkeuze, de numerieke procedures, lage rang benaderingen en het gebruik van groepsmodellen. Eerst zullen we aantonen dat modellen, die gebruik maken van een kwadratische verliesfunctie, een trainingsprocedure hebben die bestaat uit het oplossen van een lineair stelsel van vergelijkingen. We zullen zien hoe iteratieve oplossingsmethoden en lage rang benaderingen de computationele- en geheugencomplexiteit van het oplossen van het lineaire systeem kunnen beperken. Ook zullen we aantonen dat de kernfunctie zelf een belangrijke rol speelt in zowel de leerperformantie en de computationele- en geheugencomplexiteit van de algoritmen.

Als laatste zullen we het gebruik van groepsmodellen voorstellen voor het trainen van kernfunctie modellen voor grote dataverzamelingen. In plaats van één model te trainen op een trainingsverzameling, zal een collectie van modellen getraind worden op deelverzamelingen van de originele trainingsverzameling. Op deze manier kan het scaleerbaarheidsprobleem

vermeden worden. Hierbij zullen we ook een nieuwe methode van gekoppeld leren introduceren. In deze methodologie zullen de leden van de collectie van leermethoden hun kennis delen tijdens het leerproces. Dit leidt tot een nieuwe manier van transductief leren.

Abstract

As a result of the ever-growing influence of IT in research and companies terabytes of data are handled and stored daily. Therefore the interest in using this source of information is increasing steadily. It is there that data mining and machine learning gives us a hand. One recently developed set of tools in data mining are kernel models. These models excel in a variety of problem situations like classification, regression, time-series prediction problems with respect to the generalization performance tested on unseen data. The disadvantage of these models is that the computational demands for training them scale quadratically with the size of the training set. In this work we will show how this scalability problem for kernel models can be overcome by making use of a combination of methods from numerical and learning theory origin.

The proposed solutions, which have to be taken into account when one wants to use kernel models on large scale applications, will be presented on the basis of five pillars. These five pillars are: the model choice, the numerical procedures, the choice of the kernel, low rank approximations and the use of ensemble models. First we will show how models using a quadratic loss function will have a training procedure that consists of a linear system. We will show how iterative methods and low rank approximations can reduce the computational and memory complexity for solving this linear system. Also we will show that the kernel itself plays an important role in both the learning performance and the computational and memory complexity of the algorithms.

As a last solution we will propose the use of ensemble models for training kernel models on large data sets. Instead of training one model on a training set, a whole set of models is trained on subsets of the original training set. In this way the scalability problem can be avoided. In addition we will introduce a new method of coupled learning. In this methodology the members of the ensemble will share the knowledge during training. This leads to a new way of transductive learning.

Nederlandse Samenvatting: Kernfunctie Modellen voor Grootschalige Toepassingen

Inleiding

Als resultaat van de alsmaar toenemende invloed van IT in de onderzoeks- en bedrijfswereld worden er dagelijks terabytes aan data opgeslagen en verwerkt. De interesse om deze data te gebruiken groeit gestaag. Voorbeelden zijn de vele bedrijven die hun klanten data-bank willen gebruiken voor het vinden van ongekende klant-produkt relaties. Anderen willen de evolutie in hun verkoopcijfers voorspellen of bepaalde productie processen controleren. Ook in de onderzoekswereld groeit de vloed aan informatie door de toememende automatizatie van de experimenten. Daardoor wordt het haast onmogelijk om al deze data manueel te analyseren. Het is op dit punt dat gegevensontginning en machine-leertechnieken ons kunnen helpen. Deze relatief nieuwe domeinen zijn ontstaan uit een smeltkroes van verschillende technieken uit de statistiek, kunstmatige intelligentie, systeemidentificatie en anderen.

Het doel van gegevensontginning en machine-leertechnieken is het creëren van computer programma's die een taak vervullen, niet gebaseerd op voorgedefiniëerde regels, maar op basis van relaties die ze geleerd hebben. Dit leren gebeurt op basis van informatie, data of terugkoppeling die het programma ontvangt. De basis vraag blijft echter: Wat is leren?

"Van een computerprogramma wordt gezegd dat het leert van ervaringen *E met betrekking tot een taak T en performantie P, indien zijn performantie gemeten door P over de taak T toeneemt met ervaring.*"[88]

Om het werk te situeren zullen we elk van deze aspecten in meer detail behandelen.

De taak T waarop we ons in deze thesis zullen focusseren kan het best beschreven worden als *voorspellende gegevensontginning*. Met voorspelling wordt bedoeld dat het programma wordt gevraagd een beslissing te nemen over een situatie die het daarvoor nog nooit gezien heeft. Simpele opzoekingsprocedures worden daardoor uitgesloten. In deze predictieve taken kunnen we drie groepen onderscheiden. De eerste groep zijn de classificatie problemen, de tweede groep zijn de interpolatie of regressie problemen en de derde en laaste groep zijn de tijdsreeksvoorspellingsproblemen.

De manier waarop een programma ervaring E opdoet is door het beschikken over trainingsdata waarvan de oplossing gekend is. Tijdens de trainingfase kan het deze data direct gebruiken. Daardoor kan de trainingsfase omschreven worden als een direct en gesuperviseerd proces. Tijdens dit proces beschikt het trainingsalgoritme enkel over de trainingsdata. Dit maakt dat de hypothese, die door het leeralgoritme gevormd wordt, zich enkel baseert op deze trainingsdata. Dit proces noemt men inductief leren. In een andere situatie zal het trainingsalgoritme ook beschikken over de data waarover uitspraak moet gedaan worden. Dit is de zogenaamde testdata. Indien de hypothese, aangenomen door het programma, zowel gebeurt op basis van de train-en testdata spreekt men over transductief leren. Dit zal aan bod komen in het laaste hoofdstuk over groepsleren. In elk van deze gevallen zal de performantie P gemeten worden op basis van de aangepaste maten met betrekking tot het respectievelijke probleem: regressie, classificatie of tijdsreeksvoorspelling.

De bovengenoemde principes zijn van toepassing op verschillende types van leermodellen. In dit werk zullen we ons echter concentreren op één specifieke familie van modellen, met name de kernfunctie modellen. Een nieuw type van kernfunctie modellen zijn de ondersteunende-vector-machines. Deze nieuwe modellen zijn gebaseerd op recente ontwikkelingen in de statistische leertheorie en zijn nauw gerelateerd aan de Neurale Netwerken. Het doel van deze thesis is het uitdiepen van onze kennis over het gebruik van ondersteunende-vector-machines (SVM), regularizatie netwerken en kleinste-kwadraten ondersteunende-vector-machines (LS-SVM) voor toepassingen met grote trainingsverzamelingen.

De motivatie voor dit werk vindt zijn oorsprong in de volgende probleemstelling. Voor het trainen van het kernfunctie model moet er een kwadratische programmeringsprobleem (KP) opgelost worden. Dit KP scaleert met het aantal elementen van de trainingsverzameling. De geheugenvereisten voor het oplossen van dit mathematisch probleem scaleren op hun beurt kwadratisch met het aantal trainingspunten N. Daardoor zal men bij het werken met grote data sets al snel de fysische limieten van de hedendaagse computers bereiken.

De huidige desktop computer heeft beschikt over een vaste hoeveelheid geheugen en rekenkracht. Deze fysiche limieten dwingen ons een keuze te maken in het gebruiken van bepaalde trainingsalgoritmen. Voor bepaalde groottes van dataverzamelingen zullen de huidige kernfunctie modellen en hun overeenkomstige trainingsalgoritmen onpraktisch of zelfs onmogelijk worden.

Het vereiste geheugen voor een algoritme wordt uitgedrukt onder de vorm van geheugen complexiteit. Voor het trainingsalgorime van kernelfunctie modellen is de *geheugencomplexiteit* in de orde van $\mathcal{O}(N^2)$. Dit betekent dat de geheugenvereisten stijgen met de orde van het kwadraat van het aantal trainingspunten. Concreet kan men hieruit afleiden dat bij verdubbeling van het aantal trainingspunten N de geheugenvereisten stijgen met een factor 4. Dit wordt veroorzaakt door de stokering van de Hessiaanse matrix in het optimizatieprobleem. Voor een leerprobleem met 5000 trainingspunten betekent dit dat de benodigde geheugenvereisten bij een dubbele precissie voorstelling van numerieke getallen gelijk is aan 200Mbyte geheugen. In Figuur 1 wordt dit vereiste geheugengebruik uitgezet in functie van het aantal trainingspunten N. Vermits deze informatie typisch in het RAM geheugen van de computer wordt opgeslagen, kan men hieruit afleiden dat het met de huidige generatie van desktop computers, zeer moeilijk wordt om dataverzamelingen van meer dan 15000 elementen te behandelen.

Een mogelijke oplossingsstrategie is om al deze informatie niet volledig in het RAM geheugen op te slaan en te werken met intelligente tijdelijk opslag. Deze zogenaamde 'buiten-kern' algoritmen zullen niet behandeld worden in dit werk. Daarentegen zal er soms overwogen worden om een herberekeningsstrategie te gebruiken. Hierbij zal de informatie, die niet in het geheugen kan opgeslagen worden, herberekend worden op het moment dat deze nodig is. In dat geval moet men naast het geheugenverbruik ook de benodigde rekenkracht in het oog houden. Dit zal duidelijk aangegeven worden in de thesis.

In deze thesis zullen we een samenvatting geven van de literatuur en experimenten betreffende het gebruik van kernfunctie modellen voor grootschalige toepassingen. Op basis van deze ervaringen zullen we nieuwe oplossingen en richtlijnen geven. Deze zullen geformuleerd worden op basis van vijf belangrijke steunpijlers rond dewelke deze thesis is opgebouwd. Deze vijf pijlers zijn:

1. modelkeuze,

2. numerieke methoden,

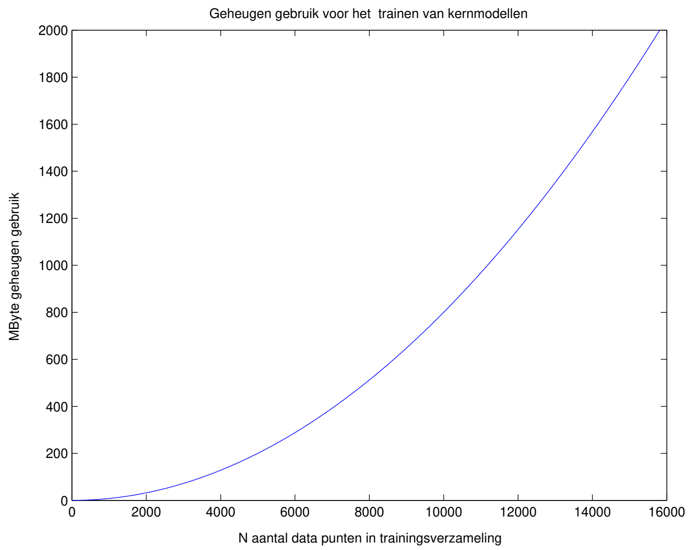

Figure 1: In deze figuur is het benodigde geheugengebruik uitgezet in functie van het aantal trainingspunten N voor kern modellen.

3. keuze van de kernfunctie,
4. lage rang benaderingen,
5. groepsleren.

Aan elk van deze pijlers wordt een hoofdstuk van de thesis gewijdt. Het zijn ook deze oplossingen die leiden tot de 'beslissingsboom' weergegeven in Figuur 2. Deze beslissingsboom gidst de lezer naar de meest aangewezen oplossing voor het trainingsprobleem waarmee hij/zij te maken heeft.

Na deze inleiding, probleemsituatie en motivatie zullen we een overzicht geven over de verschillende hoofdstukken zoals deze voorkomen in de thesis.

Theorie en Implementaties van LS-SVM en RN

De modelkeuze: De eerste stap in het beslissingsproces voor het trainen van kernfunctie modellen is het definiëren van het model. Elk kernfunctie model heeft zijn eigen specifieke trainingsprocedure en corresponderende geheugen- en rekencomplexiteit. In het eerste hoofdstuk geven we een inleiding in de wiskundige concepten op dewelke de kernfunctie modellen zijn gebaseerd.

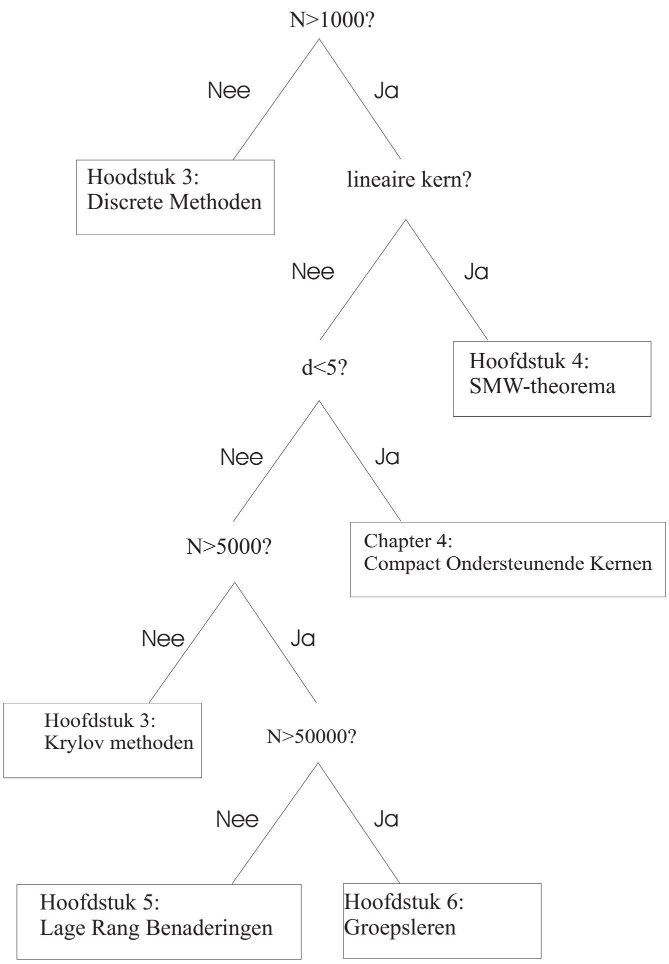

Figure 2: Richtlijnen voor het gebruik van de aangepaste methode voor een kernfunctie model met N trainingspunten in een d-dimensionele ruimte met een lineaire of niet-lineare kernfunctie.

Hierbij zullen de modellen gedefineerd worden in een voortbrengende kernfunctie Hilbert ruimte of als linear model in een kenmerken ruimte. Deze benadering leidt tot een primaal-duale interpretatie die zeer nuttig is vanuit van optimalizatie- en leerperspectief.

In deze thesis zullen we voornamelijk kernfunctie modellen gebruiken die gebruik maken van een kwadratische verliesfunctie zoals LS-SVM modellen en regularizatie netwerken. Beide formalismen geven aanleiding tot modellen die kunnen gebruikt worden voor zowel classificatie als regressie problemen. In beide gevallen kan dit het model, getrained op een dataverzameling $\mathcal{D}$, gedefinieerd worden als de oplossing van het stelsel

$$\begin{bmatrix} 0 & \mathbf{z}^T \\ \mathbf{z} & H \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{u} \end{bmatrix}$$

waarbij de Hessiaan H bestaat uit elementen $h_{ij} = z_i z_j K\left(\mathbf{x}_i, \mathbf{x}_j\right) + \delta_{ij}$. Voor classificatie is $\mathbf{z} = \mathbf{y}, \mathbf{u} = \mathbf{1}_N$, en voor regressie $\mathbf{z} = \mathbf{1_N}, \mathbf{u} = \mathbf{y}$. Het finale model neemt respectievelijk de vorm $f\left(\mathbf{x}\right) = sign(\sum_{p=1}^N \alpha_p y_p k(\mathbf{x}_p, \mathbf{x}) + b)$ voor classificatie en $f\left(\mathbf{x}\right) = \sum_{p=1}^N \alpha_p k(\mathbf{x}_p, \mathbf{x}) + b$ voor regressie aan. Zodoende worden deze modellen gekenmerkt door een trainingsprocedure die bestaat uit het oplossen van een stelsel van N lineaire vergelijkingen in N onbekenden. Deze modellen worden getypeerd door de volgende eigenschappen:

- de unieke oplossing van de parameters α en b,
- enorme flexibiliteit door de keuze van de kernfunctie,
- de rekencomplexiteit is onafhankelijk van de dimensie van de kenmerkenruimte,
- de parameters α en b zijn direct gerelateerd aan de modelfout $\mathbf{e}$ vermits $\alpha = \gamma\mathbf{e}$,
- de kwadratische scalering tussen het aantal trainingspunten en de complexiteit van het corresponderende trainingsproces.

Merk op dat de modellen die we beschrijven voor regressie nauw verwant zijn aan Gaussiaanse processen [81], Kriging ([73],[27]) en kernfunctie richel regressie (ridge-regressie) [116] en de reeds besproken regularizatie netwerken [108]. Maar ook voor classificatie bestaan er vele analoge en nauwgerelateerde modellen zoals: kernfunctie Fisher discriminant analyse [87], dichtsbijzijnde ondersteunende vector machines (proximal support vector machines) [50] en geregulariseerde kleinste kwadraten classificatie [113]. Deze relaties worden weergegeven in Figuur 3.

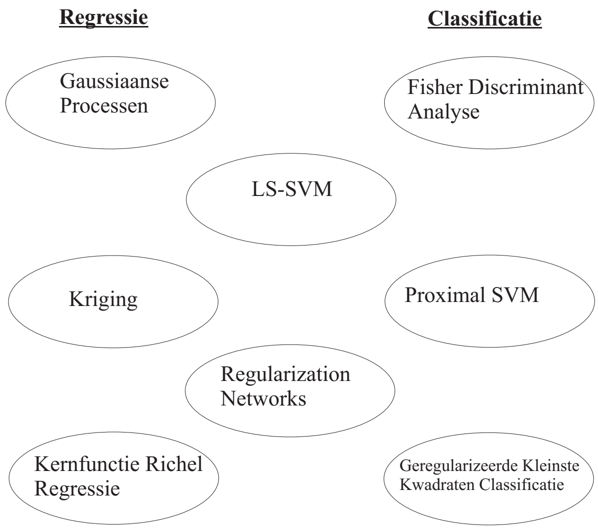

Figure 3: De voorgestelde LS-SVM ([129]) en Regularizatie Netwerken ([108]) zoals beschreven voor regressie zijn nauw verwant aan Gaussiaanse processen [81], Kriging ([73],[27]), kernel richel (ridge) regression [116]. Maar ook in classificatie vindt men links met: kernel Fisher discriminant analyse [87], proximal ondersteunende-vector-machines [50] en geregularizeerde kleinste kwadraten classificatie [113].

Numerieke Aspecten voor het trainen van LS-SVM's

De numerieke methoden: Na een wiskundige beschijving van de modellen gaan we dieper in op de numerieke methoden die nodig zijn voor het oplossen van de trainingsprocedure. We zullen een onderscheid maken tussen directe methoden en iteratieve methoden voor het oplossen van lineaire stelsels. Omwille van de postief definiete structuur van de coëfficienten matrix van het lineaire stelsel hebben we altijd een unieke oplossing.

De meest aangewezen directe methode voor deze stelsels is de Cholesky factorizatie. Deze heeft een rekencomplexiteit van de orde $\mathcal{O}(N^3)$. Voor grootschalige toepassingen waarbij N toeneemt zal dit al snel tot zeer hoge rekenvereisten leiden. Daardoor is deze direct methode meer aangewezen voor kleinere problemen ($N < 1000$).

Voor grotere toepassingen zijn iteratieve methoden meer aangewezen. Deze iteratieve methoden worden gekenmerkt door het feit dat zij de optimale oplossing berekenen door het iteratief aanpassen van een oplossingsvector. Dit update-proces kan op elk moment stopgezet worden waarbij er een tussentijdse oplossing wordt geleverd. Dit kan voordelen bieden in tijdskritische toepassingen. Iteratieve methoden hebben rekencomplexiteit van in de orde van $\mathcal{O}(lN^2)$. waarbij l het benodigde aantal iteratie-stappen voor convergentie is. Zolang het aantal iteraties kan beperkt worden, bieden deze iterative methoden een voordeel t.o.v. de directe methoden. Wij beschouwen twee groepen van iteratieve methoden.

De eerste groep is gebaseerd op het succesieve overrelaxatie principe (SOR). Dit algoritme wordt geoptimalizeerd naar onze toepassing. Hierbij maken we gebruik van symmetrische- en blokvarianten van het algoritme. Deze leiden tot een snellere convergentie en een efficienter geheugen gebruik. Verder zullen we aantonen dat deze SOR methoden verbeterd kunnen worden door de Gravis-Morris acceleratie. Deze veralgemening van het Aitken Δ^2 proces leidt tot een verbeterde convergentie en een lagere afhankelijkheid van de overrelaxatieparameter.

Een tweede groep van iterative methoden die we beschouwen zijn de Krylov methoden. De meest aangewezen Krylov methoden voor positief definiete matrices maken gebruik van toegevoegde gradienten. Experimenten tonen aan dat de toegevoegde-gradient methode een veel betere convergentiesnelheid heeft dan de SOR methoden. Deze convergentie kan, in het geval van het dubbel lineaire systeem van de LS_SVM training, nog verbeterd worden door het blok-toegevoegde-gradient algoritme. Deze methode toont een betere convergentie snelheid voor sets van lineaire systemen met een snel degenererend eigenwaardespectrum. Deze eigenschap wordt vaak experimenteel vastgesteld bij het leerproces van kernfunctie modellen.

Voor al de bovengenoemde algoritmen worden implementatie details weergegeven. Hierbij wordt er ook aandacht besteed aan goede stopcriteria voor de iteratieve processen. Links tussen de convergentie-snelheid en het conditie-getal en hun afhankelijkheid van de hyperparameters van het model worden besproken.

De theoretische aspecten samen met de experimentele resultaten laten ons besluiten dat iterative methoden meer aangewezen zijn voor problemen tot $N < 5000$.

De keuze van de kernfunctie

Aan de basis van elk kernfunctie model, ontworpen voor het vervullen van een bepaalde taak, ligt de keuze van de kernfunctie. Niet enkel heeft deze keuze invloed op de veralgemeningsperformantie van het leeralgoritme, ook heeft ze een bepalende invloed op de benodigde rekenkracht voor het trainen en uitvoeren van de modellen. Indien we 10 standaard UCI dataverzamelingen testen in een classificatie setup merken we dat het gebruik van niet-lineaire kernfuncties in sommige gevallen een significante verbetering biedt t.o.v. lineaire kernfuncties. De grote diversiteit aan kernfuncties biedt een enorme flexibiliteit aan kernfunctie modellen. Deze kernfuncties worden bepaald op basis van vereisten opgelegd door het Mercer theorema. Dit leidt tot een set van positief definiete kernfuncties. Binnen deze set van kernfuncties gelden een aantal rekenregels die het onder andere mogelijk maken lineair convexe combinaties te maken van kernfuncties. Dit biedt vele mogelijkheden voor het ontwerpen van kernfuncties voor iedere specifieke taak.

In dit hoofdstuk zullen we de kernfuncties indelen naargelang hun eigenschappen. We onderscheiden drie groepen: stationaire, lokaal stationaire en niet-stationaire kernfuncties.

De stationaire kernfuncties zijn translatie invariant functies. Tot deze groep van stationaire kernfuncties behoren de veel gebruikte radiale basis functies, splines en Matèrn kernfuncties. Voor grootschalige toepassingen bestaat er een deelklasse van de stationaire kernfuncties die speciale aandacht verdient. Dit is de klasse van de isotrope kernfuncties met compacte ondesteuning. Deze kernfuncties worden gekenmerkt door het feit dat ze nul worden buiten een bepaalde afkappingsafstand θ'. In Figuur 4 tonen we een compact ondersteunde radiale basis functie. Dit biedt veel voordelen op rekenkundig vlak. Twee voorbeelden van dergelijke functies zijn de

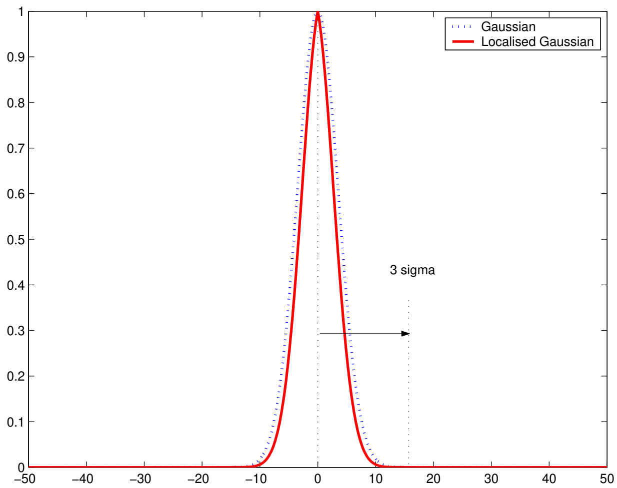

Figure 4: Deze figuur toont de compact ondersteunde Gaussiaanse radiale basis kernfunctie. De afkappingsafstand is $\theta' = 3\sigma$ waarbij σ de bandbreedte is van de kernfunctie.

spline-kernfunctie en de compact ondersteunde radiale basis functie. Wegens numerieke redenen zullen we enkel de laatste beschouwen.

Deze compact ondersteunde radiale basis functie biedt vele voordelen in zowel het trainings- als het evaluatieproces van kernfunctie algoritmen. Bij het gebruik van directe methoden tijdens de training kan de factorizatie een spaarse matrix oplossing geven. Dit kan zelfs verbeterd worden door permutatie algoritmen zoals kolom-tel permutaties, symmetrische minimum-graad permutaties en omgekeerde Cuthill-McKee algoritmen. Dit vergemakeijkt de achterwaartse substitutie na de factorizatie. Maar ook voor Krylov methoden biedt eem compacte kernfunctie voordelen. In Krylov methoden is de zwaarste rekenkundige stap het maken van een matrix-verctor vermenigvuldiging. Een compact ondersteunde kernfunctie levert een spaarse matrix op. Dit maakt dat de matrix-verctor vermenigvuldiging veel efficienter kan uitegevoerd worden.

Experimenten tonen deze rekenkundige voordelen aan. Maar voor bepaalde leerproblemen moet er echter de nodige voorzichtigheid gehandhaafd worden. In sommige niet-linearie tijdsreeksvoorspellingtaken leidt het gebruik van compact ondersteunde kernfuncties tot een vermindering van de leerperformantie. Dit is te weiten aan de hoge invoerdimensie van deze probleemsituatie. Daarom raden we deze compact ondersteunde kernfunc-

	Training	**Model Evaluatie**
Nyström factorizatie	$\mathcal{O}(m^2N + m^3)$	$\mathcal{O}(N)$
Cholesky factorizatie	$\mathcal{O}(p^2N + p^3)$	$\mathcal{O}(N)$
Gereduceerde basis	$\mathcal{O}(m^2N + (m+1)^3)$	$\mathcal{O}(m)$
vaste-grootte benadering	$\mathcal{O}(m^2N + 2m^3)$	$\mathcal{O}(m)$

Table 1: De rekenkundige complexiteit van de training en model evaluatie voor verschillende voorstellen van lage rang benaderingen. We besluiten dat de rekenkundige voordelen van deze lage rang benaderingen gelijk zijn.

ties enkel aan voor lager dimensionele problemen $d < 5$.

We bespreken ook lokaal stationaire en niet-stationaire kernfuncties. Tot deze laaste categorie behoren de scheidbare en lineaire kernfunctie. Beiden worden gekernmerkt door hun intrinsieke rang-deficiëntie. Deze rang-deficiëntie leidt tot een zeer efficiënte factorizatie. Deze kan op haar beurt uitgebuit worden in het oplossen van het lineaire stelsel door gebruik te maken van het Sherman-Morisson-Woodbury theorema. Dit maakt deze kernfuncties zeer interessant voor grootschalige toepassingen.

Lage Rang Benaderingen

Voor dataverzamelingen waar $5000 < N < 50000$ is het om wille van geheugenproblemen niet meer mogelijk om de trainingsprocedure van een kernfunctie model uit te voeren met een niet-lineaire kernfunctie. In deze gevallen moet men overschakelen op benaderingsmethoden op gebied van de numerieke procedures en/of de model keuze. We geven een overzicht van de verschillende mogelijke benaderingsmethoden waarbij we dieper ingaan op hun onderlinge relaties en de individuele computationele vereisten.

In het voorgaande hebben we aangetoond dat we de computationele complexiteit kunnen terugdringen door het maken van lage-rang benaderingen voor niet-lineaire kernels. We bespraken twee types van benaderingen.

De eerste techniek leidde tot een factorizatie van de kernfunctie matrix. Door het Sherman-Morrison-Woodbury theorema toe te passen op deze gefactorizeerde kernfunctie matrix kan het lineair systeem op een veel efficiëntere manier worden opgelost. Een van de manieren om de factorizatie te bekomen is door gebruik te maken van de Nyström benadering. De benadering kan gedaan worden een vaste maar lage computationele kost. We illustreren verschillende types van Nyström factorizaties en hun corresponderende computationele benodigdheden. Een tweede manier om de factorizatie door te voeren is gebaseerd op de Cholesky factorizatie. Deze methode buit de rank van de matrix volledig uit op een iteratieve manier. Hoewel de benadering resulterend van de lage rang benaderingen bijna nooit een vermindering in performantie aangeven, zijn de rekenkundige voordelen niet altijd verzekerd. Veel hangt af van de taak die men probeert op te lossen. In regressie taken toonde de Cholesky factorizatie een indrukwekkend rekendkundig voordeel. Maar voor classificatie taken toonde de Nyström benadering dan weer een sterk computationeel voordeel.

Een tweede classe van lage rang methoden is gebaseerd op een volledig andere methodologie. Om de computationele complexiteit te verminderen worden het aantal parameters in het model verminderd. Dit kan bekomen worden door ofwel het aantal parameters in de duale ruimte te verminderen, door een optimale basis constructie, ofwel door het aantal parameters in de primaire ruimte te verminderen. Deze laaste methode maakt opnieuw gebruik van de Nyström benadering wat leidt tot de *vast-grootte benadering* met een spaarse representatie. Het basis idee van deze methoden is de constructie van een basis door gebruik te maken van een subsample door middel van verscheidene selectie criteria.

Een verder aandachtspunt voor het gebruik van kernfunctie modellen voor grootschalige toepassingen is de evaluatietijd van nieuwe punten. Na training zal een orgineel LS-SVM model voor de evaluatie van een nieuw punt een rekencomplexitiet van in de orde van $\mathcal{O}(N)$ nodig hebben. Voor grootschallige toepassingen loopt deze evaluatie tijd dus lineair op met het aantal trainingspunten. In het geval van de lage rang benaderingsmethoden kan dit echter varieëren. De lage rang benaderingen op het model-niveau hebben typisch een lagere evaluatie complexitiet. Deze scaleert proportioneel met de grootte van de subselectie m. Voor tijdskritische toepassingen kan dit een belangrijke troef zijn. Een samenvatting voor alle lage rang benaderingen wordt gegeven in Tabel 1.

Verder vat Tabel 1 de computationele complexiteit voor training en model evaluatie van de verschillende lage rang benaderingen samen. Hierbij wordt er verondersteld dat er geen active selectie van subsample met grootte m wordt verricht. Verder wordt er vanuit gegaan dat de lineaire stelsels en eigenwaardenberekeningen gedaan worden op basis van directe methoden. We besluiten hieruit dat de rekenkundige voordelen van de lage rang benaderingen gelijk zijn. Op experimenteel vlak blijkt dat deze rekenkundige voordelen sterk afhankelijk zijn van de taak die men uitvoerd. Vanuit leerperspectief dient er op gewezen te worden dat deze lage rang benaderingen zelden tot significante reductie in generalizatie-performantie leiden. In vergelijking met het vorige hoofdstuk over numerieke methoden, kunnen we besluiten tot het volgende. We maken hierbij de volgende vergelijking met de modellen uit het vorige hoofdstuk. In het voorgaande hebben we reeds besproken dat de numerieke methoden voor het trainen van een kernfunctie model op basis van N trainingspinten een computationele complexitiet van $\mathcal{O}(N^3)$ heeft. Voor 5000 data punten komt dit in het slechtste geval overeen met $\mathcal{O}(N^3) = \mathcal{O}(5000^3) \approx 10^{11}$. Op basis van een gelijkaardige redenering begrenzen we het aantal datapunten in het geval van lage rang benaderingen tot $N < 50000$. In dit geval is de rekencomplexiteit $\mathcal{O}(m^2 N) = \mathcal{O}(m^2 50000) \approx 10^{12}$ met een subselectie grootte van $m = \frac{N}{10}$. Daarom zijn deze lage rang benaderingsalgoritmen adviseerbaar voor grote toepassingen met $5000 < N < 50000$.

Groepsleren

In Hoofdstuk 6 geven we een overzicht van de meest populaire groepsleer methoden. Deze methoden zijn gebaseerd op het opdelen van de computationele kost over verscheidene leeralgoritmen dewelke achteraf worden gecombineerd. Deze 'verdeel en heers' strategie maakt dat deze modellen kunnen gebruikt worden voor dataverzamelingen met $N > 50000$. Hierbij leggen we uit welke de voordelen zijn van deze methoden zowel op leer als rekenkundig vlak. Het vernieuwende is dat we het leerproces uitleggen als een procedure in twee delen. In het eerst deel stelt men een verzameling van leermethoden, dewelke men wil combineren, samen. Deze worden dan in een tweede stap samengevoegd tot één model. Deze benadering stelt ons in staat een duidelijke opdeling te maken van de verschillende bestaande methoden zoals: bagging, boosting, samenvoegen van experts en opeengestapelde modellen.

Vermits ons hoofddoel het gebruik is van kernfunctie modellen voor grootschalige toepassingen is, zullen we uitleggen hoe deze groepsleer modellen hierop kunnen toegepast worden. Om dit nader toe te lichten hebben we enkele van de theoretische resultaten uit de literatuur opgesomd. We bespreken hier concepten als stabilitet en generaliserend vermogen van de groepsleermethoden. Telkens worden de computationele aspecten besproken en vergeleken.

In een laaste deel bespreken we het effect van koppeling in groepsleermodellen. Dit idee komt voort uit het gebruik van gekoppelde locale minimizatie voor niet-convexe problemen. We introduceren het concept van koppelingsverzameling en tonen aan hoe dit kan gebruikt worden tot een manier van transductief leren voor zowel classificatie als regressie problemen. Links tussen meer-taaks leren en koppeling worden besproken en we tonen aan dat koppeling gezien kan worden als een vorm van groepsregularizatie. Vele aanpassingen met betrekking tot grootschalige problemen worden aangebracht. In de experimentele testen tonen we aan dat de koppeling tot een verbeterde leerperformantie leidt.

Algemene Conclusies en Verder Onderzoek

Samenvatting

In de voorgaande secties hebben we verschillende afleidingen voor kernfunctie modellen besproken die gebruik maken van een kwadratische verliesfunctie. Een eerste benadering is opgebouwd in een voortbrengende-kernfunctie-Hilbert ruimte (RKHS). Hierbij wordt de functionaal geoptimaliseerd rekeninghoudend met een goede balans tussen het empirische en struturele risico. Dit leidt tot een lineair geparameteriseerd model. Door gebruik te maken van een kwadratische verliesfunctie voor het minimaliseren van de empirische risico functie, kunnen de optimale parameters van het model gevonden worden door het oplossen van een lineair stelsel. Deze oplossing leidt tot een regularizatie netwerk model. Een gelijkaardige afleiding kan gedaan worden vanuit een optimizatie benadering door het definiëren van een model in een zogenaamde kernmerken ruimte. Hierbij definiëren we opnieuw een kostfunctie die een optimale balans maakt tussen het strucurele en empirische risico. Deze benadering introduceert het kleinste-kwadraten ondersteunende-vectormachine (LS-SVM) model op.

We toonden aan dat dat beide methodologieën voor zowel regressie, tijdsreeksvoorspelling als classificatie problemen toepasbaar zijn. De hoofdeigenschap van deze modelformulering is de unieke oplossing van het optimalizatieprobleem bepaald door de keuze van het model. De oplossing van dit optimalizatieprobleem kan gevonden worden uit een stelsel van lineaire vergelijkingen in N onbekenden. Dit toont aan dat dit probleem scaleert met het aantal invoerpunten van het leerprobleem.

Deze regularizatie netwerken en LS-SVM's hebben vele relaties met andere theorieën. Daardoor kunnen de meeste van de gepresenteerde resultaten uit deze thesis toegepast worden op al deze andere modellen.

Ook hebben we de relaties met de orginele ondersteunende-vector-machines (SVM) uitgelegd. Hierbij hebben we een overzicht gegeven over de meest gekende formuleringen en implementaties voor grootschalige toepassingen en dit met bestrekking tot spaarsheid van oplossingen en de gevolgen voor de optimalizatieproblemen.

In het tweede deel hebben we verschillende numerieke methodes gepresenteerd voor het trainen van LS-SVM's. We bestudeerden directe en iteratieve methoden en vergeleken hun eigenschappen met betrekking tot grootschalige problemen. Herinner hierbij dat het trainen van LS-SVM modellen en RN bestaat uit het oplossen van een linerair stelsel. Doordat de coëfficientenmatrix van het lineaire systeemde een positief definiete structuur heeft, bestaan er zeer efficiënte numerieke oplossingsmethoden. Met betrekking tot grote toepassingen hebben de iteratieve methoden zoals (SOR, CG,...) de meeste voordelen.

De eerste iteratieve methode die we gestest hebben was de succesieve overrelaxatie methode (SOR). Door het optimaal benutten van de structuur van de coefficientenmatrix in combinatie met een blok-structuur implementatie konden we de efficiëntie van het Symmetrische SOR verbeteren. Bijkomend konden we het aantal iteratie stappen beperken door de GM-acceleratie methode. Bovendien zorgde deze GM-acceleratie voor een verminderde afhankelijkheid van de blok-grootte en de overrelaxatie-parameter. Dit alles maakt dat de blok-SSOR vele voordelen biedt t.o.v de klassieke SOR.

Vergelijking tussen de blok-SSOR methode en Krylov-methoden, zoals de toegevoegde-gradient methoden, tonen echter aan dat deze laatste in alle testen tot betere performaties leiden. Een tweede Krylov methode, de blok-toegevoegde-gradient methode kan deze performantie zelfs nog verbeteren. Voor alle trainingsmethoden op niet-linearie classificatieproblemen gebaseerd op de blok-toegevoegde-gradient methode toonde dit algoritme een verbeterde convergentiesnelheid. Hieruit besluiten we dat de Krylov-methoden de meest aangewezen numerieke methoden zijn voor trainingsproblemen met $N < 5000$.

In een derde deel bestudeerden we de rol van de keuze van de kernfunctie en de invloed op het trainingsproces. Om de computationele gevolgen van de kernfunctie keuze te bestuderen delen we deze in op basis van verschillende groepen van kernfuncties. Deze opdeling bestaat uit stationaire, lokaal-stationaire en niet-stationaire kernfuncties. Voor elk van deze types bespraken we computationele -en geheugen voordelen die behaald kunnen worden door gebruik te maken van hun eigenschappen.

Veel aandacht werd besteed aan het gebruik van compact ondersteunende kernfuncties. We toonden aan dat de populaire radiale basis functies op een efficiënte manier kunnen omgevormd worden in kerfuncties met een compacte ondertsteuning. Het gebruik van compact ondersteunende kernfuncties kan de geheugen en compuationele vereisten verminderen. In onze studie hebben we gezien dat voor alle laag-dimensionele problemen zowel de generalizatie-performantie als het conditienummer van de kernfunctie-matrix niet beinvloed worden door de compact ondersteunende kernfuncties. Dit laatste is voornamelijk belangrijk voor de convergentie-eigenschappen van iteratieve methoden zoals de toegevoegde-gradient methoden. Op een chaotische tijdsreeksvoorspelling hebben we kunnen aantonen dat men faalt om een goede performantie te halen indien men een spaarse Gram-matrix probeert te verkrijgen door gebruik te maken van een compact ondersteunende kernfuncties. Daarom besluiten we uit deze studie dat de compact ondersteunende kernfuncties enkel aangewezen zijn voor laag-dimensionele problemen $d < 5$. In een laaste deel van dit hoofdstuk hebben we effiënte methoden besproken voor het trainen van LS-SVM modellen voor niet-stationaire scheidbare en lineaire kernels. We toonden aan dat de intrinsieke definitie van deze kernels aanleiding geeft tot rang-deficiënte kernfunctie matrix. Deze rangdeficiëntie leidt tot goede mogelijkheden voor het factorizeren van de kernfunctie matrix. Dit kan op zijn beurt uitgebuit worden tot een betere geheugen-en computationele complexiteit van de algoritmen.

Maar ook voor algemene niet-lineaire kernfuncties kunnen er effieciënte factorizaties gemaakt worden. Hiervoor doet men beroep op lage rang benaderingen. We hebben gezien dat er lage rang benaderingen kunnen gemaakt worden op twee niveaus. In het eerst geval maakt men een benadering van de kernfunctie matrix. Deze is gebaseerd op de Nyström benadering of op de Cholesky factorizatie. Beide leiden tot een efficiënte factorizatie die kan uitgebuit worden door gebruik te maken van het Sherman-Morrison-Woodbury theorema. Ten tweede kunnen we ook lage rang benaderingen op het model niveau toepassen. Hierbij reduceren we het aantal parameters in de primaire of duale ruimte. Deze geven aanleiding tot een gereduceerde basis of vaste-grootte benaderingen. Deze tweede groep van lage rang benaderingen heeft het bijkomend voordeel dat ze ook een computationeel voordeel bieden tijdens evaluatie.

We besluiten dat de computationele voordelen voor training van deze lage benaderingen bijna allemaal in dezelfde orde liggen. Afhankelijk van de taak zal de ene methode een betere performantie geven dan de andere. Ook vanuit leerperspectief is er bijna nooit een vermindering van de performantie. Daarom worden deze lage rang methoden geadviseerd voor grootschalige taken tussen $5000 < N < 50000$.

Voor dataverzamelingen waar het aantal datapunten groter is dan $N > 50000$ wordt het zeer moeilijk om één model te trainen op al de data. Daarom stelden we in deze gevallen een 'verdeel-en-heers' strategie voor. We hebben een inleiding gegeven in verschillende groepslleermodellen en hun eigenschappen naar grootschalige toepassingen. Verder hebben we een nieuwe groepsleermethode ontwikkeld waarbij het leren van de verschillende submodellen gebeurt door middel van koppeling. Hierbij toonden we aan hoe dit kan leiden tot een nieuwe manier van transductief leren.

Verder onderzoek

Alhoewel een ruime vooruitgang is geboekt binnen dit werk, zijn er nog steeds vele pistes die verder onderzoek vereisen. Laten we enkele voorbeelden overlopen in dezelfde volgorde als de stuctuur van de thesis.

In het eerste deel stoonden we het belang aan van de keuze van de kostfunctie. Iedere kostfunctie leidt tot een specifiek optimalizatie probleem. We weten ook dat voor een specifieke kostfunctie de oplossing spaars is wat op zijn beurt vele voordelen heeft voor grootschalige toepassingen. Verder onderzoek moet uitwijzen welke andere kostfuncties een spaarse oplossing kunnen opbrengen voor de modellen die behandeld zijn in deze thesis. Maar dit zal waarschijnlijk niet meer leiden tot een lineair systeem.

Op numeriek vlak is het oplossen van een semi positief definiet lineair stelsel door middel van Krylov methoden nog steeds een actief onderzoeks domein. Wellicht zullen andere Krylov methoden de performantie nog kunnen opdrijven. Verdere testen moeten dit nog uitwijzen.

Ook voor het ontwerp van kernfuncties zijn er nog vele open vragen. Er is weinig geweten over een optimale kernfunctie die optimale theoretische eigenschappen heeft voor leertaken in een hoogdimensionele ruimte en dit tegen een zo laag mogelijke computationele kost. Een eerste voorstel werd in deze thesis gedaan met betrekking tot kernfuncties met een compacte ondersteuning. Theoretisch is het echter nog niet goed begrepen hoe groot de afkappingsafstand mag zijn. Verder is er ook weinig geweten over scheidbare niet-stationaire en andere rang deficiente kernfuncties. Deze dragen de optimale eigenschappen voor het reduceren van de rekenkundige obstructie waarmee we nu geconfronteerd worden. Maar de vraag is of dergelijke scheidbare niet-stationaire kernfuncties bestaan en hoe ze het generalizerend vermogen van het leeralgoritme beinvloeden.

Verder hebben we gezien dat de methoden die gebruik maken van een factorizatie van de kernfunctie matrix afhankelijk zijn van een subselectie op dewelke de factorizatie is gebaseerd. In de literatuur worden er veel verschillende subselectie schema's voorgesteld op basis van informatie theorie, optimale basis opspanning en vele anderen. Het is echter niet goed geweten welke oplossing er zowel op theoretisch als op experimenteel vlak de beste resultaten biedt.

In het laatste deel stelden we het gebruik van groepsleermethoden op basis van subselecties voor. Pas recent zijn de eerste theoretische bewijzen gegeven voor het aantonen van het nut van deze strategie. Hierbij is het nog steeds niet geweten of de subselectie met of zonder overlapping moet gebeuren. In onze bespreking zijn we er steeds van uit gegaan dat al de data moet gebruikt worden. Een andere mogelijk scenario zou zijn dat de performantie van het groepsleermodel constant wordt geëvalueerd. Hierbij zou men tijdens de training het groepsleermodel iteratief kunnen uitbreiden zodat een performantie criterium niet wordt bereikt.

Ook het concept van gekoppeld leren opent nog vele vragen. Mogelijke interessante te onderzoeken pistes zijn het gebruik van deze methode voor groepsleermethoden met modellen die ofwel verschillende kernfuncties gebruiken of verschillende hyperparameters. Indien men in dit laatste geval het gekoppelde lokale minimizatie principe gebruikt voor de hyperparameterselectie krijgt men zowel een sychronizatie op het hyperparameterselectie als op het groepsleer niveau.

List of Symbols and Notations

This section lists the symbols and acronyms that will be used in this thesis. The notation in the thesis allows to distinguish among scalars, vectors and matrices. Scalars are denoted by lower case characters, vectors in boldface and matrices in capital characters.

Acronyms

RBF	radial basis function
CG	conjugate gradient
BCG	block conjugate gradient
KKT	Karush-Kuhn-Tucker
MSE	mean squared error
RKHS	reproducing kernel Hilbert space
AUC	Area Under the Curve
ROC	Receiver Operation Characteristic
SOR	Successive Overrelaxation
SSOR	Symmetric Successive Overrelaxation
SymmLQ	Symmetric LQ
MinRes	Minimum Residual
I/O	input/output
GM	Graves-Morris Acceleration
VC	Vapnik-Chervonenkis
QP	quadratic programming
CLM	coupled local minimizers

Scalar, Vector and Matrix Notations

x	scalar or element of $\mathbb{R}$
$\mathbf{x} = [x_1 x_2 \dots x_d]^T$	column vector in $\mathbb{R}^d$ with components x_i
$\mathbf{1}_d = [1 \dots 1]^T$	vector of ones in $\mathbb{R}^d$
$\mathbf{0}_d$	null vector in $\mathbb{R}^d$
$\mathbf{x}^T\mathbf{x}' = \sum_{i=1}^{d} x_i x_i'$	inner product in $\mathbb{R}^d$ between $\mathbf{x}$ and $\mathbf{x}'$
$\prod_{i=1}^{d} x_i = x_1 \cdot \ldots \cdot x_d$	product notation
$\lVert\mathbf{x}\rVert_1 = \sum_{i=1}^{d} \lvert x_i\rvert$	the one norm
$\lVert\mathbf{x}\rVert_2 = \left(\sum_{i=1}^{d} x_i^2\right)^{1/2}$	the Euclidean or two norm
$\lVert\mathbf{x}\rVert_p = \left(\sum_{i=1}^{d} x_i^p\right)^{1/p}$	the Holder norm or p-norm
$\lVert\mathbf{x}\rVert_\infty = \max_i \lvert x_i\rvert$	the infinity norm
A	matrix in $\mathbb{R}^{m\times n}$
A^T	the transpose of A
A^{-1}	the inverse of an invertible matrix A
I_m	the identity matrix of dimension $m \times m$
$\lVert A\rVert_2 = \sqrt{\lambda_{\max}}$: λ eigenval. of A^TA	2-norm of A
$\lVert A\rVert_p = \max_{\mathbf{x} \text{ s.t. } \lVert\mathbf{x}\rVert_p=1} \lVert A\mathbf{x}\rVert_p$	p-norm of A
$\lVert A\rVert_F = \left(\sum_{k=1}^{m}\sum_{l=1}^{m} a_{kl}^2\right)^{1/2}$	the Frobenius norm of A
$cond_p(A) = \lVert A\rVert_p \left\lVert A^{-1}\right\rVert_p$	condition number w.r.t. the p-norm
$\rho(A)$	spectral radius of a matrix A
$\mathrm{diag}(\mathbf{x})$	a matrix of $\mathbb{R}^{d\times d}$ with the vector $\mathbf{x}$ on the diagonal, all other elements being 0

Symbols

$\mathbb{N}$	the set of natural numbers
$\mathbb{N}_0$	the set of natural numbers excluding 0
$\mathbb{R}$	the set of real numbers
$\mathbb{R}_0$	the set of real numbers excluding 0
$\mathbb{R}_0^+$	the set of positive real numbers excluding 0
$\mathbb{Z}$	the set of integer numbers
$2\mathbb{Z}+1$	the set of odd integer numbers
$f = f(\cdot)$	a function
$f(x)$	evaluation of f in x
$\|f\|_k$	norm in a RKHS induced by a kernel k
$\frac{\partial f(\mathbf{x})}{\partial x_i}$	the partial derivative of $f(\mathbf{x})$ w.r.t. the i-th component of $\mathbf{x}$
$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = [\frac{\partial f(\mathbf{x})}{\partial x_1} \dots \frac{\partial f(\mathbf{x})}{\partial x_d}]^T$	gradient of $f(\mathbf{x})$
N	number of data points in the training set
d	dimension of the input space
$d_{\mathcal{H}}$	dimension of the feature space
k	kernel function
K	kernel or Gram matrix
$\mathcal{L}$	Lagrangian
γ	regularization constant
$\mathcal{H}$	Hilbert space
$\mathcal{F}$	Hypotheses Space
$\mathcal{X}$	input space
$\mathcal{Y}$	output space
sup	supremum
$a \ll b$	a is much smaller than b
$a \simeq b$	a approximates b
$\delta_{i,j} = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{if} \quad i \neq j \end{cases}$	the Kronecker delta symbol
$L^2(\mathcal{X})$	set of square integrable functions over $\mathcal{X}$
$n!$	factorial of $n \in \mathbb{N}$

Statistics

$P(x)$	probability of an event x
$P(x \mid y)$	conditional probability measure evaluated at x
$p(x)$	probability density function evaluated at x
$p(x \mid y)$	conditional probability density function evaluated at x
$E[x]$	expected value of x
$\mathcal{D}$	training data set
$f = f_{\mathcal{D}}$	a model trained on a training data set $\mathcal{D}$

Contents

Chapter 1

Introduction

1.1 Situation

As a result of the ever-growing influence of IT in research and companies, terabytes of data are handled and stored everyday. Therefore the interest in using this source of information is increasing steadily. Examples of this are the many companies that want to use their costumer information databases to extract new unknown relations from it. Others want to predict further evolutions in their sales rate or control certain production processes. Also in research, the flood of information gathered by the increased automatization of experiments, is growing every day. It is almost impossible to manually analyze all the data. *Data Mining* and *Machine Learning* can help us at this point. Both are relatively new domains that include techniques from statistics, artificial intelligence, system identification and others.

The aim in Data Mining and Machine Learning is to design computer programs that solve a task not based on predefined rules provided by the user but using relations that they 'learned' from the information, data or feedback that they receive. The question now remains: What is learning?

"*A computer program is said to* **learn** *from experience E with respect to some class of tasks T and performance P, if its performance at tasks in T, as measured by P, improves with experience.*" [88]

To situate this work we will explain each of these aspects in more detail.

The tasks T that we will focus on, can best be described as *predictive data mining*. By prediction one means the situation where the program is asked to make a decision in a situation that it has not seen before. This excludes simple look-up procedures. Correspondingly one can subdivide the learning tasks into three major groups. The first are *classification problems*, the second class are interpolation or *regression problems* and the third are *time-series* problems. Further in this section we will give a real-life examples for each of these problems.

The way the program gains experience E is defined by the training process. In the situations we discuss a *supervised* and *direct* training setup will be used. The program will in its training phase dispose of a set of examples together with the assumed outcome of the task. Since the correct outcome is delivered, this information can be used during the training or learning phase. This is called *supervised learning*.

A second aspect is that the program receives *direct feedback* during the training in fulfilling the goals. This is different from learning schemes like *reinforcement learning* [88] where the program only gets indirect feedback about the accomplished task. Based on this set of examples, called the training data, the training algorithm will choose a possible state of the program in which it is best suited to fulfill further tasks. This can be understood as the choice of a *hypothesis* on which further predictions are made and which has been learned during training. Since one lacks further information, the assumption is that the most suitable hypothesis is the one that best fits the observed training data. This is called *inductive learning* [88],[147].

This inductive learning setup will be mainly used in this work. However, in some problem situations one disposes also in advance of the data on which the further predictions have to be made. For example the new data that have to be classified by the algorithm. These data can also be used for choosing a hypothesis. This is called *transductive learning* and will be discussed in the last chapters.

The performance P of the program is its predictive power on new unseen data. Different measures exist to test this. The rate of misclassified unseen examples is for example a good performance measure for a classification algorithm. Therefore these unseen data are often called the *test data*. Measures for regression and time-series will be discussed in the following chapters.

Different model structures may obey these defined demands we stated above. In this work we will focus on one class: *kernel methods*. A new generation of these kernel models, Support Vector Machines, received a lot of attention recently. Support Vector Machines methods are based on new advances in statistical learning theory and have a close relation with the domain of neural networks. This work led to an explosion of applications and a deepening of the theoretical analysis of these methods. Based on these accomplishments on both the theoretical and experimental level, ker-

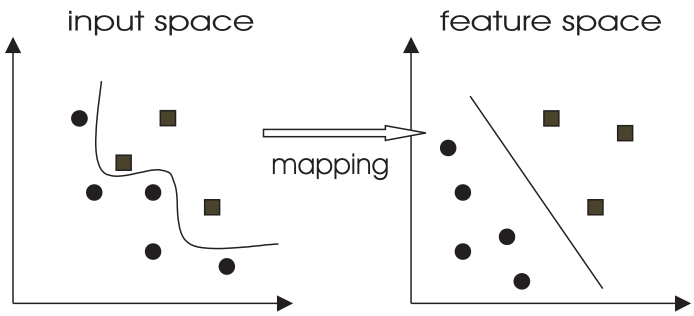

Figure 1.1: This figure shows the idea of kernel classification models where it solves non-linearly separable classifcation problem by mapping the input data into a high-dimensional feature space in which it becomes linear separable. The two classes of training points are indicated by circles and squares.

nel methods are now established as a standard tool for a variety of problems such as classification problems, regression, de-noising, and dimensionality reduction. The goal of this research is to study the use of Support Vector Machines (SVM), Regularization Networks and Least Squares Support Vector Machines as a machine learning technique for large data sets.

The idea of SVM originated from a classification problem formulation. The SVM uses a linear hyperplane with a maximal margin to create a classifier that is able to divide a set of attributes into different classes. Training the SVM to perform its task consists of finding this hyperplane on the basis of a training set of experimental data. In other words the SVM learns its experience based on the training set of known examples by choosing the most optimal hyperplane that separates the training data. The power of the algorithm lies in the fact that it can also handle non-linearly separable data. For these problems the SVM maps the input data, by means of a non-linear transformation, into a high-dimensional *feature space*. As a result, the original non-linear problem is transformed into a linearly separable one (see Figure 1.1). The generalization capability of the algorithm (i.e. its prediction accuracy on previously unseen data) is often better than that of neural networks, decision trees and many other classical methods. Finding the ideal separating hyperplane in this high-dimensional space becomes more difficult if one uses large amounts of training data. In mathematical terms it consists of solving a quadratic programming problem. These quadratic programming problems demand huge memory requirements in the case of large data sets. In our specific case we will focus on a special group of kernel models: the Least Squares Support Vector Machines and Regularization Networks. Training these models involves solving a linear system instead of quadratic programming problem. But also in this case the linear systems will scale with the number of data points. Therefore we will study the characteristics of this training procedure and show how one can improve them towards large data sets.

1.2 Motivation

The reason why we are interested in upscaling the training procedure towards large data sets has its origin in the application areas of these techniques. Examples of projects where these techniques can be used are: fraud detection in credit cards and cellular phone networks, money laundering detection, classification of micro-array experiments, image classification, and many others. One common aspect of the applications mentioned above is the vast amount of data that one has available. Since using more data is likely to improve the performance of the algorithm, the challenge in these tasks is to use as much data as possible.

This situation is different form a classical setup where one only has a limited amount of data and the goal is to achieve the best statistical prediction on this task. In this setup computational aspects are less a bottleneck. In our setup a limited amount of data is not a problem, the challenges rather shift to the computational aspects of the training. Training models with high amounts of data will demand more computational and memory resources. The current desktop computers have a fixed amount of computational power and memory resources which forces us to make a trade-off in the choice of models and training algorithms one can use. In some cases of very large data sets certain models, and their corresponding training, become impractical, if not impossible, to use. Therefore in the remainder of this work we will see that one sometimes has to make approximations or has to use other models formulations to be able to train these large amounts of data. Approximations will in some cases lead to models with a different or possible inferior performance compared to the original models trained on the whole data. The most optimal models in this context are the ones with the smallest decrease in performance and the highest computational and memory advantage.

1.2.1 Examples

Before going deeper into the model definitions, first some real life examples which can be solved using the models discussed in this work, are shown. The examples below are not large scale examples but are chosen based on the fact that they are easy to graph. However, similar large scale problems can be created just by increasing the number of measurements.

Regression

In this example we want to find the functional relation between the acceleration of the head of the motorcycle driver and the time after impact of the vehicle. To accomplish this measurements in time are made of the acceleration of the head on simulated impacts. This gives us a data set of 133 measurements. The input data are the time instances on which a measurement is made. They are nonequidistant on a millisecond scale. The acceleration measurements are considered as the output of the system. As is often the case in real life tests, the measurements are contaminated by noise. The goal is now to construct a model that learns the functional behavior of these measurements in order to make predictions at time instances where no measurements were taken. Notice hereby that both the input and the output values are elements of a real and continuous interval. This is called a *regression* setup.

In Figure 1.2 the measurements are indicated by the dots. The functional relationship between the input and output is learned by using an LS-SVM kernel model. So, one can see that this model is capable to give the relation between both variables based on noisy measurements. On top of this predictions can be made in new unseen time instances. The LS-SVM model is trained with a Gaussian RBF kernel with optimal hyperparameters $\left(\gamma, \sigma^2\right) = (2.21, 34.8)$ found by the 10-fold cross-validation routine in LS-SVMlab.

Classification

In this example a classification task is illustrated. The goal of the model is to learn the distinction between the different classes of the iris flower family. This task needs be learned based on some given examples, based on the measurements one has of the individuals together with the corresponding class labels. In this example the data set consists of 150 random samples of flowers from the iris species setosa, versicolor, and virginica. From each

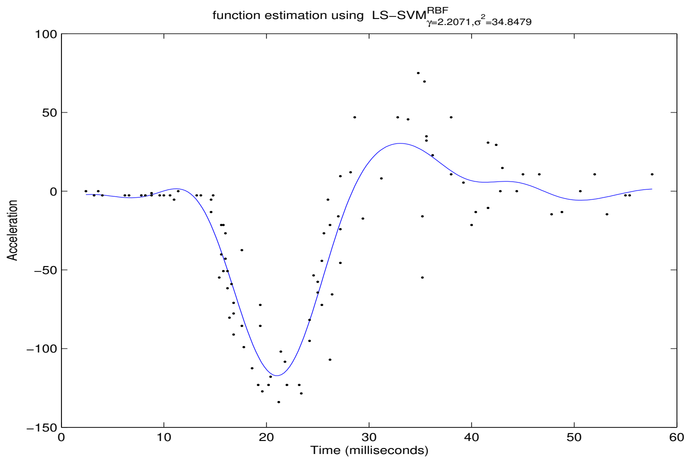

Figure 1.2: This is an example of a non-linear regression task that can be accomplished by the models studied in this work. The measurements of the acceleration of the head of a motorcycle driver on impact are given in a millisecond time scale are indicated by the dots. They are used as training data. The functional relationship between the time and acceleration is learned by the LS-SVM kernel model and is given by the full line. The LS-SVM model is trained with a Gaussian RBF kernel with optimal hyperparamters $\left(\gamma, \sigma^2\right) = (2.21, 34.8)$ found by the 10-fold crossvalidation routine in LS-SVMlab.

species there are 50 observations or input data for sepal length, sepal width, petal length, and petal width in centimeters.

In Figure 1.3 the different observations of the iris flower against two of its attributes are graphed. The three subfigures show the classification boundary the model has learned. For each of the three classification tasks one class was taken versus the two others. The color difference shows the clear class distinction that is learned by the model based on the given data. These classification models can be used for further classification of new unseen examples. The LS-SVM model is trained with a Gaussian RBF kernel with hyperparameters $\left(\gamma, \sigma^2\right) = (10, 4)$ in all three examples.

Time-series prediction

A classical time-series problem is given by the Wolpert sunspot data. Each year the sunspots visible on the surface of the sun are counted. This work started already in 18th century by Wolpert and was continued until now. The data are given as a series of discrete numbers over time. The measurements show the evolution in time of the amount of sunspot as can be seen in Figure 1.4. The goal now is to make a prediction or extrapolation in time of this data. One wants to know how many sunspots can be expected in the years to come. This is called a *time-series* problem. As will be shown later, this problem can be translated into a regression problem that can be solved by the kernel models that we will study. In this experiment we used the data of the first 220 years to predict the result for the next 60 years. The result of the prediction is shown in Figure 1.5. The prediction of the amount of sunspots is made by a NARX model based on an LS-SVM with a sliding window approach of size 30 trained on the data given in Figure 1.4. As kernel the Gaussian RBF is used with hyperparameters $\left(\gamma, \sigma^2\right) = (10, 15)$.

These three examples demonstrate that the LS-SVM and RN kernel models show a good performance on non-linear classification, regression and time-series prediction tasks.

1.2.2 Memory Usage

In the previous examples we saw a classification, regression and time-series problem with a few hundred data points that represented the result of measurements. In many other practical applications these amounts can increase from a few thousand to some millions of data points. The main bottleneck for training kernel models with many data points are the necessary memory resources. But what does this mean in practice? How is memory indicated

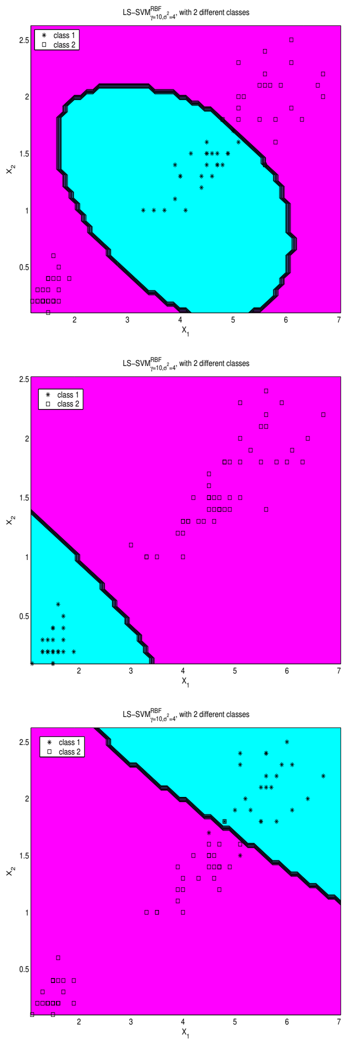

Figure 1.3: This is an example of a classification task on the iris data set. Each of the three figures shows a classification of one of the iris species setosa, versicolor, and virginica in relation to the two other species. The class labels of the training points based on actual measurements are indicated by different markers (circles or stars). The non-linear classification indicated by the full line is learned an LS-SVM model. Each figure shows one class versus the two other. The LS-SVM models are trained with a Gaussian RBF kernel with hyperparamters $(\gamma, \sigma^2) = (10, 4)$ in all three examples.

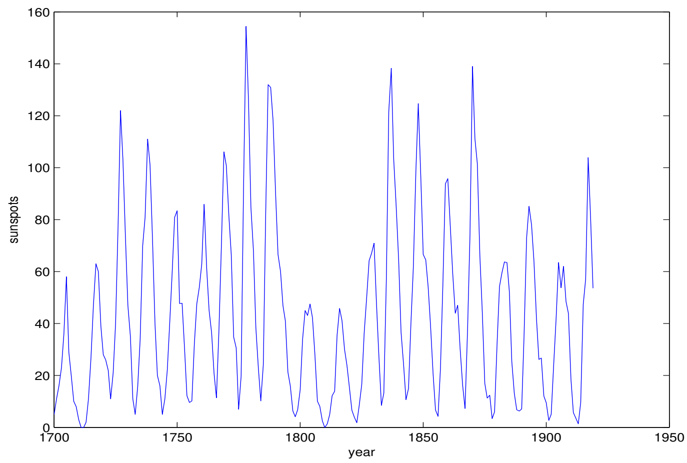

Figure 1.4: The sunspot data, 1700-1920.

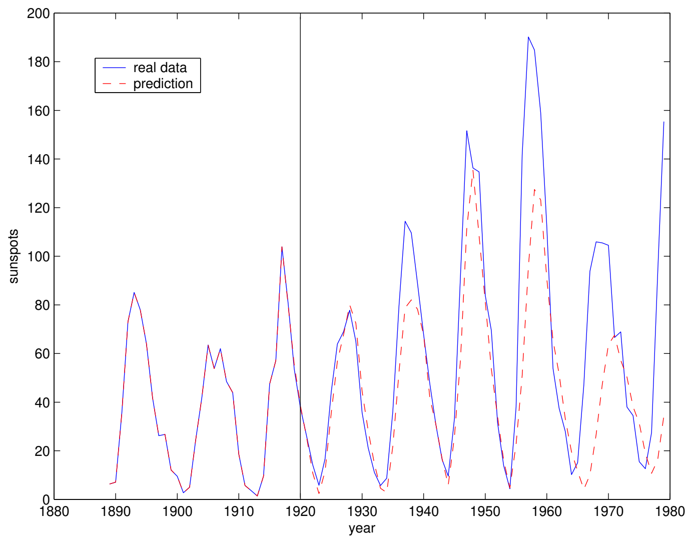

Figure 1.5: This is an example of a time-series task on the sunspot data.The prediction of the amount of sunspots by a NARX model based on an LS-SVM with a sliding window approach of size 30 trained on the data given in Figure 1.4. The prediction starts in 1920 and is indicated by dashed line. The real measured data is given by the full line. As kernel the Gaussian RBF is used with hyperparamters $\left(\gamma, \sigma^2\right) = (10, 15)$.

and how much memory is needed by the algorithms that are used to train the kernel models?

Most algorithms running on a computer use either *single precision* or *double precision* for mathematical computations. This means that each number is represented by respectively 4 Bytes or 8 Bytes of memory. The RAM (Random Access Memory) of a computer is always indicated in MB or Mega Byte. Desktop PCs or workstation have typical 256MB (256×10^6Byte) up to 2 GB or 2 $\times 10^7$ Byte RAM.

The memory needed for training kernel models scales with the number of training points. The standard measure for accounting the memory usage is the *memory complexity*. The interesting aspect of this measure is that it shows how complexity scales with the size of the data points (the 'scalability'), where the size of the data set is described by the number N. The typical measure for memory and computational complexity is given in the *big-O notation*, $\mathcal{O}(\centerdot)$. This is defined as follows: let n be an integer variable which tends to infinity. Also, let $\tau\,(n)$ be a positive function and $\upsilon(n)$ any function then $\upsilon = \mathcal{O}(\tau)$ to means that $|\upsilon\,(n)| < c\tau(n)$, $\forall n$, where c is some constant [66].

An algorithm may have memory complexity $\mathcal{O}(N^2)$ (also read as 'of the order of the square of the size of N'), in which case if the number of data points N doubles in size, the needed memory resources are four times higher.

For the kernel models we will see that the memory complexity is in most cases (unless indicated) in the order of $\mathcal{O}(N^2)$ where N is the number of data points. In the next chapters we will show that this is caused by the storage of a Hessian matrix which is needed in the optimization process involved in training these kernel models. A small calculation easily shows that; if we use 5000 data points the memory usage is $\left(5 \times 10^3\right)^2 \times 8$ Byte= 200 MByte. In Figure 1.6 one can notice the memory usage needed for increasing number of data points. Around 15000 data points the physical limits of the current 32 bit desktop computers running a Windows operating system are met.

It is this physical limit that each computer today has that will force us to look for other methods to train kernel models on large data sets. According to Figure 1.6 it is not possible to train kernel models on data set larger then 15000 data points. Fortunately this situation is not that strict in practice. To overcome this problem there are two possible solutions.

The first solution is to store the necessary information outside the RAM memory of the computer such as the hard disk. Algorithms that use this strategy are called 'out-of-core' algorithms. In such situations the number of input-output I/O operations need to be reduced as much as possible. Therefore specific caching strategies become very important. But also the

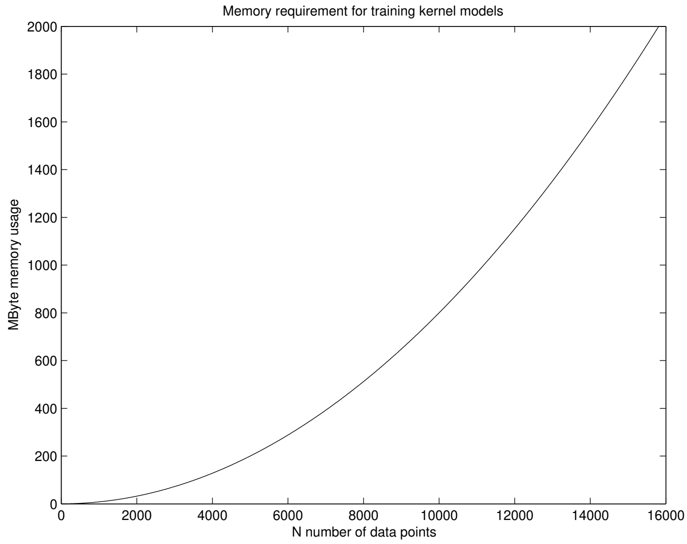

Figure 1.6: In this figure the memory usage, needed by the training process of kernel methods, is illustrated with increasing number of data points N.

disk space is not infinite and soon the maximum possible memory space will be reached.

A second solution uses a totally different approach. Instead of storing the whole information, the Hessian matrix, into the RAM memory of the computer it is also possible that one recomputes this information every time it is needed. In this case the memory bottleneck is transferred into a computational problem. Although it is theoretically possible to recompute all necessary information on demand, it soon becomes very impractical because of the enormous computation time needed.

In the remainder of this work we will formulate our solutions towards large scale data sets such that we do not have to apply the 'out-of-core' strategy. In some cases we will use a recomputation strategy. In general we will assume that the presented methods and algorithms can be implemented in such a way that all the necessary information can be stored into the RAM memory of the computer.

In the following section some rules of thumb will be given that will guide the reader towards a specific solution method corresponding to his problem as presented in this work.

1.3 Results presented in this work

1.3.1 Goals

The goal of this thesis is to study the effect of scalability on the design of kernel models. Hereby we compare the state-of-the-art algorithms and improve them with respect to their ability to learn from large data sets. Our purpose will be to modify the algorithms in such a way that they can be executed on a standard PC or low-end workstation. Hereby the constraining factor is their limited RAM memory. We will not specify strict limits on the number of data points that can or cannot be used since it depends on the hardware and software of the user. But instead during the text we will provide approximations of the computational and memory complexity of the algorithms.

1.3.2 Five Pillars

In this thesis we will give a summary of the literature and experiments we conducted concerning the training of kernel models on large data sets. In view of these experiences we will formulate different solutions and guidelines. The results can be formulated as five important pillars on which this thesis is built and which can be found in the following chapters of the work. These five pillars are:

1. model choice,
2. numerical methods used for training,
3. choice of the kernel function,
4. low rank approximations,
5. ensemble learning.

We will explain our results on the basis of these five pillars and give guidelines as shown in Figure 1.7. This 'decision tree' is based on the results achieved in this thesis. It can guide the reader towards the specific solution methodology he/she needs for the problem he/she want to solve. The problem is characterized as a classification, regression or time-series problem with N training points in a d dimensional input space using a kernel model with a linear or non-linear kernel. An assumption is that the user has a standard desktop computer with a RAM memory of about 256-512

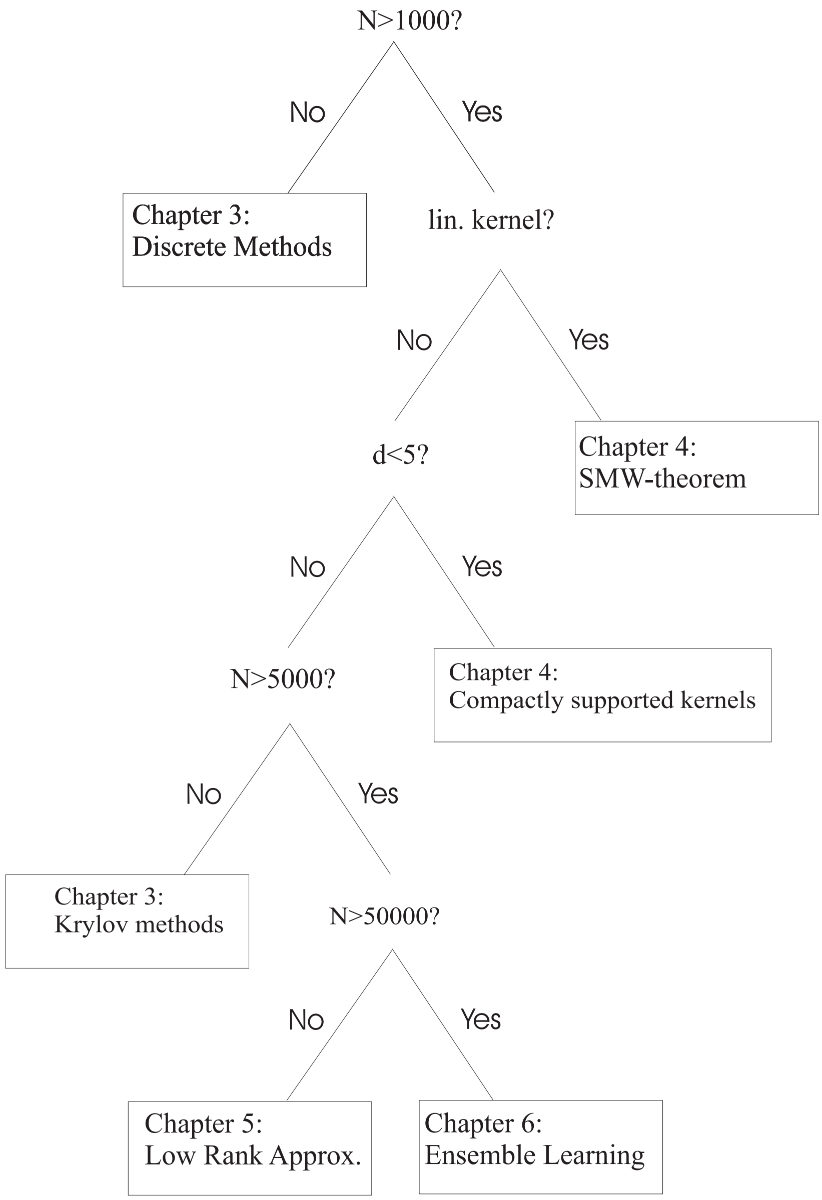

Figure 1.7: Guidelines about which methods or algorithms one can use if one faces a problem with N data points in a d dimensional input space using a kernel model with a linear or non-linear kernel.

MB and normal computational power that is current these days on an Intel Pentium or AMD 32 bit processor.

The following will motivate and explain the five pillars in more detail:

The model choice: A first step in the decision process for training kernel models is defining the actual kernel model one wants to use. Each kernel model has its own specific training procedure and its corresponding computational and memory complexity. In this thesis we will be mainly focussing on kernel models using a squared loss function like used in LS-SVM models and regularization networks. In the first chapter we will give the necessary mathematical background on these models. We will study the influence of the kernel function and its relation to the Mercer theorem and reproducing kernel Hilbert spaces. The derivation of these models will be done from both a regularization viewpoint and from an optimization viewpoint. We show that both explanations of these models, using a squared loss function, lead to a training process that consists of solving a linear system. Links with other kernel models like the support vector machine are explained and a comparison for training large data sets will be studied.

The numerical methods: In the first chapter we explained that model using a squared loss function are favorable because of the training procedure. Training these models on a data set of N points consists of solving a square linear system of dimension N. Therefore in the second chapter we will be studying the numerical methods that are best suited for solving this linear system. An overview and explanation of the different methods will be given together with guidelines how to implement them. Our attention will go to both direct and iterative methods. We will show why direct methods are favorable for smaller problems ($N<$ 1000) but are impractical for larger data sets. On the other hand iterative methods are more advisable for training sets until $N < 5000$. The main advantages of these methods are the fast convergence rates and the possibility to use intermediate solutions in case one has a time restriction on the training process. We will show experimentally that Krylov methods like conjugate gradient and block conjugate gradient outperform SOR based iterative methods. However, since these iterative methods still have a memory complexity of $\mathcal{O}(N^2)$ they have a limited usability. For problems where $N > 5000$ recomputation strategies can be applied to reduce the memory bottleneck. Since in these cases the memory bottleneck shifts to a computational bottleneck, reducing the number of computations also becomes essential. For this reason in Chapter 3 we will advise iterative methods with low computational complexity and memory complexity.

The choice of the kernel: At the basis of each kernel model lies the choice of the kernel function. Not only the choice of the kernel function has an impact on the generalization performance of the model, it also has its computational implications. In Chapter 4 we will discuss the rules that a good kernel function has to obey and formulate ways to construct new kernels. We will show how the kernel function has an impact on the structure of the matrix involved in the linear system for training the model. In the most general circumstances this matrix will be dense. We will study new compactly supported kernels that result in sparse matrices which need less memory to store. As indicated in Figure 1.7 these compactly supported kernels are advisable in problems with a low dimensional input $d < 5$. Other kernel functions (linear and separable) lead to matrices with good algebraic properties. For example, linear kernels and separable kernels have intrinsically a low rank. This can be exploited during the training process based on the Sherman-Morisson-Woodbury formula.

Low rank approximations: The next step is for data sets where $5000 < N < 50000$ using non-linear kernels (See Figure 1.7). In this case it is impossible to store the matrix and this matrix is too large to recompute in each step of the training procedure. Therefore we will need to define approximation methods that overcome the memory and computational bottleneck. These will be called low rank approximations. We will give an overview of different approximation methods and explain the links between them. Two groups can be identified. The first group defines approximations of the matrix involved in training process based on the Nyström approximation and the Cholesky factorization. The second group defines approximations at the model level. Here models with a reduced number of free parameters are defined based on optimal basis reduction methods or optimal prototype vector methods. We will focus on their computational and memory complexity and give advice for the best implementation methods and numerical routines that should be used. Experimental results are shown.

Ensemble Learning: Data sets where the number of data points exceeds 50000 using a non-linear kernel are difficult to train by only one model. In all the previously explained methods the goal was to train one model on the whole data set. A last methodology is based on the idea of 'divide and conquer'. This is done based on ensemble learning. In ensemble learning one combines different models trained on one data set. Most of the literature on ensemble learning uses this strategy to improve the performance of the individual models where each of them is trained on almost the whole original data set. In this work our objective will be to study this methodology for tackling the memory and computational problems kernel models face.

Instead of training one kernel model on the whole data set, we will train a collection or ensemble of models each on a subset of the original data set. These submodels will be combined afterwards to have a new ensemble model estimator which is trained on the whole data set.

We will give an introduction on different ensemble learning strategies and explain their usage towards training large data sets. Comparison on the computational complexity of the combined models will be made. Therefore we will give a literature overview of recent advances in ensemble learning with kernel models. To motivate our study we will zoom in on specific recent theoretical results on ensembles of kernel models.

In addition we will introduce a new ensemble learning methodology where the training of the individual elements of the ensemble is done in a coupled way. Using this coupled learning will lead to models which are more robust towards *outlier* submodels. This is achieved by an information exchange during training of the individual submodel caused by the coupling. The idea of coupled learning will be motivated based on the idea of coupled local minimizer and multitask learning. We will also show how this coupled learning can be used for transductive learning. All the results are experimentally verified.

1.4 Overview

1.4.1 Outline of the thesis

In this **Chapter 1** we gave a problem situation and motivation of this work. Based on some examples we showed what type of problems we will be able to tackle with the models that will be discussed in this work. We explained the goals of the thesis.

Chapter 2 will explain the theoretical framework on which kernel models are based. We will zoom in on concepts like regularization, generalization and learning. We will especially focus on the model formulation in the primal or dual space and the optimization process involved in training the models. Special attention is paid to Regularization Networks and Least Squares Support Vector Machines. Links with Support Vector Machines, Kriging, Gaussian Processes and kernel ridge regression will also be discussed.

In **Chapter 3** the numerical aspects involved in training LS-SVM models will be discussed. Since the training procedure of these models comes down to solving sets of linear equations, the latter will be handled in more detail. A comparison between direct and iterative methods like Cholesky factorization, successive overrelaxation and conjugate gradient will be given

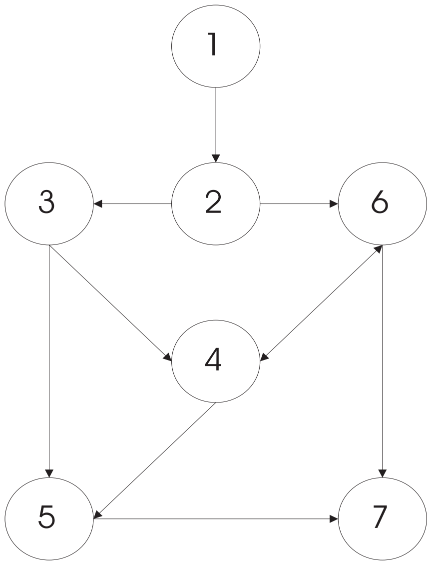

Figure 1.8: The structure of the thesis. This figure shows the different trajectories one can follow to read this thesis. Each circle indicates a chapter and the arrows pointing to a chapter explain what are the prerequisites that are necessary to read this chapters.

both theoretically and experimentally. Scalability towards large linear systems will be the measure for comparison. This chapter includes the earlier work [60] written by the author. Some of the results shown in this chapter were already included in the Matlab toolbox LS-SVMlab and were described in [104] and presented at NIPS 2002.

Chapter 4 gives an overview of the different types of kernel functions that are used in the SVM literature. Some general rules for constructing kernels are explained. The kernel functions are classified into three groups: stationary, locally stationary and nonstationary. Attention goes to the computational aspects of these kernels and we will provide general rules of thumb for certain classes of kernels. In this chapter also the effects of compactly supported kernels are studied. These kernels can lead to a reduction in computational complexity of the training phase. But they also have an influence on the generalization performance of the kernel model. This work has been published in [63].

In **Chapter 5** we will give an overview of different low rank approximation methods as presented in literature. We will study methods based on Nyström approximation and the incomplete Cholesky factorization. Links towards subspace methods in feature space, Radial Basis Networks and Fixed Size LS-SVM will be given. Suitably we will take a close look at their computational aspects and give some guidelines for best numerical routines that should be used.

Chapter 6 introduces an alternative method for tackling the scalability problem. The setup is not to train one model on the whole data set but to train a collection or ensemble of models. Each of the elements of the ensemble is trained on a subset of the original data set and the end results are combined in an ensemble model. In this chapter we give an overview of different strategies to construct an ensemble model. We will explain different ways of formulating an ensemble of learning algorithms and show several strategies to combine them. The focus will be on constructing ensembles of the kernel models that we have studied in the previous chapters.

An important part of this chapter will be devoted to a new strategy of coupled ensemble models developed by the author. This new methodology of constructing ensembles is based on the principle of coupled local minimizers. We will explain how this strategy leads to a more robust way of constructing ensembles and how one can achieve a transductive learning strategy in a straightforward way. Preliminary work on the subject of coupled ensemble learning was published in [62]. The extended results and links towards transductive learning are summarized in [61].

Chapter 7 concludes this work and presents some open problem for further research.

1.4.2 Contributions

In this section we will overview the main contributions made in this work:

- *Comparison of Numerical Procedures*: We make a profound study and comparison of different numerical procedure for training LS-SVMs and RNs. Implementation details and acceleration techniques are given for each of the algorithms. A special attention is paid to iterative solvers which have the best characteristics for our problems. The relation between the numerical stability and the influence of regularization is shown. These results are described in Chapter 3, which includes the earlier work [60] written by the author. Some of the results shown in this chapter were already included in the Matlab toolbox LS-SVMlab and were described in [104] and presented at NIPS 2002.

- *Compactly Supported Kernels:* In this thesis show that the RBF kernels are a special case of the Matèrn kernel. Therefor exist efficient compact formulation which remain positive definite. We propose the *compactly supported kernel* for sparsifying the Gram matrix. Computational and memory advantages resulting form this sparsification are explained. Additionally we show that the compactly supported kernel has a influence on the generalization performance of the learning model. These results are explained in Chapter 4 and led to the publication [63].

- *Low Rank Approximations:* We give a literature overview of the most popular low rank approximation methods. We divide them into two classes. We show that one approximations on the matrix level and on the model level. Links between those models will be shown. We will give implementation details and compare the numerical aspects of these approximation models. This is explained in Chapter 5.

- *Ensemble Learning and Coupled Learning:*We introduce the concept of ensemble learning for training large data sets. We motivate this form a 'divide and conquer' strategy. We will give an overview of the most recent advances in this domain. We introduce a new viewpoint on ensemble learning where we explain the different models as the result of a two step procedure. This new viewpoint introduces the concept of *Coupled Learning*. The idea originated from the 'coupled local minimizers' methods for solving non-convex optimization problems. In the Coupled Learning methodology the ensemble members are coupled during training based on a *coupling set*. In this way there is an information exchange between the models that leads to a better generalization performance. These results we published in [62].

- *Transductive Learning for Regression and Classification*: The concept of Coupled Learning can be extended towards a new form of *Transductive Learning*. This is achieved by constructing a coupling set using both training and test set. A strong advantage is that this method can be applied to both classification and regression problems. We will show that the idea of coupled learning has strong links with multitask learning and learning can be regarded as group regularization imposed on all the methods of the ensemble. The results of ensemble learning are presented in Chapter 6. This chapter includes the earlier work [61] in which results towards transductive learning are summarized.

1.4.3 Nomenclature

The first step to be taken in a learning task is to define a *model*. One uses an optimization algorithm to compute the optimal parameters of the model. This is the *training phase* of the model. Although these are two different and separate steps in the process of learning, they will always come together. Therefore we will not always make a distinction between the terms "model", "learning model" and "learning algorithm". Our main focus will go to the algorithmic complexity of the training phase of the models.

Chapter 2

Theory and Implementation of Least Squares Support Vector Machines and Regularization Networks

"You don't understand anything until you learn it more than one way."
Marvin Minsky.

In this chapter we will explain the theoretical framework on which kernel models are based. We will study concepts like regularization, generalization and learning. The mathematical concept of reproducing kernel Hilbert spaces will be introduced and its link to mapping data into a feature space will be explained. We will focus on the model formulation in the primal or dual space and the corresponding optimization process involved in the training process of these models will be studied. Special attention is paid to regularization networks and least squares support vector machines. We will explain what the implications are if one trains these models with large data sets both from learning and optimization perspective.For the traditional support vector machines the state-of-the-art solution methods for large scale applications are explained and compared.

2.1 Introduction

In this thesis we consider a statistical learning framework as proposed by Vapnik ([146],[147]). We concentrate on learning from data with input-output patterns, called *supervised learning*. We assume that $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} \subseteq \mathbb{R}$ are two sets of random variables that are related by a probabilistic relationship. This means that an element of $\mathbf{x} \in \mathcal{X}$ does not determine uniquely an element of $y \in \mathcal{Y}$, but rather a probability distribution on $\mathcal{Y}$. This relationship is formalized by assuming a probability distribution $p(\mathbf{x}, y)$ over the set $\mathcal{X} \times \mathcal{Y}$. Under very general conditions the relations between the joint and conditional probabilities can be written as:

$$p(\mathbf{x}, y) = p(y|\,\mathbf{x})p(\mathbf{x}). \tag{2.1}$$

This joint distribution $p(\mathbf{x}, y)$ is fixed but unknown since we are given only N samples from it. The data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y}$, sampled independently and identically distributed according to $p(\mathbf{x}, y)$, is called the *training set*. Although these training points are the result of a random sampling according to the joint distribution $p(\mathbf{x}, y)$, we assume that the $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ could be measured accurately with an error that is small compared to the accuracy by which the $y_1, y_2, \ldots, y_N$ are measured. Learning now consists of finding a function $f : \mathcal{X} \to \mathcal{Y}$, given the data set $\mathcal{D}$ that can be used to predict a value y given an $\mathbf{x} \in X$. In statistical terms f approximates $E[y\,|\mathbf{x}]$. This function f is called an *estimator or hypothesis.*

This is approached in two steps:

- Define a set of possible candidate hypotheses f. This set is called the *hypothesis space* $\mathcal{F}$.
- Define an error criterion by which the best candidate is selected within this hypothesis space $\mathcal{F}$. Since one only disposes of the set of examples $\mathcal{D}$, this criterion is based on the minimization of a predefined error measure of the function or hypothesis f over the training set.

This error criterion is defined as a *risk functional* $I : \mathcal{F} \to \mathbb{R}$, which measures the average error of a hypothesis. For each f it gives its theoretical or *expected risk:*

$$I\,[f] = \iint\limits_{\mathcal{X}\times\mathcal{Y}} V\,(y, f\,(\mathbf{x}))\,p(\mathbf{x}, y)d\mathbf{x}dy, \tag{2.2}$$

based on a chosen loss function $V\,(y, f\,(\mathbf{x}))$, which measures the error made when y is predicted by $f\,(\mathbf{x})$. Therefore, the choice of the loss function determines the learning scheme.

Once the loss function is defined, the next step is to find the function f that minimizes the expected risk in the set of possible hypotheses $\mathcal{F}$. This optimal estimator on a sufficiently large class of functions $\mathcal{F}$:

$$f^* = \arg\min_{f \in \mathcal{F}} I[f] \tag{2.3}$$

is often called the *target function* f^*.

Since we only have the training set $\mathcal{D}$ available and since the joint probability $p(\mathbf{x}, y)$ is unknown, we can not compute the expected risk directly. As a result of that, we cannot compute the target function f^*. One of the possibilities to overcome this shortcoming was proposed by Vapnik. By using another *induction principle* we are able to learn with a limited set of training points. One uses stochastic approximation of the expected risk using the finite data set $\mathcal{D}$. This approximation is called the *empirical risk* and is defined as:

$$I_{emp}[f, \mathcal{D}] = \frac{1}{N} \sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)). \tag{2.4}$$

Finding the optimal hypothesis that minimizes this functional over the hypothesis space is called *empirical risk minimization.* The question now is whether the minimum of the expected risk and the solution of the empirical risk minimization lead to the same solution in a probabilistic sense. It is proven by Vapnik ([146],[147]) that empirical risk minimization can be consistent if one restricts the hypothesis space. It states that the empirical risk minimization is nontrivial consistent or in other words the following uniform law of large numbers is a necessary and sufficient condition (for a detailed description see [147]):

$$\lim_{N \to \infty} P\left(\sup_{f \in \mathcal{F}} (I[f] - I_{emp}[f, \mathcal{D}]) > \epsilon \right) \to 0, \forall \epsilon > 0. \tag{2.5}$$

Now, since solving the minimization of the empirical risk minimization is ill-defined as such, we will limit the set of hypotheses when minimizing the empirical risk $I_{emp}[f, \mathcal{D}]$. This restriction on the hypothesis space, by demanding an extra smoothness of the functions, is to avoid so called overfitting. In the case of overfitting the estimated function gives a very good performance on the data it has learned form the data set but it has a extremely poor generalization performance in the regions of the input space $\mathcal{X}$ where it was not trained. This leads to situations where the minimum of the empirical risk can be very small but still the expected risk, in which we are really interested, is large.

The theory of kernel models will demand that the hypothesis space is restricted to a bounded convex subset of a *Reproducing Kernel Hilbert Space* $\mathcal{H}$ (RKHS) as induced by a kernel k. This kernel k is a symmetric, real-valued, positive definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. This results in an empirical risk minimization with regularization also known as *Ivanov Regularization:*

$$\min_{f \in \mathcal{H}, \|f\|_k^2 \leq r^2} \quad \frac{1}{N} \sum_{i=1}^{N} V\left(y_i, f\left(\mathbf{x}_i\right)\right), \tag{2.6}$$

where $\|\mathbf{.}\|_k$ is the norm in the RKHS and $r \in \mathbb{R}$. (For a more elaborate explanation of a Reproducing Kernel Hilbert Space and the norm in this space see the next section.)

In this setup Statistical Learning Theory provides us with probabilistic bounds between the expected and the empirical risk based on the size of the training set N and the 'capacity' h of the hypothesis space, a combinatorial measure for the model complexity. One of the most general capacity measures is the VC (Vapnik-Chervonenkis) dimension. This VC dimension for a set of functions or hypotheses is the maximum number of training points that can be separated by the members of the set of functions for each possible labelling of the training points. For example, the set of linear functions in $\mathbb{R}^d$ has a VC-dimension $h = d + 1$ since is is not possible to separate more then $d + 1$ points by a linear hyperplane in $\mathbb{R}^d$ for each possible labelling. Using this capacity h, the bound has the following general form [42]: with a probability at least η

$$I[f] \leq I_{emp}\left[f, \mathcal{D}\right] + \zeta\left(\sqrt{\frac{h}{N}}, \eta\right), \tag{2.7}$$

where ζ is an increasing function of $\frac{h}{N}$ and η. It is this relation that plays an important role in the use of the learning theory. This formula states that it is not only important to reduce the empirical risk, but one should also take care about the hypothesis used. Remember that the VC dimension is a measure for the separation capability of a set of functions. But it can also be understood as the flexibility in the functions. The higher this flexibility or capacity is, the easier it will be to reduce the empirical risk. This means, however that the right-hand side will also increase. So, it becomes more likely that the discrepancy between the empirical and expected risk increases. Hence there is a constant trade-off between empirical risk and the capacity of the hypothesis space.

To overcome this problem Vapnik formulated the *Structural Risk Minimization*. The idea of structural risk minimization is to define a nested sequence of hypothesis spaces $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_q$ where each hypothesis space $\mathcal{H}_p$, $p = 1 \ldots q$ has a finite capacity h_p larger than all the previous spaces such that $h_1 \leq h_2 \leq \ldots \leq h_q$. The strategy of *Structural Risk Minimization* now is with increasing complexity of the Hypothesis space to minimize the empirical risk. The *structural risk* as given by the right-hand side of inequality (2.7) can be controlled by the choice of $\mathcal{H}_p$. In this way one can get an optimal trade-off between the empirical and structural risk. Notice that the idea of structural risk minimization has a strong resemblance with the concept of bias-variance trade-off as known from statistics and often applied on neural networks [7]. Also here a balance has to be found between the model complexity and the error minimization. We will come back to this later.

If this is applied to the Ivanov Regularization, we end-up with the following scheme:

$$\min_{f \in \mathcal{H}, \|f\|_k^2 \leq a_p^2} \quad \frac{1}{N} \sum_{i=1}^{N} V\left(y_i, f\left(\mathbf{x}_i\right)\right), \tag{2.8}$$

where the sequence $\{a_p\}_{p=1}^{q} \subset \mathbb{R}$ is a monotonically increasing sequence of real constants. The nested sequences of hypothesis spaces in the Reproducing Kernel Hilbert Space is constructed by bounding the norm in this space, induced by the kernel such that $\mathcal{H}_p = \{f \in \mathcal{H} : \|f\|_k^2 \leq a_p^2\}$.

It can be shown that optimizing this formulation of Ivanov Regularization is equivalent to minimizing:

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{i=1}^{N} V\left(y_i, f\left(\mathbf{x}_i\right)\right) + \gamma_p \left(\|f\|_k^2 - a_p^2\right), \tag{2.9}$$

with respect to $f \in \mathcal{H}$ and maximizing with respect to $\gamma_p \geq 0$. Following the structural risk minimization, this optimization should be done for increasing a_p thereby constantly monitoring the balance between the structural and empirical risk. However, this is rarely done in practice because it is computationally very intensive. Instead the problem (2.8) is reformulated to a closely related *Tikhonov regularization* problem ([137],[42]):

$$\min_{f \in \mathcal{H}} \quad \frac{1}{N} \sum_{i=1}^{N} V\left(y_i, f\left(\mathbf{x}_i\right)\right) + \frac{1}{2\gamma} \|f\|_k^2. \tag{2.10}$$

The Tikhonov regularization smoothly trades off the structural risk $\|f\|_k^2$ and the empirical risk, controlled by the *regularization parameter* γ. Within the Statistical Learning framework this regularization parameter can be seen as a penalty for functions with a high capacity; the larger γ, the smaller the RKHS norm of the solution will be. This parameter, similar to the optimal a_p for Ivanov regularization (2.8), is not known beforehand. Applying the structural risk minimization for Tikhonov regularization is done by using various values γ and picking the optimal one by using a technique like cross-validation ([7],[150]).

Notice that until now we did not specify the loss function $V(y, f(\mathbf{x}))$. There exist many different choices of loss function each with their own characteristics. For regression the most used are (see also Figure 2.1):

- The L_2 norm or squared loss function : $V(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$,
- The L_1 norm loss function: $V(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$,
- The Vapnik epsilon-insensitive loss function:

$$V(y, f(\mathbf{x})) = \begin{cases} |y - f(\mathbf{x})| & \text{if} \quad |y - f(\mathbf{x})| \geq \epsilon, \\ 0 & \text{if} \quad |y - f(\mathbf{x})| < \epsilon, \end{cases} \tag{2.11}$$

- The Huber Loss function:

$$V(y, f(\mathbf{x})) = \begin{cases} \epsilon\,|y - f(\mathbf{x})| - \frac{\epsilon^2}{2} & \text{if} \quad |y - f(\mathbf{x})| \geq \epsilon, \\ \frac{1}{2}(y - f(\mathbf{x}))^2 & \text{if} \quad |y - f(\mathbf{x})| < \epsilon. \end{cases} \tag{2.12}$$

In the remainder of this thesis, we will mainly us the quadratic loss function. It should be noted that the concepts of structural risk minimization based on the VC dimension are not straightforwardly applicable to the quadratic loss function. Evgeniou [44] therefore extended the formulation of structural risk minimization based on a different capacity criterion for the hypothesis space, the V_γ-dimension[1]. With this extension the idea of structural risk minimization is still applicable as formulated.

In the previous paragraphs we explained a model selection formalism for regression problems. Often the target or output values $\mathcal{Y}$ do not span the whole continuous space $\mathbb{R}$. In many problems $\mathcal{Y}$ is a discrete subset; for example $\mathcal{Y} = \{-1, 1\}$ or $\mathcal{Y} = \{0, 1, 2, ..., z\}$ where $z \in \mathbb{N}_0$. The first is called a binary classification problem. The second is a multi-class problem. Since most multi-class classification problems are solved by using binary classification models with the appropriate coding-scheme ([144], [123]), we

[1]For an exact definition of this V_γ we refer to [44]. The γ in this definition is not related to the regularization parameter used in this work.

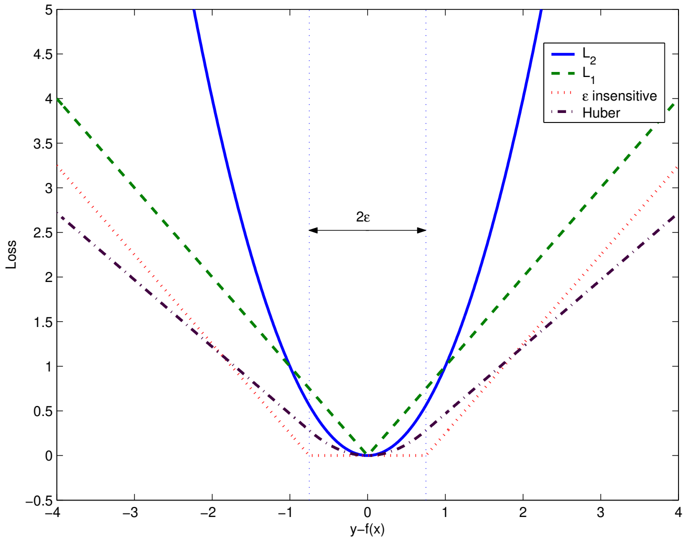

Figure 2.1: Loss functions considered for kernel models. We see the squared loss (L_2), the L_1 loss function, the Vapnik epsilon-insensitive loss function and the Huber loss function.

will only focus on the first class of problems. The goal is to find a function $g^* : \mathcal{X} \subseteq \mathbb{R}^d \rightarrow \{-1, 1\}$ that minimize the probability of $g^*(\mathbf{x}) \neq y$, i.e. to find a function g^* such that

$$P\left(g^*(\mathbf{x}) \neq y\right) = \min_{g:\mathcal{X}\rightarrow\{-1,1\}} \quad P\left(g(\mathbf{x}) \neq y\right), \tag{2.13}$$

where g^* is called the Bayes decision function, and $P\left(g(\mathbf{x}) \neq y\right)$ the probability of misclassification. The binary function g can also be defined as

$$g(\mathbf{x}) = \operatorname{sign}(f(\mathbf{x})), \tag{2.14}$$

where f is a real function on $\mathcal{X}$ as used before. The difference is that the expected and empirical error are now based on indicator functions [147]. In [113] it was shown that this leads to a similar formulation as the Tikhonov regularization with different possible loss functions as shown above. Therefore we will restrict ourselves to this formulation of Tikhonov regularization (2.10) for both regression and classification problems.

2.2 Reproducing kernel Hilbert spaces and Mercer kernels

In the previous section we mentioned that in order to restrict our hypothesis space we demand that all the functions f are elements of a bounded Reproducing kernel Hilbert space. However, what does this mean?

In this section we will use the following notations and definitions. A function f over domain $\mathcal{X}$ is given as $f = f(\cdot)$, while the evaluation of this function in a point $\mathbf{x} \in \mathcal{X}$ is $f(\mathbf{x})$. A symmetric, real-valued function $k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ of two variables is said to be *positive definite* if for any $a_1, a_2, \ldots, a_n \in \mathbb{R}$ and $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathcal{X}$:

$$\sum_{i,j=1}^{n} a_i a_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0. \tag{2.15}$$

If this inequality is always strictly positive, it is called a *strictly positive definite kernel*.

A *Reproducing Kernel Hilbert Space* $\mathcal{H}$, abbreviated as RKHS, is a set of functions $f = f(\cdot)$ defined over a bounded domain $\mathcal{X} \subset \mathbb{R}^d$. This set is a Hilbert space and has the property that for each $\mathbf{x} \in \mathcal{X}$, the evaluation functionals $\mathcal{F}_{\mathbf{x}}$, defined as

$$\mathcal{F}_{\mathbf{x}}[f] = f(\mathbf{x}), \ \forall f \in \mathcal{H}, \tag{2.16}$$

are linear and bounded. Bounded means that there exists a $U = U_{\mathbf{x}} \in \mathbb{R}^+$ such that $|\mathcal{F}_{\mathbf{x}}[f]| = |f(\mathbf{x})| \leq U \|f\|_k$ for all f in the RKHS, where $\|f\|_k$ is the norm defined by the inner product $\langle \cdot,\cdot \rangle_{\mathcal{H}}$ in the Hilbert space[2].

It can be proven ([89], [150]) that for each RKHS there exists a unique *positive definite function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that $k(\mathbf{x},\cdot) \in \mathcal{H}, \forall \mathbf{x} \in \mathcal{X}$, and has the following property:

$$f(\mathbf{x}) = \langle f(\cdot), k(\mathbf{x},\cdot) \rangle_{\mathcal{H}}, \ \forall f \in \mathcal{H}. \tag{2.17}$$

This kernel is called a *reproducing kernel* since:

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle k(\mathbf{x},\cdot), k\left(\mathbf{x}',\cdot\right) \right\rangle_{\mathcal{H}}, \ \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}. \tag{2.18}$$

A second class of important kernels are the *Mercer kernels*. These kernels follow from the Mercer Theorem.

[2] Later in this section we will give define the inner produkt $\langle \cdot,\cdot \rangle_{\mathcal{H}}$ in more detail.

Theorem 1 ***Mercer Theorem*** *([86],[28]): Let $\mathcal{X}$ be a compact subset of $\mathbb{R}^d$. Suppose $k(\cdot,\cdot)$ is a continuous symmetric function such that the integral operator $T_k : L^2(\mathcal{X}) \to L^2(\mathcal{X})$,*

$$T_k f(\mathbf{x}) = \int_{\mathcal{X}} k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') d\mathbf{x} \tag{2.19}$$

is positive, that is

$$\iint_{\mathcal{X}\times\mathcal{X}} k(\mathbf{x}, \mathbf{x}') g(\mathbf{x}) g(\mathbf{x}') d\mathbf{x}d\mathbf{x}' \geq 0, \tag{2.20}$$

for all $g(\cdot) \in L^2(\mathcal{X})$ (square integrable functions). We can expand $k(\mathbf{x}, \mathbf{x}')$ in a uniform convergent series (on $\mathcal{X} \times \mathcal{X}$):

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d_{\mathcal{H}}} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}'), \tag{2.21}$$

where $\{\phi_i\}_{i=1}^{d_{\mathcal{H}}} \in L^2(\mathcal{X})$ is an orthogonal set of eigenfunctions of the integral operator T_k normalized in such way that $\|\phi_i\|_{L^2} = 1$ [3]. Correspondingly, the $\{\lambda_i\}_{i=1}^{d_{\mathcal{H}}}$ are the positive associated eigenvalues of integral operator T_k where $d_{\mathcal{H}}$, the dimension of this Hilbert space, is either $d_{\mathcal{H}} \in \mathbb{N}$ or $d_{\mathcal{H}} = \infty$.

The requirements in this theorem are often called the *Mercer conditions*.

Because of the positivity condition (2.20) it is easy to see that Mercer kernels are also positive definite. This can be seen intuatively by taking $g(\cdot)$ as a weighted sum of delta functions. It is even so, that this positivity condition is an alternative condition for positive definiteness. If a function does not fulfill this positivity condition, it will not be a positive definite function.

Now, for every Mercer kernel $k(\cdot,\cdot)$ defined over the domain $\mathcal{X} \subseteq \mathbb{R}^d$, there exists an RKHS $\mathcal{H}$ of functions defined over $\mathcal{X}$ for which $k(\cdot,\cdot)$ is the reproducing kernel. We already mentioned that $k(\cdot,\cdot)$ is positive definite. We will now show how one can construct the RKHS.

[3]Two functions $f, g \in L^2(\mathcal{X})$ are orthogonal if $\int_{\mathcal{X}} f(\mathbf{x}) g(\mathbf{x}) d\mathbf{x} = 0$. A functions $f \in L^2(\mathcal{X})$ is normalized in $L^2(\mathcal{X})$ if $\|f\|_{L^2} = \int_{\mathcal{X}} [f(\mathbf{x})]^2 d\mathbf{x} = 1$.

Let us construct a Hilbert space $\mathcal{H}$ of functions of the form:

$$f(\cdot) = \sum_{i=1}^{d_{\mathcal{H}}} c_i \phi_i(\cdot), \tag{2.22}$$

where the coefficients $c_i \in \mathbb{R}$ and $\{\phi_i\}_{i=1}^{d_{\mathcal{H}}}$ and $\{\lambda_i\}_{i=1}^{d_{\mathcal{H}}}$ are the eigenfunctions and eigenvalues of the integral operator corresponding to the Mercer kernel. The inner product of two functions $f(\cdot) = \sum_{i=1}^{d_{\mathcal{H}}} c_i \phi_i(\cdot)$ and $g(\cdot) = \sum_{i=1}^{d_{\mathcal{H}}} d_i \phi_i(\cdot)$ in Hilbert space $\mathcal{H}$ is defined as:

$$\begin{aligned}\langle f(\cdot), g(\cdot)\rangle_{\mathcal{H}} &= \left\langle \sum_{i=1}^{d_{\mathcal{H}}} c_i \phi_i(\cdot), \sum_{i=1}^{d_{\mathcal{H}}} d_i \phi_i(\cdot) \right\rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{d_{\mathcal{H}}} \frac{c_i d_i}{\lambda_i}.\end{aligned} \tag{2.23}$$

It can be proven that this Hilbert space is an RKHS with the reproducing kernel $k(\cdot,\cdot)$ as defined above. This can be seen easily since:

$$\begin{aligned}\langle f, k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} &= \left\langle \sum_{i=1}^{d_{\mathcal{H}}} c_i \phi_i(\cdot), \sum_{j=1}^{d_{\mathcal{H}}} \lambda_j \phi_j(\mathbf{x}) \phi_j(\cdot) \right\rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{d_{\mathcal{H}}} \sum_{j=1}^{d_{\mathcal{H}}} c_i \lambda_j \phi_j(\mathbf{x}) \langle \phi_i(\cdot), \phi_j(\cdot)\rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{d_{\mathcal{H}}} \frac{c_i \lambda_i \phi_i(\mathbf{x})}{\lambda_i} \\ &= f(\mathbf{x}).\end{aligned} \tag{2.24}$$

From this relation follows that the Mercer kernel k is a reproducing kernel in this Hilbert space as defined in (2.17). Remarkably, the converse also holds ([28], [150], [40]). For any RKHS the corresponding reproducing kernel is also a Mercer kernel and therefore an existence of $\{\phi_i\}_{i=1}^{d_{\mathcal{H}}}$ and $\{\lambda_i\}_{i=1}^{d_{\mathcal{H}}}$ are guaranteed[4].

This leads to the following equivalence:

[4]Notice hereby that the number $d_{\mathcal{H}}$ of eigenfunctions ϕ_i and eigenvalues λ_i is possibly but not necessarily infinite for a kernel.

- choosing an RKHS
- choosing a set ϕ_i and λ_i.
- choosing a reproducing kernel k.

In the previous section, we explained that the methodology of the structural risk minimization could be achieved by choosing a nested sequence of hypotheses spaces $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_q$, where the increase in capacity was regulated by relaxing the bound on the norm of the functions f in the RKHS. What is this norm in the RKHS? The norm in the RKHS is the norm as induced by the inner product in the Hilbert space $\mathcal{H}$ such that $\|\cdot\|_k = \sqrt{\langle\cdot,\cdot\rangle_{\mathcal{H}}}$. Note that the notation $\|\cdot\|_k$ is used in literature to reflect the norm in the RKHS induced by the kernel k. According to the notation used above, for $f(\cdot) = \sum_{i=1}^{d_{\mathcal{H}}} c_i \phi_i(\cdot)$:

$$\|f\|_k^2 = \sum_{i=1}^{d_{\mathcal{H}}} \frac{c_i^2}{\lambda_i}. \tag{2.25}$$

It is proven in [150] that f is an element of an RKHS if and only if $\|f\|_k^2 = \sum_{i=1}^{d_{\mathcal{H}}} \frac{c_i^2}{\lambda_i} < \infty$.

In [40] it is proven that the capacity of the set of functions $\left\{ f \in \mathcal{H} \,\middle|\, \|f\|_k^2 < a \right\}$ depends on a, with a a positive constant. By relaxing the norm $\|f\|_k^2 < a_p$ for $\{a_p\}_{p=1}^q$ a monotonically increasing sequence of positive constants, a nested sequence of hypotheses spaces $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_q$ is created in the RKHS $\mathcal{H}$ with increasing capacity. This is exactly what we wanted to achieve in order to implement the structural risk minimization.

We end this chapter by giving some commonly used kernels:

- linear kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- polynomial kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = \left(\sigma + \mathbf{x}_i^T \mathbf{x}_j\right)^d$; $d = 1, 2, \ldots$ and $\sigma \in \mathbb{R}_0^+$
- radial basis function: $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{\sigma^2}\right)$; $\sigma \in \mathbb{R}_0^+$
- MLP-kernels: $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\kappa \mathbf{x}_i^T \mathbf{x}_j + \delta\right)$; $\kappa, \delta \in \mathbb{R}$. [5]

[5] Notice that this kernel does not satisfy the Mercer condition for all values of $\kappa, \delta \in \mathbb{R}$. (see also [17, p.21])

In Chapter 4 we will give a more elaborate classification of the possible kernels.

2.3 The reproducing and the Mercer kernel map

If we now assume that k is a positive definite kernel, we can show that it defines a so called *Reproducing Feature Space* $\mathcal{F}_r$, where this kernel represents an inner product between the elements of this feature space. The relation between the input space $\mathcal{X}$ and the feature space is given by the following mapping:

$$\varphi_r : \mathcal{X} \subseteq \mathbb{R}^d \rightarrow \mathcal{F}_r : \mathbf{x} \rightarrow k(\mathbf{x}, \boldsymbol{\cdot}). \tag{2.26}$$

This mapping, called the *Reproducing Kernel Map* φ_r associates with each input point $\mathbf{x} \in \mathcal{X}$ a function $\varphi_r(\mathbf{x})(\boldsymbol{\cdot}) : \mathcal{X} \subseteq \mathbb{R}^d \rightarrow \mathbb{R} : \mathbf{x}' \rightarrow k(\mathbf{x}, \mathbf{x}')$. (Notice that in the remainder of this work we will simplify the notation by $\varphi_r(\mathbf{x})(\boldsymbol{\cdot}) = \varphi_r(\mathbf{x})$). The feature space $\mathcal{F}_r$ is the image of the input space $\mathcal{X}$ under this mapping φ_r.

This feature space $\mathcal{F}_r$ is a vector space where the elements are linear combinations of the form:

$$\begin{aligned} f(\boldsymbol{\cdot}) &= \sum_{i=1}^{m} \alpha_i \varphi_r(\mathbf{x}_i) \\ &= \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \boldsymbol{\cdot}), \end{aligned} \tag{2.27}$$

with $m \in \mathbb{N}_0$, $\alpha_i \in \mathbb{R}$ and $\{\mathbf{x}_i\}_{i=1}^m \subseteq \mathcal{X}$. The inner product $\langle \boldsymbol{\cdot}, \boldsymbol{\cdot} \rangle_{\mathcal{F}_r}$ in the feature space $\mathcal{F}_r$ of $f(\boldsymbol{\cdot})$ and the function $g(\boldsymbol{\cdot}) = \displaystyle\sum_{j=1}^{m'} \beta_j k(\mathbf{x}'_j, \boldsymbol{\cdot})$ where $m' \in \mathbb{N}_0$, $\beta_j \in \mathbb{R}$ and $\{\mathbf{x}'_j\}_{j=1}^{m'} \subseteq \mathcal{X}$ is defined as

$$\langle f(\boldsymbol{\cdot}), g(\boldsymbol{\cdot}) \rangle_{\mathcal{F}_r} = \sum_{i=1}^{m} \sum_{j=1}^{m'} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}'_j). \tag{2.28}$$

(For proof of the bilinearity of this inner product see [119, p.33]). Based on this inner product one can define a norm $\|\boldsymbol{\cdot}\|_{\mathcal{F}_r}$ in this feature space $\mathcal{F}_r$ induced by the kernel k as $\|\boldsymbol{\cdot}\|_{\mathcal{F}_r} = \sqrt{\langle \boldsymbol{\cdot}, \boldsymbol{\cdot} \rangle_{\mathcal{F}_r}}$. For each element $f(\boldsymbol{\cdot}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \boldsymbol{\cdot})$ of the feature space $\mathcal{F}_r$ the squared norm in this space is equal to:

$$\|f\|_{\mathcal{F}_r}^2 = \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j). \tag{2.29}$$

This formulation will be often used in the next sections.

From these definitions it can be seen that

$$\langle f(\cdot), k(\mathbf{x},\cdot)\rangle_{\mathcal{F}_r} = f(\mathbf{x}), \tag{2.30}$$

so the positive definite kernel is a reproducing kernel in this space. Therefore it also holds that $\langle k(\mathbf{x},\cdot), k(\mathbf{x}',\cdot)\rangle_{\mathcal{F}_r} = k(\mathbf{x},\mathbf{x}')$ from which it becomes clear that this reproducing kernel can be seen as an inner product in this *feature space* $\mathcal{F}_r$:

$$\left\langle \varphi_r(\mathbf{x}), \varphi_r(\mathbf{x}')\right\rangle_{\mathcal{F}_r} = k(\mathbf{x},\mathbf{x}'). \tag{2.31}$$

So, the space of functions $\mathcal{F}_r$ of the form (2.27) induced by its kernel k is also an RKHS and can be used as a possible hypothesis space. Therefore the inner product in this feature space $\mathcal{F}_r$ is similar as the inprodukt/norm in the RKHS $\mathcal{H}$ such that: $\langle f(\cdot), g(\cdot)\rangle_{\mathcal{F}_r} = \langle f(\cdot), g(\cdot)\rangle_{\mathcal{H}}$ and $\|f\|_{\mathcal{F}_r} = \|f\|_k$.

A second way to show the relation between kernels and inner products is based on the Mercer theorem. As was shown earlier, there is a set of orthogonal eigenfunctions $\{\phi_i\}_{i=1}^{d_{\mathcal{H}}}$ and eigenvalues $\{\lambda_i\}_{i=1}^{d_{\mathcal{H}}}$ to every positive definite kernel. These eigenfunctions can be used to define the elements of a $d_{\mathcal{H}}$-dimensional vector associated to each element of $\mathcal{X}$. In this way we define a mapping as follows:

$$\varphi_m : \mathcal{X} \subseteq \mathbb{R}^d \to \mathcal{F}_m : \mathbf{x} \to [\sqrt{\lambda_1}\phi_1(\mathbf{x}) \ldots \sqrt{\lambda_{d_{\mathcal{H}}}}\phi_{d_{\mathcal{H}}}(\mathbf{x})]^T. \tag{2.32}$$

The *Mercer Kernel feature space* $\mathcal{F}_m$, as target of the *Mercer Kernel Map* φ_m, is a $d_{\mathcal{H}}$-dimensional vector space. The dimension $d_{\mathcal{H}}$, called *the dimension of the feature space*, and it is determined by the choice of the kernel. The inner product in this vector space is given by

$$\begin{aligned}\left\langle \varphi_m(\mathbf{x}), \varphi_m(\mathbf{x}')\right\rangle_{\mathcal{F}_m} &= \sum_{i=1}^{d_{\mathcal{H}}}\lambda_i\phi_i(\mathbf{x})\,\phi_i(\mathbf{x}') \\ &= k(\mathbf{x},\mathbf{x}')\end{aligned} \tag{2.33}$$

where the last equality follows from the Mercer theorem 1. With this inner product it can be shown that the vector space $\mathcal{F}_m$ is a Hilbert space. Furthermore, this shows that the kernel reflects the inner product of the mapping of two vectors in this feature space $\mathcal{F}_m$.

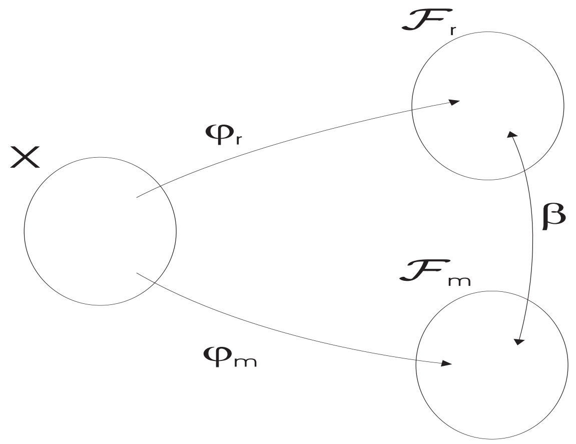

Figure 2.2: The feature spaces induced by the kernels. The reproducing kernel map φ_r maps the input space $\mathcal{X}$ into the feature space $\mathcal{F}_r$ and the Mercer kernel map maps it into the feature space $\mathcal{F}_m$. Between both feature spaces there exists an isometric isomorfism β [119].

It should be noticed that the *Mercer Kernel Feature Space* $\mathcal{F}_m$ and the *Reproducing Feature Space* $\mathcal{F}_r$ are different. However, between any two separable Hilbert spaces there exists an isometric isomorphism [119, p.37]. So, there exists a one to one mapping $\beta : \mathcal{F}_m \rightarrow \mathcal{F}_r$ between the *Mercer Kernel Feature Space* $\mathcal{F}_m$ and the *Reproducing Feature Space* $\mathcal{F}_r$.

Therefore, the main message in the above reasoning is that for each positive definite kernel $k(\cdot,\cdot)$ there is an interpretation as being an inner product in another vector space $\mathcal{F}$; $k(\mathbf{x},\mathbf{x}') = \langle\varphi(\mathbf{x}), \varphi(\mathbf{x}')\rangle_{\mathcal{F}}$, where φ is the corresponding mapping between the input domain $\mathcal{X}$ and this induced feature space $\mathcal{F}$. A very important aspect of this relation is that in the feature space the mapped data is of dimension $d_{\mathcal{H}}$, which can be infinite dimensional. However, the inner product of two mapped data points $\varphi(\mathbf{x})$ and $\varphi(\mathbf{x}')$ can easily be computed by the kernel $k(\mathbf{x},\mathbf{x}') \in \mathbb{R}$. So, each model or algorithm defined in the feature space based on inner products can easily be expressed in terms of kernels. It is even not necessary to know the exact expression of the mapped data points. Using a model, based on a specific positive definite kernel, induces automatically a corresponding mapping of the input space into a high dimensional feature space.

Until now we showed how the Feature map can be constructed from the kernel. But also the converse holds. Assume we have a mapping φ from the input space $\mathcal{X}$ into an inner product space. Because a norm is always nonnegative, a kernel defined as $k(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle$ is always positive definite. This is proven as follows, $\forall a_1, a_2, \ldots, a_n$ and $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathcal{X}$:

$$\begin{aligned}\sum_{i,j=1}^{n} a_i a_j k(\mathbf{x}_i, \mathbf{x}_j) &= \left\langle \sum_{i=1}^{n} a_i \varphi(\mathbf{x}_i), \sum_{j=1}^{n} a_j \varphi(\mathbf{x}_j) \right\rangle_{\mathcal{F}} \\ &= \left\| \sum_{i=1}^{n} a_i \varphi(\mathbf{x}_i) \right\|_{\mathcal{F}}^2 \geq 0. \qquad (2.34)\end{aligned}$$

It is this duality between kernels and Feature spaces that gives rise to the so called *Kernel Trick [119]: "Given a model or algorithm which is formulated in terms of a positive definite kernel $k(\cdot,\cdot)$, one can construct an alternative algorithm by replacing $k(\cdot,\cdot)$ by another positive definite kernel $k'(\cdot,\cdot)$."*

With respect to the previous, a model or algorithm working with the input data set $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathcal{X}$ and formulated in terms of a kernel $k(\cdot,\cdot)$ can be understood as operating on the mapped vectorial data $\varphi(\mathbf{x}_1), \varphi(\mathbf{x}_2), \ldots, \varphi(\mathbf{x}_N) \in \mathcal{F}_1$. If one now replaces the kernel $k(\cdot,\cdot)$ with another kernel $k'(\cdot,\cdot)$, this can be understood as working on another set of mapped vectorial data $\varphi'(\mathbf{x}_1), \varphi'(\mathbf{x}_2), \ldots, \varphi'(\mathbf{x}_N) \in \mathcal{F}_2$. The most well-known exchange between kernels is the replacement of inner product in the input space, which corresponds to the linear kernel, with a nonlinear kernel. But it is also possible to exchange two nonlinear kernels.

To avoid that the nomenclature becomes even more a burden to the reader, we first have to make something more clear. In the beginning of this chapter we introduced the concept of hypothesis space. This was a set of functions/hypotheses out of which the best function was chosen fitting the empirical risk minimization criteria. Notice that the feature space is also a space of functions. Therefore the feature space is a hypothesis space for which the same symbol $\mathcal{F}$ is used. In later sections we will show how models can be constructed in this feature space. A third synonym is often used in literature, which is the *Primal Space.* All three terms have the same meaning in this thesis.

2.4 Regularization networks: a regularization approach

As we explained earlier, the Tikhonov regularization leads to a formulation where there is a balance between Empirical Risk minimization and regularization for a fixed γ. Let $\mathcal{H}$ be an RKHS induced by the kernel k, the hypothesis f is defined by the variational problem as explained in [108]

$$\min_{f \in \mathcal{H}} \quad \sum_{i=1}^{N} V\left(y_i, f\left(\mathbf{x}_i\right)\right) + \frac{1}{2\gamma} \left\|f\right\|_k^2 , \tag{2.35}$$

where $V\left(y_i, f\left(\mathbf{x}_i\right)\right)$ is an arbitrary loss function evaluated on the data set $\left\{\left(\mathbf{x}_i, y_i\right)\right\}_{i=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y}$ of size N and γ is the regularization parameter. Notice that compared (2.10) to we omitted the factor $\frac{1}{N}$ since it does not affect the solution.

In Section 2.2 we saw that each element of a RKHS can be written as $f\left(\centerdot\right) = \sum_{i=1}^{d_{\mathcal{H}}} c_i \phi_i\left(\centerdot\right)$ with $\left\|f\right\|_k^2 = \sum_{i=1}^{d_{\mathcal{H}}} \frac{c_i^2}{\lambda_i}$. By filling in this expansion and computing the solution by setting the first derivative w.r.t. c_i equal to zero one gets $\forall i = 1, \ldots, d_{\mathcal{H}}$:

$$\sum_{j=1}^{N} V'\left(y_j, f\left(\mathbf{x}_j\right)\right) \phi_i\left(\mathbf{x}_j\right) + \frac{1}{\gamma}\frac{c_i}{\lambda_i} = 0, \tag{2.36}$$

where $V'\left(y_j, f\left(\mathbf{x}_j\right)\right)$ is the derivative w.r.t. c_i of the loss function $V\left(y_j, f\left(\mathbf{x}_j\right)\right)$. If one defines $\alpha_j = \gamma V'\left(y_j, f\left(\mathbf{x}_j\right)\right)$ one can express the coefficients $c_i = \lambda_i \sum_{j=1}^{N} \alpha_j \phi_i\left(\mathbf{x}_j\right)$. The solution of the variational problem (2.35) can be written as:

$$\begin{aligned} f\left(\mathbf{x}\right) &= \sum_{i=1}^{d_{\mathcal{H}}} c_i \phi_i\left(\mathbf{x}\right) \\ &= \sum_{j=1}^{N} \alpha_j \sum_{i=1}^{d_{\mathcal{H}}} \lambda_i \phi_i\left(\mathbf{x}_j\right) \phi_i\left(\mathbf{x}\right) \\ &= \sum_{j=1}^{N} \alpha_j k(\mathbf{x}_j, \mathbf{x}). \end{aligned} \tag{2.37}$$

This is essentially what the the *Representer Theorem,* according to [119] and [150], states. The solution to this regularization problem (2.35) over the domain $\mathcal{X}$ can be expressed as a linear combination kernel functions. The $\{\alpha_1, \ldots, \alpha_N\}$ are the parameters of the model f that we have to estimate. The remarkable fact about this theorem is that although the Tikhonov regularization problem is a minimization problem on a possibly infinite dimensional RKHS $\mathcal{H}$ of functions of the form (2.27) where the kernels are lying in arbitrary points of $\mathcal{X}$, the solution space consists of functions spanned by kernels $k(\mathbf{x}_p,\cdot)$ at the training points. But still this solution lives in this Reproducing Kernel Hilbert Space $\mathcal{H}$. This formulation of the solution in terms of the kernels $k(\mathbf{x}_p,\cdot)$ is called the *Dual Formulation* or *Support Vector expansion*. This last name originates from the feature space interpretation of this solution. Remember that each kernel function $k(\mathbf{x}_p,\cdot)$ corresponds to a vector representation $\varphi(\mathbf{x}_p)$ in the feature space. So, the solution is always a linear combination of this set of 'support vectors' $\varphi(\mathbf{x}_1),\ldots,\varphi(\mathbf{x}_N)$.

Once the choice of the kernel is done together with a fixed regularization parameter γ, the parameters $\{\alpha_{1,}\ldots,\alpha_N\}$ of the solution depend on the choice of the loss function $V(y, f(\mathbf{x}))$. We already mentioned in the previous sections that our main attention in this work will go to the squared loss function, defined as $V(y,f(\mathbf{x})) = \frac{1}{2}(y - f(\mathbf{x}))^2$. Solutions of this functional

$$\min_{f\in\mathcal{H}} \quad \frac{1}{2}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))^2 + \frac{1}{2\gamma}\|f\|_k^2 \tag{2.38}$$

are known as *Regularization Networks* ([42],[108]). This regularization problem can be solved easily by substituting the solution as given by (2.37). Remember that the norm in the RKHS $\mathcal{H}$ constructed by the functions of the form $f(\mathbf{x}) = \sum_{j=1}^{N}\alpha_j k(\mathbf{x}_j,\mathbf{x})$ is (see also 2.29):

$$\|f\|_k^2 = \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j). \tag{2.39}$$

This results in the following matrix formulation:

$$\min_{\alpha} \quad U(\alpha) = \min_{\alpha} \quad \frac{1}{2}(\mathbf{y} - K\alpha)^T(\mathbf{y} - K\alpha) + \frac{1}{2\gamma}\alpha^T K\alpha, \tag{2.40}$$

where $\alpha = [\alpha_1 \ldots \alpha_N]^T$ and K is the symmetric semi-positive definite matrix defined over the data set $\mathcal{D}$ and given by $K_{lm} = k(\mathbf{x}_l,\mathbf{x}_m)$, $l, m = 1\ldots N$.

At this stage it is important to mention that there are confusing names used in the kernel literature. This originates in the different definition of the concept of 'positive definiteness' for functions and matrices. In (2.15) we saw the definition for (strictly) positive definite kernels functions. The eigenvalues as used in the Mercer theorem (1) for (strictly) positive definite kernel functions are (strictly) positive. Kernel functions that have a limited number of eigenvalues equal to or smaller than zero are called conditionally positive definite. Using a data set $\mathcal{D}$, a matrix K can be defined based on a kernel function $k(\cdot, \cdot)$ such that $K_{lm} = k(\mathbf{x}_l, \mathbf{x}_m)$, $l, m = 1 \ldots N$. This matrix is called the *kernel matrix* or *Gram matrix.* For matrices, however, other names are used. A matrix is called positive definite is all its eigenvalues are strictly positive (see also Appendix A.2). Matrices that have some eigenvalues equal to zero are called *positive semi definite.* If some of the eigenvalues are negative the matrix is called *indefinite.* this explain why positive definite kernel functions construct positive semi definite kernel matrices. In general the following 'translation' schema can be applied:

Kernel function	Kernel Matrix
strictly positive definite	positive definite
positive definite	positive semi definite
conditional positive definite	indefinite

Let us come back to the model formulation. Finding the optimal solution of the Tikhonov regularization is equivalent to finding the optimal parameters α by minimizing the above optimization function $U(\alpha)$. This is done by setting the gradient of this optimization function $U(\alpha)$ equal to zero

$$\begin{aligned}\frac{\partial U(\alpha)}{\partial \alpha} &= \frac{1}{2}\frac{\partial}{\partial \alpha}\left(\mathbf{y}^T\mathbf{y} - 2\alpha^T K\mathbf{y} + \alpha^T K^2\alpha + \frac{1}{\gamma}\alpha^T K\alpha\right) \\ &= -K\mathbf{y} + K^2\alpha + \frac{1}{\gamma}K\alpha \\ &= 0. \end{aligned} \tag{2.41}$$

So the solution of this problem in the general case that the matrix K is a positive semi-definite, is given by $\left(K^2 + \frac{1}{\gamma}K\right)\alpha = K\mathbf{y}$. Since the matrix in this linear system is also positive semi-definite, it does not have a unique solution but a subspace of solutions. In the case that K is positive definite, it is also invertible. So, the optimal parameters α satisfies the following linear equation:

$$(K + \frac{1}{\gamma}I)\alpha = \mathbf{y}. \tag{2.42}$$

Because we only need one solution, in the case that K is not invertible we will also use the solution of (2.42).

This shows that choosing a model based on Tikhonov regularization with a squared loss function leads to a convex optimization problem were the solution can be found by solving a set of linear equations. This linear system (2.42) is positive definite and therefore invertible. Notice hereby that regularization leads to the term $\frac{1}{\gamma}I$, controlled by the regularization parameter γ. The uniqueness of the solution of this formulation is a very important property. The uniqueness follows from the convex formulation of this problem. Other convex optimization problems can be formulated by other loss functions but only the squared loss leads to a linear system.

These results also follow from a more general regularization setup as studied by Poggio [108]. In that work it was shown that the optimal function f that approximates the true function f^* using the data $\mathcal{D}$ is the result of the minimization of the functional:

$$\min_f \quad \frac{1}{2}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))^2 + \frac{1}{2\gamma}\|Pf\|^2, \tag{2.43}$$

where P is a constraint operator (usually a differential operator), $\|\cdot\|$ is a norm in the space to which Pf belongs (usually L^2). The solution of this variational problem can be found via the Euler-Lagrange equations and can be written as the solution of the partial differential equations:

$$\hat{P}Pf(\mathbf{x}) = \frac{1}{\gamma}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))\,\delta(\mathbf{x}-\mathbf{x}_i), \tag{2.44}$$

where $\hat{P}$ is the adjoint of the differential operator P and $\delta(\cdot)$ is the Dirac delta function. The solution of this partial differential equation is given by

$$f(\mathbf{x}) = \frac{1}{\gamma}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))\,G(\mathbf{x},\mathbf{x}_i), \tag{2.45}$$

where $G(\cdot,\cdot)$ is the Green's function of the differential operator $\hat{P}P$ satisfying the differential equation $\hat{P}PG(\mathbf{x},\mathbf{x}') = \delta(\mathbf{x}-\mathbf{x}')$. It is also shown in this paper [108] that for specific choices of P the corresponding Green's functions are the RBF kernels we use in this work. The similarity with the derivation of (2.37) is then easy to see.

2.5 The b-term

In the previous section we have seen that the solution of the Tikhonov regularization (2.10) always has the form

$$f(\mathbf{x}) = \sum_{p=1}^{N}\alpha_p k(\mathbf{x}_p,\mathbf{x}). \tag{2.46}$$

But in many other models, like support vector machines ([146],[147]) or Splines [150], solutions of the form

$$f(\mathbf{x}) = \sum_{p=1}^{N} \alpha_p k(\mathbf{x}_p, \mathbf{x}) + b \tag{2.47}$$

are suggested where the constant $b \in \mathbb{R}$ is an *intercept* or *bias.* In [110] the question was raised if this b is really necessary and what the consequences are when it is introduced?

Following [110], the solution to this question depends on the kernel we consider. Until now we focussed on positive definite kernels that fulfill the Mercer condition. This Mercer condition tells us that to each kernel an eigenvalue spectrum $\{\lambda_i\}_{i=1}^{d_{\mathcal{H}}}$ and eigenfunctions $\{\phi_i\}_{i=1}^{d_{\mathcal{H}}}$ can be associated according to the integral operator (2.2). Kernels for which all of these eigenvalues are strictly positive are strictly positive definite kernels. Those positive definite kernels that have a finite number of eigenvalues that are negative or zero are called *conditionally positive definite.* The number of non-positive eigenvalues is referred to as the order of the kernel.

In the case of a conditionally positive definite kernel of order 1 the extra term b is necessary ([150],[44],[110]).It is shown in [110] that for the Mercer kernels it is not necessary to use an extra b-term. However, since every positive definite kernel is also conditionally positive definite, using a b-term is not wrong.

They also showed that using an extra b-term corresponds to solving the Tikhonov regularization problem in another RKHS induced by the conditionally positive definite kernel of order 1: $k'(\mathbf{x}, \mathbf{y}) = k(\mathbf{x}, \mathbf{y}) - \lambda_1 \phi_1(\mathbf{x}) \phi_1(\mathbf{y})$ (for an exact definition we refer to [110]). Hereby this second term $c = \lambda_1 \phi_1(\mathbf{x}) \phi_1(\mathbf{y})$ is a positive constant. The optimal solution if one uses such a kernel is of the form ([150],[44]) : $f(\mathbf{x}) = \sum_{p=1}^{N} \alpha'_p k'(\mathbf{x}_p, \mathbf{x}) + b$. The optimal parameters α' follow from the Tikhonov regularization problem:

$$\min_{f \in \mathcal{H}} \quad \frac{1}{2} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 + \frac{1}{2\gamma} \|f\|_{k'}^2 . \tag{2.48}$$

By substituting the solution into this optimization problem and omitting all the constants, we end up with the following cost function for α' and b (notice that there is no regularization on the b):

$$\min_{\alpha', b} U(\alpha', b) = \min_{\alpha', b} \frac{1}{2} \left(\mathbf{y} - K'\alpha' - b\mathbf{1}\right)^T \left(\mathbf{y} - K'\alpha' - b\mathbf{1}\right) + \frac{1}{2\gamma} \alpha'^T K' \alpha'. \tag{2.49}$$

The optimality conditions of this minimization problem are found by setting the gradients equal to zero:

$$\begin{cases} \frac{\partial U(\alpha',b)}{\partial \alpha'} = K'\left(K'\alpha' + \frac{1}{\gamma}\alpha' - \mathbf{y} + b\mathbf{1}\right) = 0, \\ \frac{\partial U(\alpha',b)}{\partial b} = \mathbf{1}^T\left(K'\alpha' - \mathbf{y} + b\right) = 0 \end{cases} \tag{2.50}$$

or

$$\begin{cases} (K' + \frac{1}{\gamma}I)\alpha' + b\mathbf{1} = \mathbf{y}, \\ \mathbf{1}^T\alpha' = 0. \end{cases} \tag{2.51}$$

Solving this set of linear equations with respect to α' and b gives the optimal solution. But since k' corresponds to the sum of a positive definite kernel k minus a constant c we find, by using the optimality conditions, that:

$$\begin{aligned} f(\mathbf{x}) &= \sum_{p=1}^{N} \alpha'_p k'(\mathbf{x}_p, \mathbf{x}) + b \\ &= \sum_{p=1}^{N} \alpha'_p k(\mathbf{x}_p, \mathbf{x}) - c\sum_{p=1}^{N} \alpha'_p + b \\ &= \sum_{p=1}^{N} \alpha'_p k(\mathbf{x}_p, \mathbf{x}) + b. \end{aligned} \tag{2.52}$$

So, the remarkable result of this relation is that the solution of the models based on either k or k' are similar. This means that the optimal parameters α' and b of the model of the form $f(\mathbf{x}) = \sum_{p=1}^{N} \alpha'_p k(\mathbf{x}_p, \mathbf{x}) + b$ with a positive definite matrix K can be found from the set of linear equations [44]:

$$\begin{cases} (K + \frac{1}{\gamma}I)\alpha' + b\mathbf{1} = \mathbf{y}, \\ \mathbf{1}^T\alpha' = 0. \end{cases} \tag{2.53}$$

Hence using the b-term can be interpreted as solving the Tikhonov regularization problem in another RKHS induced by k'.

If we now compare the relations between the models with (2.53) and without the b-term (2.42) for a positive definite k, the following closed form relations hold between the unknowns of the models:

$$\alpha - \alpha' = (K + \frac{1}{\gamma}I)^{-1} b\mathbf{1}. \tag{2.54}$$

This proves that there is a one to one relation between both models. And using a model b-term is similar to choosing a different kernel in a different feature space relative to the initial $k(\cdot,\cdot)$ and in the standard notation without b.

In practical applications where the data are not always centered, the use of a b-term is often beneficial. Shifts by a constant therefore should not be penalized. Especially in classification problems, bias corrections are often applied based on the Reciever-Operator-Characteristics (ROC) ([135],[136]). In classification the binary output is achieved by the sign-operator: $f(\mathbf{x}) = \text{sign}(\sum_{p=1}^{N} \alpha'_p k(\mathbf{x}_p, \mathbf{x}) + b)$. Using a b-term acts as a threshold adapted according to the data, instead of using the arbitrary zero-threshold. It are these practical reasons that justify to use models including a b-term, which is computed based on the data, in the remaining of this work.

2.6 Least squares support vector machines: an optimization approach

In the previous section we have shown how to derive Regularization Network models of the form $f(\mathbf{x}) = \sum_{p=1}^{N} \alpha_p k(\mathbf{x}_p, \mathbf{x}) + b$ for regression tasks starting from a Tikhonov regularization scheme (2.10). These models are elements of an RKHS induced by a kernel $k(.,.)$, which can be chosen freely. We also showed how this kernel $k(.,.)$ can be related to a mapping $\varphi(.)$ into feature space $\mathcal{F}$. Using a quadratic loss function leads to a set of linear equations for finding the optimal parameters α and b of our model.

We explained that for classification tasks a similar approach can be used, based on the Tikhonov regularization. The model then takes the form $f(\mathbf{x}) = sign(\sum_{p=1}^{N} \alpha_p k(\mathbf{x}_p, \mathbf{x}) + b)$, where the binary output is enforced by the sign-operator.

In this section we will rederive the formulations, starting form a different viewpoint. We will build the models starting from a formulation in the feature space. This approach will give an extra primal-dual interpretation. To show a clear distinction between models for classification and regression, we will explain them in different sections.

2.6.1 LS-SVM classifiers

We have explained in the previous sections, that the use of a kernel can be associated with a mapping of the data $\mathcal{D}$ in a high dimensional feature space $\mathcal{F}$ of dimension $d_{\mathcal{H}}$, possibly infinite. This feature space is a vector space and therefore the mapped data $\varphi(\mathbf{x}_1), \varphi(\mathbf{x}_2), \ldots, \varphi(\mathbf{x}_N) \in \mathcal{F}$ will be handled as is they are finite dimensional data. In the case that the dimension $d_{\mathcal{H}}$ is infinite we assume that all the convergence criteria hold.

The problem of classification is different from regression $y \in \mathcal{Y}$ in the sense that the target values are $\mathcal{Y} = \{-1, 1\}$. The idea is that given a set of data points, belonging to two different classes, a classifier is trained to divide new points in those two classes, based on similar ideas as for regression. Suykens and Vandewalle showed that this classification problem can be treated as a least squares problem in feature space. This binary classification setup in feature space is called *Least Squares Support Vector Machines* ([132],[131]). Recently also multiclass kernel classifiers have been proposed for the LS-SVM ([133],[144]). Because they are all based on the same idea, we will only discuss the binary version of this method.

The Least Squares Support Vector Machine classifier (LS-SVM) [132] is a modification of the standard Vapnik SVM classifier. Let there be a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with inputs $\mathbf{x}_i \in \mathbb{R}^d$ and given class label $y_i \in \{-1, +1\}$. The idea of Vapnik for training an SVM classifier consists of finding the linear separating hypersurface $\mathbf{w}^T\varphi(x) + b = 0$ in the feature space $\mathcal{F}$ that separates the mapped data $\{(\varphi(\mathbf{x}_1), y_1), (\varphi(\mathbf{x}_N), y_2), \ldots, (\varphi(\mathbf{x}_N), y_N)\}$. This comes down to finding the appropriate vector $\mathbf{w}$ and scalar b so that the following relations hold $\forall i \in \{1, \ldots, N\}$:

$$\begin{cases} \mathbf{w}^T\varphi(\mathbf{x}_i) + b \geq 0 \text{ if } y_i = +1, \\ \mathbf{w}^T\varphi(\mathbf{x}_i) + b \leq 0 \text{ if } y_i = -1. \end{cases} \tag{2.55}$$

Remember that the function $\varphi : \mathcal{X} \subseteq \mathbb{R}^d \to \mathcal{F}$ is a non-linear mapping of the *input space* $\mathcal{X} \subseteq \mathbb{R}^d$ into a higher dimensional *feature space* $\mathcal{F}$. The $\varphi(\mathbf{x}_1), \varphi(\mathbf{x}_2), \ldots, \varphi(\mathbf{x}_N)$ are functions of an RKHS or $d_{\mathcal{H}}$-dimensional vectors depending respectively on the reproducing kernel mapping or Mercer kernel mapping interpretation of φ. (See Section 2.3). Since the model is defined in this feature space it is sometimes also called the *Primal Space.*

According to statistical learning theory [146], [147]) there is an additional element that needs to be taken into account. For a good generalization one demands that both classes are separated with a certain margin. This is achieved by demanding that $\forall i \in \{1, \ldots, N\}$:

$$\begin{cases} \mathbf{w}^T\varphi(\mathbf{x}_i) + b \geq +1 \text{ if } y_i = +1, \\ \mathbf{w}^T\varphi(\mathbf{x}_i) + b \leq -1 \text{ if } y_i = -1. \end{cases} \tag{2.56}$$

The distance between those two hyperplanes $\mathbf{w}^T\varphi(x_i) + b = +1$ and $\mathbf{w}^T\varphi(x_i) + b = -1$ in the feature space is called the *separating margin* and is given by $\frac{2}{\|\mathbf{w}\|_2}$. So, besides finding the separating hyperplane, also this margin has to be maximized. This leads to the constrained optimization problem

$$\left\{\begin{array}{l}\min\limits_{\mathbf{w},b} \quad \frac{1}{2}\left\|\mathbf{w}\right\|_2^2 \\ \text{s.t. } y_i\left(\mathbf{w}^T\varphi\left(\mathbf{x}_i\right)+b\right)\geq 1,\ \forall i\in\{1,\ldots,N\}\,. \end{array}\right. \tag{2.57}$$

In this formulation we assumed that the data are strictly separable, which means that one can always find a hyperplane in feature space that separates all the data. However, this requirement is often too strong and easily leads to overfitting. Therefore it is sometimes better to allow some points to be misclassified. This does not necessarily mean that this induces a bad classifier. In practice it is often the case that the data set is disturbed by noise. Some of the data points therefore are bad representatives of their class. It is therefore not a mistake to allow them to be misclassified. Another reason is that we always try to find a balance between the empirical and structural risk. Allowing misclassification increases the empirical risk, but if the decrease in structural risk is higher, the end-result is still favorable.

In the Vapnik formalism, this misclassification is monitored by attaching a penalty $\xi_i \geq 0$ for each point that is misclassified. For each point that is classified correctly one has $\xi_i = 0$. The sum of these ξ_i's is added to the cost function in a way that misclassification is penalized:

$$\left\{\begin{array}{l}\min\limits_{\mathbf{w},\xi,b} \quad \frac{1}{2}\left\|\mathbf{w}\right\|_2^2 + C\sum\limits_{i=1}^{N}\xi_i \\ \text{s.t. } y_i\left(\mathbf{w}^T\varphi\left(\mathbf{x}_i\right)+b\right)\geq 1-\xi_i,\ \forall i\in\{1,\ldots,N\}\,, \\ \qquad \xi_i\geq 0,\ \forall i\in\{1,\ldots,N\}\,. \end{array}\right. \tag{2.58}$$

The weight of the penalization of misclassification is controlled by an extra hyperparameter C. The geometric interpretation of the Vapnik classification in feature space is given in Figure 2.3. Later in this chapter will we show that this optimization problem can be reformulated as a constrained quadratic programming problem ([17],[28]).

To avoid these sometimes hard to solve optimization problems, the training problem can be reformulated in another form. Instead of using the inequality constraints to find the decision hyperplane, the LS-SVM uses equality constraints. The difference is compensated by adding an extra term to the cost function that penalizes the deviations from the hyperplanes $\mathbf{w}^T\varphi\left(x_i\right)+b=+1$ and $\mathbf{w}^T\varphi\left(x_i\right)+b=-1$ and this for each point. These deviations are measured by a scalar error e_i $\forall i\in\{1,\ldots,N\}$. The training problem now has the form

$$\left\{\begin{array}{l}\min\limits_{\mathbf{w},\mathbf{e},b} \quad \frac{1}{2}\left\|\mathbf{w}\right\|_2^2 + \frac{\gamma}{2}\left\|\mathbf{e}\right\|_2^2 \\ \text{s.t. } y_i\left(\mathbf{w}^T\varphi\left(\mathbf{x}_i\right)+b\right)= 1-e_i,\ \forall i\in\{1,\ldots,N\}\,, \end{array}\right. \tag{2.59}$$

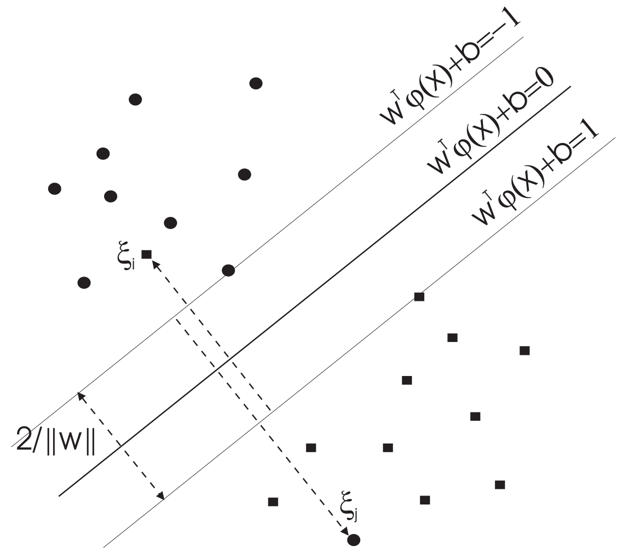

Figure 2.3: In this figure we see the classification behaviour in Feature Space of the Vapnik SVM. The squares and the circles represent the elements of the different classes. We show the geometric interpretation of the *margin* for the separating hyperplane and the misclassification measure ξ_i attached to each point.

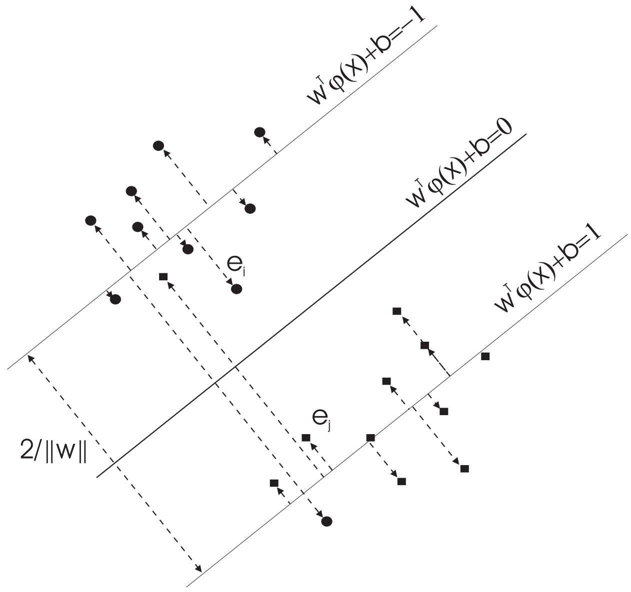

Figure 2.4: In this figure we see the classification behaviour in Feature Space of the LS-SVM. The squares and the circles represent the elements of the different classes. We show the geometric interpretation of the *margin*, called soft margin, in this context and the error e_i attached to each point.

where γ plays the role of a regularization parameter. For each point $\varphi(\mathbf{x}_i)$ an extra error e_i is introduced stacked in the vector $\mathbf{e} = [e_1; e_2; \ldots; e_N]$. This is visualized in Figure 2.4. Notice however that by changing the inequality constraints into equality constraints, the original concept of *margin*, as used in the work of Vapnik, is lost. In this setup the norm constraint on the $\mathbf{w}$ should be seen as a regularization of the parameters of the model as is common in the Neural Network literature [7]. Hereby the goal is to keep the parameters of the model small in order to avoid overfitting and thereby increase the generalization error. In [129] also the link with Fisher discriminant analysis was shown. This formulation of the LS-SVM corresponds to discriminant analysis in feature space where maximizing the margin, as given in this model, is equal to minimizing the within class scatter.

Notice that both optimization problems are formulated in feature space. It is therefore called the *Primal Formulation* of the optimization process. After solving this optimization problem the *Primal Formulation* of the classification model is:

$$y = \operatorname{sign}(\mathbf{w}^T \varphi(\mathbf{x}_i) + b). \tag{2.60}$$

Since the dimension $d_{\mathcal{H}}$ of the feature space is high, possibly infinite, this problem is difficult, if not impossible, to solve. Furthermore, the mapping $\varphi(\mathbf{.})$, corresponding to a kernel, is not always known. Hence the following trick is applied. For solving an optimization problem the Lagrangian is constructed. The optimality conditions of this constrained optimization problem are given by the saddle point of this Lagrangian also known as the Karush-Kuhn-Tucker conditions [97]. The Lagrangian is given by

$$\begin{aligned}\mathcal{L}(\mathbf{w}, b, \mathbf{e}, \alpha) &= \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{\gamma}{2}\|\mathbf{e}\|_2^2 \\ &\quad - \sum_{i=1}^{N} \alpha_i \left(y_i \left(\mathbf{w}^T \varphi(\mathbf{x}_i) + b\right) - 1 + e_i\right),\end{aligned} \tag{2.61}$$

where α is the vector of the Lagrange multipliers. One gets the following set of linear equations from the Karush-Kuhn-Tucker conditions:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 \rightarrow \mathbf{w} = \sum\limits_{i=1}^{N} \alpha_i y_i \varphi(\mathbf{x}_i), \\ \frac{\partial \mathcal{L}}{\partial b} = 0 \rightarrow \sum\limits_{i=1}^{N} \alpha_i y_i = 0, \\ \frac{\partial \mathcal{L}}{\partial e_i} = 0 \rightarrow \alpha_i = \gamma e_i, \ \forall i \in \{1, \ldots, N\}, \\ \frac{\partial \mathcal{L}}{\partial \alpha_i} = 0 \rightarrow y_i \left(\mathbf{w}^T \varphi(\mathbf{x}_i) + b\right) = 1 - e_i, \ \forall i \in \{1, \ldots, N\}. \end{cases} \tag{2.62}$$

If we substitute these into the primal optimization problem and eliminate $\mathbf{w}$ and b we get the *Dual Optimization Problem* or *Wolfe Dual* ([96],[97],[10]):

$$\begin{cases} \max\limits_{\alpha} \quad -\frac{1}{2} \sum\limits_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \left(k(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{\gamma}\delta_{i,j}\right) + \sum\limits_{i=1}^{N} \alpha_i \\ \text{s.t.} \ \sum\limits_{i=1}^{N} \alpha_i y_i = 0. \end{cases} \tag{2.63}$$

This formulation has two important advantages. First we see that the dimensionality of the optimization problem is equal to the number of data points N. This means that the training process is not dependent on the dimensionality of the Feature space, nor the dimension of the input space d. Second, one uses the property that the kernel is an inner product in feature space: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle_{\mathcal{H}}$. This opens a lot of opportunities. By applying the *kernel trick* (see Section 2.3) one can show that for different kernels the general optimization problem remains the same. Therefore kernel models are considered to be very flexible. A third important property of those optimization problems is that they are convex. The Hessian matrix is a full rank positive definite matrix, which guarantees a unique solution.

The solution of this *Dual Optimization problem* is given by the solution of the following linear system which can be found from (2.62) by eliminating $\mathbf{w}$ and b :

$$\begin{bmatrix} 0 & \mathbf{y}^T \\ \mathbf{y} & H \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{1}_N \end{bmatrix}, \tag{2.64}$$

where $\mathbf{y} = [y_1; y_2; \ldots; y_N]$ denotes the column vector formed by the labels of the training points. The positive definite matrix H is defined by its elements $h_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + \delta_{ij}\frac{1}{\gamma}$ where δ_{ij} denotes the Kronecker delta. Note that this linear system, for the classification case, has a similar form as the linear system we derived for the regression problems (2.53).

After the optimal parameters are found, the *Dual Formulation* of the resulting classifier takes the form

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b\right). \tag{2.65}$$

2.6.2 LS-SVM regression

In a similar way as we did for the classification problem, we now give a derivation of an LS-SVM model for regression starting from the model in feature space. As we have explained previously, the goal of regression is to find a function $f : \mathcal{X} \to \mathcal{Y}$, given the data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ that can be used to predict a value y given an $\mathbf{x} \in \mathcal{X}$. Hereby $y \in \mathbb{R}$ is a real value instead of binary one as in the case of classification. The model in Primal or feature space is defined as

$$y = \mathbf{w}^T \varphi(\mathbf{x}) + b \tag{2.66}$$

where the function $\varphi : X \subseteq \mathbb{R}^d \to \mathcal{F}$ is a non-linear mapping of the *input space* $X \subseteq \mathbb{R}^d$ into a higher dimensional *feature space* $\mathcal{F}$. Where this feature space is an RKHS of functions $\varphi(\mathbf{x}_1), \varphi(\mathbf{x}_2), \ldots, \varphi(\mathbf{x}_N)$. Given the data, the cost function of the training process in the Primal Space is

$$\begin{cases} \min\limits_{\mathbf{w},\mathbf{e},b} & \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{\gamma}{2}\|\mathbf{e}\|_2^2 \\ \text{s.t. } y_i = \mathbf{w}^T \varphi(\mathbf{x}_i) + b + e_i, & \forall i \in \{1, \ldots, N\}, \end{cases} \tag{2.67}$$

where γ plays the role of a regularization parameter. For each fitted data point $(\mathbf{x}_i, y_i)$ an extra error e_i is introduced combined in the vector $\mathbf{e} = [e_1; e_2; \ldots; e_N]$. These e_i's are the error terms for the model in the feature space. The optimal model will be chosen by minimizing the cost function (2.67) where the errors $\mathbf{e}$ are minimized in *Least Squares* sense. This formulation corresponds to *ridge regression* [57] in feature space. Since the dimension of the feature space is high, possibly infinite, this problem is difficult if not impossible to solve. By constructing the Dual Formulation of this optimization problem this will be circumvented ([116],[129]). The Lagrangian of this optimization problem is:

$$\begin{aligned}\mathcal{L}(\mathbf{w}, b, \mathbf{e}, \alpha) = & \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{\gamma}{2}\|\mathbf{e}\|_2^2 \\ & -\sum_{i=1}^{N}\alpha_i\left(\mathbf{w}^T\varphi(\mathbf{x}_i) + b + e_i - y_i\right),\end{aligned} \tag{2.68}$$

where $\alpha = [\alpha_1\alpha_1\ldots\alpha_N]^T$ are the Lagrange multipliers. The Karush-Kuhn-Tucker optimality conditions are:

$$\begin{cases} \frac{\partial\mathcal{L}}{\partial w} = 0 \rightarrow \mathbf{w} = \sum\limits_{i=1}^{N}\alpha_i\varphi(\mathbf{x}_i), \\ \frac{\partial\mathcal{L}}{\partial b} = 0 \rightarrow \sum\limits_{i=1}^{N}\alpha_i = 0, \\ \frac{\partial\mathcal{L}}{\partial e_i} = 0 \rightarrow \alpha_i = \gamma e_i,\ \forall i \in \{1, \ldots, N\}, \\ \frac{\partial\mathcal{L}}{\partial\alpha_i} = 0 \rightarrow \mathbf{w}^T\varphi(\mathbf{x}_i) + b + e_i - y_i = 0,\ \forall i \in \{1, \ldots, N\}. \end{cases} \tag{2.69}$$

Notice from these KKT conditions it follows that there is a connection between the Lagrange multipliers α_i and the error e_i of the models in Feature space. By elimination of $\mathbf{w}$ and b and substituting these into the primal optimization problem we get the *(Wolfe) Dual Optimization Problem* ([96],[97],[10]):

$$\begin{cases} \max\limits_{\alpha} \quad -\frac{1}{2}\sum\limits_{i,j=1}^{N}\alpha_i\alpha_j\left(k(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{\gamma}\delta_{i,j}\right) + \sum\limits_{i=1}^{N}\alpha_i y_i \\ \text{s.t.} \ \sum\limits_{i=1}^{N}\alpha_i = 0. \end{cases} \tag{2.70}$$

This constrained quadratic optimization problem has a unique minimum since the Hessian matrix H with elements $h_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{\gamma}\delta_{i,j}\,\forall i, j \in \{1, \ldots, N\}$ is positive definite. Hereby we used the property that the kernel is inner product in feature space: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle_{\mathcal{H}}$. The unique solution can be found by solving the linear system:

$$\begin{bmatrix} 0 & \mathbf{1}_N^T \\ \mathbf{1}_N & H \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{y} \end{bmatrix}. \tag{2.71}$$

This linear system is identical to the linear system we derived based on the regularization theory approach (2.53).

After the optimal parameters are found, the *Dual Formulation* of the resulting function estimator takes the form

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b. \tag{2.72}$$

So, we have proven that this choice of parameter estimation for models of the form (2.72) can be justified from two different viewpoints. The first is starting form a ridge regression scheme in feature space with a least squares error minimization procedure. This gives us an extra primal-dual interpretation of the problem setup. Second, starting form a regularization network setup with a squared loss function approach in a predefined RKHS gives us a similar model. Both end up with a support vector expansion where the parameters α and b are found from a set of linear equations.

2.6.3 Time-series prediction with LS-SVM regression

Until know we have focussed on learning problems where the data set $\mathcal{D}$ consists of input and output points. The goal was, based on the experience learned from this data set, to make a new prediction $\hat{y}$ for new points of the input domain $\mathcal{X} \subseteq \mathbb{R}^d$.

Another problem formulation is time-series prediction. Hereby a real valued sequences of ordered data is given $y_1, y_2, \ldots, y_N$ and the goal is to predict the next elements of this sequence that $y_{N+1}, y_{N+2,\ldots}$ This kind of problems can be solved also with the models that we have discussed previously ([94],[91],[84]) and applied to many different domains as financial time-series ([142],[143]) and system identification ([39],[38]). The solution scheme that will be used is the *Nonlinear Auto-Regressive with eXogenous inputs scheme* or *NARX [79].* This is defined as:

$$\hat{y}_{i+1} = f(y_i, y_{i-1}, \ldots, y_{i-q}), \tag{2.73}$$

where f is the function to be estimated and the number of time-steps back q is called the *lag.* This function f will be estimated using the regression model we discussed in the previous chapter. Hereby will we use the previous short sequence of q points $y_{k-q}, \ldots, y_{k-1}, y_k$ as a q-dimensional; input to predict the next y_{k+1}. Using this method one can construct the necessary data set $\mathcal{D}$ to train the regression model as demonstrated below:

$$\begin{cases} \mathbf{x}_k & = & [y_{k-q-1}, \ldots, y_{k-2}, y_{k-1}]^T & y_k, \\ \mathbf{x}_{k+1} & = & [y_{k-q}, \ldots, y_{k-1}, y_k]^T & y_{k+1}, \\ \mathbf{x}_{k+2} & = & [y_{k-q+1}, \ldots, y_k, y_{k+1}]^T & y_{k+2}, \\ \ldots & & & \ldots \\ \mathbf{x}_N & = & [y_{N-q}, \ldots, y_{N-2}, y_{N-1}]^T & y_N. \end{cases} \tag{2.74}$$

Once the model is trained the continuation of the sequence can be predicted by the recursive definition (2.73). In the next chapters we will demonstrate how such NARX models can also predict non-linear behavior as the Santa Fe chaotic laser data. For more information about time-series and system identification we advise ([138],[45]) and [79].

2.7 Kernel models and large data sets

The goal of the model building in Hilbert Spaces with regularization is to construct a model f for which the estimated risk $I[f]$ is as low as possible. Since knowing the estimated risk $I[f]$ is impossible, the models are constructed based on the empirical risk $I_{emp}[f, \mathcal{D}]$. This empirical risk depends on the chosen loss functional. The question now is, how to control the difference between the expected and empirical risk? Bousquet and Elisseeff [9, p.517] proved an upper bound on the estimated risk $I[f]$ for the models obeying the Tikhonov functional (2.35), defined by the kernel k and using a squared loss functional. This states that assuming $\forall \mathbf{x} \in \mathcal{X}, k(\mathbf{x}, \mathbf{x}) \leq \kappa^2$, the difference between the expected and empirical risk over a randomly drawn sample $\mathcal{D}$ is bounded, with probability at least $1 - \eta$, by:

$$I[f] \leq I_{emp}[f, \mathcal{D}] + \frac{4\kappa^2 B^2 \gamma}{N} + \left(8\kappa^2 B^2 \gamma + 2B\right) \sqrt{\frac{\ln 1/\eta}{2N}}, \tag{2.75}$$

where $y \in [0, B]$ and γ is the regularization parameter. This was proven based on the learning stability criteria of these models. The bound shows the strong dependency on N for having a good generalization performance. One can see that if the data set $\mathcal{D}$ contains more points N the bound on the generalization performance becomes tighter. This highlights the dilemma that we face. For a good generalization performances, one should train the models with as many data points as possible. However, for kernel models the

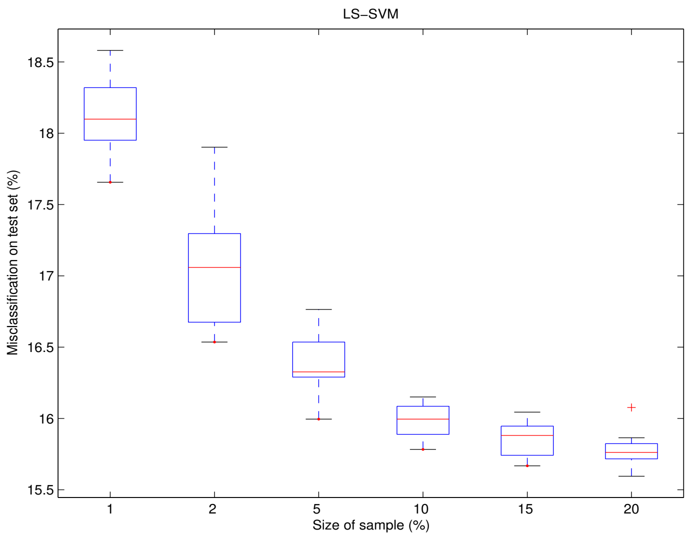

Figure 2.5: Misclassification percentage for the UCI adult data set measured on a test set. The performances shown are for increasing size of the training set samples and this for different randomizations of test and training set. One can see that both the mean and the variance of the misclassification rate on a test set decreases as one uses a larger training set.

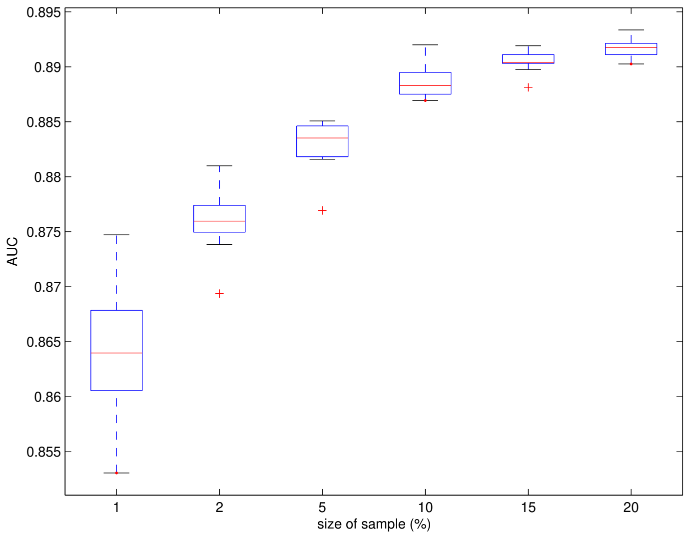

Figure 2.6: Area-under-the-curve of the ROC for the UCI adult data set measured on a test set. The performances shown are for increasing size of the training set samples and this for different randomizations. We see an increase of the mean area-under-the-curve. For larger training sets the area-under-the-curve approaches more the desired optimal value 1. Also for the area-under-the-curve one can see a decrease of the variance for the different randomizations.

computational complexity scales in worst case quadratic with the number of data points N. In the following chapters we will demonstrate how to improve this computational Achilles' heel.

To show the effect of the size of the data set on the generalization performance, we tried to visualize with the following test. For different size of subsamples of the UCI adult-data set (see Appendix C) we plotted the test error performance and the area-under-the-curve of a ROC curve, which is a standard measure for testing the classification performance ([136],[135],[35]). Recently a nice overview of the explanation of ROC curves was given in [80]. We did this for different randomly chosen subsets of the original data set. For each subsample size, measured in the ratio to the original size, we plotted the performance of the LS-SVM model for the different randomizations. For each iteration we did a hyperparameter tuning based on the cross-validation function [7] as implemented in LS-SVMlab[6]. The results are shown in Figure 2.5 and Figure 2.6. One can see the expected result that the size of the data set has a positive influence on the performance of the model. This reflects in a decrease of the mean misclassification error and a decrease of its variance. We also see an increase of the mean area-under-the-curve. For larger training sets the area-under-the-curve approaches more the desired optimal value 1. Also for the area-under-the-curve one can see a decrease of the variance for the different randomizations.

2.8 MSE and the bias-variance trade-off

The squared loss functional, which was used in the previous section, is similar to the *mean squared error criterion* often used in the statistics and neural networks literature. In these theories the mean squared error criterion is used to find the optimal model $f(\mathbf{x})$ as follows [7].

Assuming that relation between the input $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ is given by $y = f^*(\mathbf{x}) + e$ where $e \sim \mathcal{N}(0, \sigma)$ normal distributed noise and f^* is the function to be estimated. Based on a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ we will estimate the optimal function $f(\mathbf{x})$ that approximates $f^*(\mathbf{x})$ by using the mean-squared error criterion on the data set $\mathcal{D}$:

$$\mathrm{MSE}(\mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} (f(\mathbf{x}) - y_i)^2 . \tag{2.76}$$

[6]The software and documentation can be found at: 'http://www.esat.kuleuven.ac.be /sista/lssvmlab/'.

Under the assumption of the normal distributed noise, this corresponds to a maximum likelihood estimation of the parameters [7].

The error, in least squares sense, made by this scheme of model estimation can be studied by taking the limit for the size of the data set N going to infinity. Then, one can show that the *average generalization error* is

$$\begin{aligned} \mathrm{MSE} = & \int_{\mathcal{X}} \left(f(\mathbf{x}) - E\left[y | \, x \right] \right)^2 p\left(\mathbf{x}\right) d\mathbf{x} \\ & + \int_{\mathcal{X}} \left(E\left[y^2 \middle| \, \mathbf{x} \right] - E\left[y | \, \mathbf{x} \right] \right)^2 p\left(\mathbf{x}\right) d\mathbf{x}, \end{aligned} \tag{2.77}$$

where $p\left(\mathbf{x}\right)$ is the density of the input data and $E\left[y|\,\mathbf{x}\right]$ and $E\left[y^2\middle|\,\mathbf{x}\right]$ respectively the conditional average of the target data y and y^2 defined by

$$E\left[y | \, \mathbf{x} \right] = \int_{\mathcal{Y}} y p\left(y | \, \mathbf{x} \right) dy, \tag{2.78}$$

$$E\left[y^2 \middle| \, \mathbf{x} \right] = \int_{\mathcal{Y}} y^2 p\left(y | \, \mathbf{x} \right) dy. \tag{2.79}$$

This states that under optimal circumstances, which means N goes to infinity, the optimization is exact and the learning algorithm f has enough free parameters, the learning algorithm estimate $f(\mathbf{x})$ will be equal to $E\left[y|\,\mathbf{x}\right]$. Notice also that the second term is independent of the learning algorithm. It represents the intrinsic noise in the data and a lower limit on the error which can be achieved.

In practical situations we must deal with the problems arising for the fact that our data sets are limited. If a measure is needed to validate how good the estimation f is, the integrand of the first term $\left(f(\mathbf{x}) - E\left[y|\,x\right]\right)^2$ is used. Since this depends on the data set $\mathcal{D}$, which is used for training f, an average over all data sets is taken, which then gives:

$$\begin{aligned} E_{\mathcal{D}}\left[\left(f(\mathbf{x}) - E\left[y | \, \mathbf{x} \right] \right)^2 \right] = & \left(E_{\mathcal{D}}\left[f(\mathbf{x}) - E\left[y | \, x \right] \right] \right)^2 \\ & + E_{\mathcal{D}}\left[\left(f(\mathbf{x}) - E_{\mathcal{D}}\left[f(\mathbf{x}) \right] \right)^2 \right]. \end{aligned} \tag{2.80}$$

This is formulation is also known as the *bias-variance trade-off*. If we hereby refer back to (2.77) then we see that with appropriate weighting this gives:

$$(\text{bias})^2 = \int_{\mathcal{X}} \left(E_{\mathcal{D}}\left[f(\mathbf{x}) - E\left[y|\, x\right]\right]\right)^2 p(x)dx, \tag{2.81}$$

$$\text{variance} = \int_{\mathcal{X}} E_{\mathcal{D}}\left[\left(f(\mathbf{x}) - E_{\mathcal{D}}\left[f(\mathbf{x})\right]\right)^2\right] p(x)dx. \tag{2.82}$$

We can see that the average generalization error made by the estimator f consists of two different model-dependent contributions. The first, called the *bias*, defines how well, on average, the model f is capable of estimating the desired function $E\left[y|\, x\right]$. A model that estimates every data set perfectly will have a zero bias. The variance tells how the model choice is influenced by the data set $\mathcal{D}$. If for different data sets, the estimator f differs significantly, then this term will increase in value thereby increasing the general mean squared error. Therefore there is a trade-off between both terms in the models choice. This is identical to the trade-off between the structural risk and the empirical risk for optimal model choice as explained in Section 2.1. One way to achieve this balance between bias and variance is by opposing an extra regularization term as was done in the Tikhonov optimization functional (2.10). The regularization acts as smoothing criterion on the possible solutions f thereby reducing the variance. This is controlled by the regularization hyperparameter γ.

Furthermore a priori knowledge of f^* can help to reduce the bias-variance trade-off. By a good choice of the class of possible solutions f one can reduce the bias. For example, if one knows that a classification problem is linearly separable, it is better to chose a linear kernel k instead of a highly non-linear kernel. This is typically the idea of the structural risk minimization principle as explained previously.

A last method to control the bias-variance trade-off is to increase the number of data points N. By increasing the number of data points, one is allowed to use models with a higher complexity. This can be done by using non-linear kernels with a higher VC-dimension. These more complex models will be able to estimate the data more accurately thereby reducing the bias. The change in variance however will be limited because of the larger size of the data set. A larger the data set will put more restrictions on the model choice which results in an increasing inertia towards changes in the data set.

2.9 Support vector machines

One of the most popular kernel models are the SVM models introduced by Vapnik ([146],[147]). These models are characterized by a learning process that involves a constrained quadratic programming problem and the corresponding sparse solution. In recent literature many modifications and improvements have been made to the original formulation both for classification and regression problems. In this section we go deeper into the relation between problem formulation and the sparse solution. We focus only on the original classification formulation since this already gives an explanation for the most important characteristics of these models. For more information about the corresponding regression formulations and other modification to this algorithm one may consult the books: [28] and [119].

In Section 2.6.1 we introduced the Vapnik SVM formulation for classification. There we showed that in order to find the most optimal separating hyperplane that maximizes a separating margin one has to solve the primal optimization problem 2.58.

In order to solve this optimization problem one formulates the Lagrangian function

$$\begin{aligned} \mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \\ - \sum_{i=1}^{N}\alpha_i\left(y_i\left(\mathbf{w}^T\varphi(\mathbf{x}_i) + b\right) - 1 + \xi_i\right) - \sum_{i=1}^{N}\beta_i\xi_i, \end{aligned} \tag{2.83}$$

where $\alpha = [\alpha_1\alpha_1 \dots \alpha_N]^T$ and $\beta = [\beta_1\beta_2 \dots \beta_N]^T$ are the Lagrange multipliers. Based on this Lagragian function one can formulated the Karush-Kuhn-Tucker optimality conditions for this constrained convex optimization problem (see also A.5):

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 \rightarrow \mathbf{w} = \sum\limits_{i=1}^{N} \alpha_i y_i \varphi\left(\mathbf{x}_i\right), \\ \frac{\partial \mathcal{L}}{\partial b} = 0 \rightarrow \sum\limits_{i=1}^{N} \alpha_i y_i = 0, \\ \frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \rightarrow C - \alpha_i - \beta_i = 0,\ \forall i \in \{1, \ldots, N\}, \\ y_i\left(\mathbf{w}^T \varphi\left(\mathbf{x}_i\right) + b\right) \geq 1 - \xi_i,\ \forall i \in \{1, \ldots, N\}, \\ \xi_i \geq 0,\ \forall i \in \{1, \ldots, N\}, \\ \alpha_i \geq 0,\ \forall i \in \{1, \ldots, N\}, \\ \beta_i \geq 0,\ \forall i \in \{1, \ldots, N\}, \\ \alpha_i\left(y_i\left(\mathbf{w}^T \varphi\left(\mathbf{x}_i\right) + b\right) - 1 + \xi_i\right) = 0,\ \forall i \in \{1, \ldots, N\}, \\ \beta_i \xi_i = 0,\ \forall i \in \{1, \ldots, N\}. \end{cases} \tag{2.84}$$

The last two of these relations are the complementary slackness conditions. These will play an important role in explaining the sparseness in these models.

Analogously as for the LS-SVM formulation one converts the primal problem formulation into its Wolf dual. This is done by substituting the equalities of the KKT conditions and eliminating the primal variables $\mathbf{w}, b, \xi, \beta$.

$$\begin{cases} \max\limits_{\alpha} \quad -\frac{1}{2} \sum\limits_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) + \sum\limits_{i=1}^{N} \alpha_i \\ \text{s.t.} \sum\limits_{i=1}^{N} \alpha_i y_i = 0, \\ \qquad 0 \leq \alpha_i \leq C,\ \forall i \in \{1, \ldots, N\}. \end{cases} \tag{2.85}$$

We used the property that the kernel is an inner product in feature space: $k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \left\langle \varphi\left(\mathbf{x}_i\right), \varphi\left(\mathbf{x}_j\right) \right\rangle_{\mathcal{H}}$. Once the optimum of the constrained quadratic optimization problem is found, the classification model will have an expression in the Lagrange multipliers or dual variables:

$$f\left(\mathbf{x}\right) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i k\left(\mathbf{x}, \mathbf{x}_i\right) + b\right). \tag{2.86}$$

If one compares the dual formulation of the Vapnik SVM formulation with the dual of the LS-SVM formulation (2.63) one can see two major differences. The first difference is that the dual variables of the SVM satisfy the box-constraints $0 \leq \alpha_i \leq C$, $\forall i = 1, \ldots, N$. This will lead to a solution where some of the dual variables are at the boundaries of these constraints. Since for each α_i there is a corresponding input point $(\mathbf{x}_i, y_i)$ the only input points with a $\alpha_i \neq 0$ will have a contribution in the evaluation of new input points. This is the reason why they are called *support vectors*.

A second difference between both dual optimization problems is the size of the solution space. In the LS-SVM case we have seen that the Hessian matrix is strictly positive definite. Therefore the solution is always unique. In the case of SVMs the Hessian matrix is $H = \mathrm{diag}(\mathbf{y})K\mathrm{diag}(\mathbf{y})$ that is a positive semi-definite matrix. The solution space of this optimization problem therefore has the dimension $N - \mathrm{rank}(K)$. So, the solution of this optimization problem is global but not necessarily unique.

2.9.1 Sparseness and its consequences for large scale applications

We have shown how the SVM classification formulation leads to a constrained quadratic programming problem of dimension N. Because of the inequality constraints, this optimization problem can not be rewritten as a linear system. Typically *interior point methods* are used to solve these type of optimization problems ([96],[97]). In these methods the solution is computed in an iterative way following a trajectory in the interior of the feasible region demarcated by the constraints of the optimization problem. Therefore at each iteration step a linear system, comparable in size and form to the LS-SVM training step, has to be solved. Hence that the memory requirements of these methods are also quadratic in the number of data points N. So, the memory complexity for solving this quadratic programming problem is $\mathcal{O}(N^2)$. An important aspect in the SVM model is the sparseness of the solution. Many of the α_i are equal to zero. An example of a real spectrum of the α-values for a classification of two overlapping Gaussian clusters is given in Figure 2.7. The reason for the sparseness lies in the complementary slackness conditions. According to the KKT conditions these complementary slackness conditions can be rewritten as:

$$\alpha_i \left(y_i \left(\mathbf{w}^T \varphi\left(\mathbf{x}_i\right) + b \right) - 1 + \xi_i \right) = 0, \ \forall i \in \{1, \ldots, N\}, \tag{2.87}$$

$$\left(\alpha_i - C\right) \xi_i = 0, \ \forall i \in \{1, \ldots, N\}. \tag{2.88}$$

For every training point $(\mathbf{x}_i, y_i)$ these conditions have to be satisfied. Let us go through the different possible scenarios:

- Assume the data point $(\mathbf{x}_i, y_i)$ is misclassified upon the defined margin. According to this SVM theory a slack variable $\xi_i \neq 0$ is introduced. As a result of the complementary slackness conditions the following

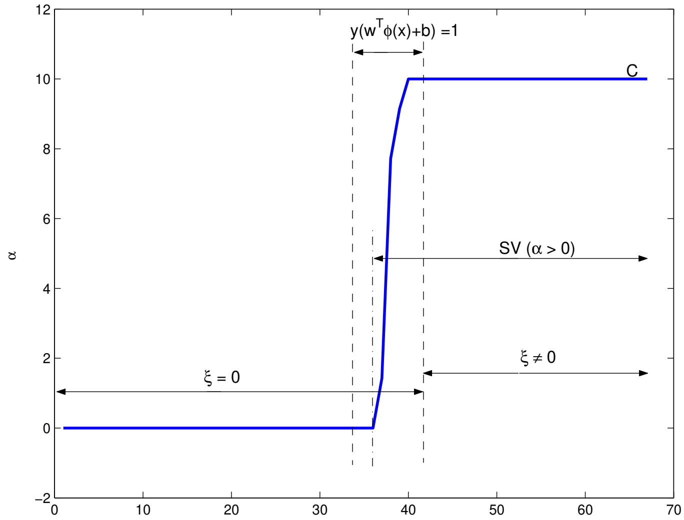

Figure 2.7: The sorted α-spectrum of an SVM model. The support vectors are indicated by the region SV.

holds:

$$\begin{cases} \alpha_i = C, \\ \xi_i = 1 - y_i\left(\mathbf{w}^T\varphi\left(\mathbf{x}_i\right) + b\right). \end{cases} \tag{2.89}$$

therefore these points are support vectors.

- Assume the data point $(\mathbf{x}_i, y_i)$ is correctly classified upon the defined margin. In this case there is no need for a slack variable and $\xi_i = 0$. One can make two distinctions. If for the data point holds that $y_i\left(\mathbf{w}^T\varphi\left(\mathbf{x}_i\right) + b\right) > 1$ the complementary slackness conditions result into :

$$\alpha_i = 0. \tag{2.90}$$

Notice that in the case of a good classifier, that separates the data, the majority of the data points belong to this group. These points are not support vectors. A second group are the data points lying on the hyperplanes defining the separating margin such that $y_i\left(\mathbf{w}^T\varphi\left(\mathbf{x}_i\right) + b\right) = 1$ for these points holds that

$$0 \leq \alpha_i \leq C. \tag{2.91}$$

Most of these points also are support vectors. All these relations are schematically visualized in Figure 2.7.

The above relations show that the KKT optimality conditions can also be formulated as a function of the Lagrange multipliers α :$\forall i$

$$\begin{array}{lll} \alpha_i = 0 & \Rightarrow & y_i f(\mathbf{x}_i) \geq 1, \\ 0 < \alpha_i < C & \Rightarrow & y_i f(\mathbf{x}_i) = 1, \\ \alpha_i = C & \Rightarrow & y_i f(\mathbf{x}_i) \leq 1. \end{array} \tag{2.92}$$

It is this formulation of the KKT conditions together with the sparseness of the solution that gives rise to chunking and other decomposition methods.

2.9.2 Chunking and other decomposition methods

In the previous section we described how the SVM training leads to a quadratic optimization problem with a memory complexity of $\mathcal{O}(N^2)$. For large data sets one soon runs into trouble because of the limited available RAM memory[7]. But the memory complexity can be reduced because of the sparse nature of the solution. This idea was first introduced by Vapnik ([145],[147]). Hereby one will exploit this sparseness to achieve a decomposition of the N dimensional QP problem into smaller ones. This is called *chunking.* The idea behind this strategy is the following. If one would train an SVM on a subsample of the large data set $\mathcal{D}$ then support vectors of this 'sub'-problem will probably also be support vectors of the large training problem. In other words, these are the only important data points and the other ones with $\alpha_i = 0$ are not essential. Therefore one can apply the following strategy. Initialize a training procedure where all N training points have $\alpha_i = 0$. Train an SVM on a small subset of the total amount of training point. Once trained, a new data set will be created containing only the support vectors from the previous training procedure. This is called the *working set.* To this working set new points are added that are sampled from the large data set $\mathcal{D}$. In this way we get a new data set on which a new SVM can be trained. This procedure can be repeated where one remains the support vectors in the working set. The Karush-Kuhn-Tucker conditions (2.92) are used both as a stopping criterion and as a measure to sample new data points in the working set. The last step of this chunking procedure solves the large QP problem. Notice hereby that the memory complexity reduces to $\mathcal{O}((\#SV)^2)$ where $\#SV$ is the number of support vectors. This still can cause memory problems if the $\#SV$ is large. The size of the working set, and the memory complexity of the corresponding QP problem, might grow beyond the limits of the machine.

[7] The memory complexity of the SVM regression formulation is $\mathcal{O}(4N^2)$. The sparseness is controlled by both C and ε of the Vapnik ε-insensitive loss function ([146],[147]).

Therefore Osuna *et al.* [100] modified the strategy of chunking into a method with a fixed working set. They showed that the large QP problem can be split into a series of smaller QP subproblems. Hereby the algorithm only updates a fixed size subset of multipliers α_i, while the others are kept constant. So every time m new points are added to the working set, m other have to be removed. Again the KKT conditions (2.92) play the role of stopping criterion and judge in order to control which points are added or removed. Notice that the core of this strategy is still based on solving a QP problem.

An extreme form of the chunking methodology with a fixed working set is the *sequential minimal optimization (SMO)* developed by Platt [107]. Hereby one optimizes the minimal subset of just two points at each iteration. The power of this technique lies in the fact that the solution of the optimization problem for two points can be done in an analytical way, which eliminates the need for a QP optimization module as part of the algorithm. Although this algorithm needs more iterations before convergence compared the other chunking algorithms, the algorithm is still much faster because of the few operations it needs and the fact that it does not have to apply any matrix-vector multiplications. This last property explains also the lower memory needs of the algorithm. The good design of SMO, as given by Platt, together with the improvements suggested by Keerthi *et al.* [71] make that the SMO strategy for training SVMs is one of the standards for large data sets.

Similar SMO strategies have also been developed for LS-SVM model as we will see in the next chapter. How sparse solutions can be implemented by changing the model formulation will be explained in Chapter 5. In Chapter 6 we will describe how we can achieve sparse solution in an ensemble learning setup.

2.10 Conclusions

In the previous sections we have shown two different of derivations kernel models with a quadratic loss function. In a first approach we started with an approximation approach in an RKHS. By optimizing the functional that balances the empirical risk and the structural risk one obtains a linearly parameterized model. By using the squared loss function for minimizing the empirical risk the optimal parameters of the model are found by solving a linear system. The same formulation can be derived starting form an optimization approach by defining a non-linear model in feature space. Also here one defines a cost function that balances between a structural risk or regularization and empirical risk. This last one is minimized in a least squares formulation.

Both methodologies apply for regression as well as classification problem. In both cases we can summarize the training of the model based on $\mathcal{D}$ by solving the linear system:

$$\begin{bmatrix} 0 & \mathbf{z}^T \\ \mathbf{z} & H \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{u} \end{bmatrix} \tag{2.93}$$

where the Hessian matrix H is defined by its elements $h_{ij} = z_i z_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) + \delta_{ij}\frac{1}{\gamma}$ and for classification $\mathbf{z} = \mathbf{y}, \mathbf{u} = \mathbf{1}_N$, for regression $\mathbf{z} = \mathbf{1_N}, \mathbf{u} = \mathbf{y}$. The final models respectively take the form $f\left(\mathbf{x}\right) = sign(\sum_{p=1}^{N} \alpha_p y_p k(\mathbf{x}_p, \mathbf{x}) + b)$ for classification and $f\left(\mathbf{x}\right) = \sum_{p=1}^{N} \alpha_p k(\mathbf{x}_p, \mathbf{x}) + b$ for regression.

The main characteristics of these models are:

- unique solution of the parameters α and b,
- the choice of the kernel offers a large flexibility,
- the computational model complexity is independent of the input dimension d,
- the computational model complexity is independent of dimension of the feature space,
- the parameters α and b of the model are directly connected to the model errors $\mathbf{e}$ since $\alpha = \gamma\mathbf{e}$,
- the computational complexity scales up with the number of input points N.

Although we did not discuss all links in detail, it should be mentioned that the presented models have many relations to other theories. The presented LS-SVM ([129]) and Regularization Networks ([108]) models we described for regression are closely related to Gaussian processes [81], Kriging ([73],[27]), kernel ridge regression [116]. But also the classification setup finds many similar or closely related derivations as there are: kernel Fisher discriminant analysis [87], proximal support vector machines [50] and Regularized Least-Square classification [113].

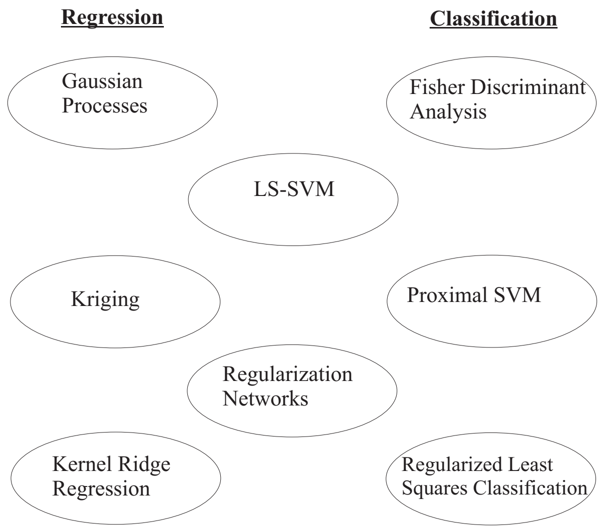

Figure 2.8: The presented LS-SVM ([129]) and Regularization Networks ([108]) models we described for regression are closely related to Gaussian processes [81], Kriging ([73],[27]), kernel ridge regression [116]. But also the classification setup finds many similar or closely related derivations as there are: kernel Fisher discriminant analysis [87], proximal support vector machines [50] and Regularized Least-Squares classification [113].

Chapter 3

Numerical Aspects of LS-SVM Training

In this chapter we will study the numerical characteristics involved in the training of the LS-SVM and RN models we introduced in the previous chapter. We will give an overview of the numerical procedures that are best suited for this specific problem and give guidance for choosing the best method according to the size of the problem. We will illustrate that for small training sets ($N < 1000$) the *Cholesky factorization is* the most recommendable algorithm. For larger training sets ($1000 < N < 5000$), however, iterative methods have more advantages. Implementation details will be discussed. All results will be verified experimentally.

3.1 Introduction

We have seen in the previous chapter that Kernel Models can be used for solving classification problems, regression and time-series prediction models. Several articles have shown that the generalization performance of these methods are among the best in a variety of application areas such as intrusion detection [93], text classification [68], genome analysis ([16],[92]),... Still an important topic of research is the use of kernel models on large data sets.

In this chapter, we investigate several algorithms for Least Squares Support Vector Machines (LS-SVM), introduced in [132], that involve solving linear systems. We have seen in Section 2.6 that the LS-SVM approach is a modification to the Vapnik formulation, where one avoids the inequality constraints in the QP-problem. The resulting constrained optimization problem for the LS-SVM reduces to the solution of a set of linear equations.

This gives us eventually the opportunity to solve large problems without applying iterative chunking. The disadvantage is that the sparseness of the solution is lost. The main focus of this chapter is the comparison of different direct and iterative solvers for an LS-SVM. This comparison will be done based on the computational and memory complexity of the models. Since these algorithms have a memory complexity of $\mathcal{O}(N^2)$ problems will occur for large data sets. To overcome this problem two solutions can be formulated.

The first solution is to store the necessary information outside the RAM memory of the computer such as the hard disk. Algorithms that uses this strategy are called 'out-of-core' algorithms. In such situations the number of input-output I/O operations need to be reduced as much as possible because they are very time consuming.

A second solution uses a different approach. Instead of storing the Hessian matrix into RAM memory it is also possible that one recomputes this information every time it is needed. In this case the memory bottleneck is transferred into a computational problem. Although it is theoretically possible to recompute all necessary information on demand, it soon becomes very impractical because of the enormous computation time needed.

In both cases it is important to reduce the number of iterations the algorithms need for computing the solution of the optimization process. The methods we will propose consult the Hessian matrix at each iteration step. Reducing the number of iteration steps results in a decrease of the I/O operation for out-of-core algorithms or a decrease of the number of times the Hessian matrix needs to be recomputed. For smaller problems where the Hessian matrix can be stored in memory, reduction of the number of iteration steps results in a faster algorithm.

Notice that because of the close relation between LS-SVMs, Regularization Networks (Section 2.4), Kriging ([73],[27]) and Gaussian Processes [81] most of the discussed solutions can be applied to all of these methods.

We will illustrate that for small training sets ($N < 1000$) the *Cholesky factorization* is still the most recommended algorithm. For large training sets ($1000 < N < 5000$), however, the iterative methods have more advantages. In this chapter we will mainly focus on two iterative algorithms: the *successive overrelaxation method* and the *conjugate gradient method.* For the first, the use of acceleration methods (GM–acceleration) and adaptations for memory efficiency (block-SSOR) will be shown. Although these methods give good results in reducing the number of iteration steps, they are outperformed by the Krylov methods (conjugate gradient). Because we need to solve two almost identical linear systems, we can use the *block con-*

jugate gradient method. This method proves to give a faster convergence on both systems simultaneously in comparison with solving the two linear systems separately.

3.2 Training an LS-SVM model

Training an LS-SVM model based on $\mathcal{D}$ for both classification and regression tasks can be written as solving the linear system of the form:

$$\begin{bmatrix} 0 & \mathbf{z}^T \\ \mathbf{z} & H \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{u} \end{bmatrix} \tag{3.1}$$

where Hessian matrix H is defined by its elements $H_{ij} = z_i z_j k(\mathbf{x}_i, \mathbf{x}_j) + \delta_{ij}\frac{1}{\gamma}$ and for classification $\mathbf{z} = \mathbf{y}, \mathbf{u} = \mathbf{1}_N$, for regression $\mathbf{z} = \mathbf{1_N}, \mathbf{u} = \mathbf{y}$. The final models respectively take the form $f(\mathbf{x}) = sign(\sum_{p=1}^{N} \alpha_p k(\mathbf{x}_p, \mathbf{x}) + b)$ for classification and $f(\mathbf{x}) = \sum_{p=1}^{N} \alpha_p k(\mathbf{x}_p, \mathbf{x}) + b$ for regression.

However, this linear system is not positive definite because of the occurrence of a zero in the top-left corner of the matrix, which makes the linear system more difficult to solve. These type of problems are called *saddle point problems.* (For further information we refer the reader to [114]). In our setup we will overcome this problem by rewriting it into the following formulation [131]

$$\begin{bmatrix} s & \mathbf{0}_N^T \\ \mathbf{0}_N & H \end{bmatrix} \begin{bmatrix} b \\ \alpha + bH^{-1}\mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{z}^T \left(H^{-1}\mathbf{u}\right) \\ \mathbf{u} \end{bmatrix}, \tag{3.2}$$

$$s = \mathbf{z}^T H^{-1} \mathbf{z} > 0. \tag{3.3}$$

This new linear system is positive definite. This opens many opportunities for using fast and efficient numerical methods. The solution of the problem can be found in the following three step method [131]:

$$\begin{array}{l} \text{step 1: } \begin{cases} H\eta = \mathbf{z}, \\ H\nu = \mathbf{u} \end{cases} \\ \text{step 2: } s = \mathbf{z}^T \eta \\ \text{step 3: } \begin{cases} b = \frac{\eta^T \mathbf{u}}{s}, \\ \alpha = \nu - b\eta. \end{cases} \end{array} \tag{3.4}$$

The next important issue is to find a good method for solving the two sets of linear equations with the same positive definite coefficient matrix $H \in \mathbb{R}^{N \times N}$. We will discuss several methods that can be found in the literature. In the sequel the two linear systems $H\eta = \mathbf{z}$ and $H\nu = \mathbf{u}$ will be denoted by $H\zeta = \mathbf{t}$ where ζ will stand for η or ν and $\mathbf{t}$ for $\mathbf{z}$ or $\mathbf{u}$, respectively.

3.3 Direct methods

One of the standard numerical methods for solving linear systems $H\zeta = \mathbf{t}$ is the *LU-factorization* [57]. In this method the matrix H, which is not required to be positive definite, is decomposed as a product of a lower triangular L and upper triangular U matrix. So $H = LU$, which gives

$$H\zeta = LU\zeta = \mathbf{t} \Leftrightarrow \begin{cases} L\mathbf{z} = \mathbf{t}, \\ U\zeta = \mathbf{z}. \end{cases} \tag{3.5}$$

The latter equations are easy to solve via forward and backward substitution.

Since H is a symmetric positive definite matrix (see Appendix B) one can prove that there exists a unique matrix G, lower triangular with positive diagonal entries, so that $H = GG^T$. With this special feature we can rewrite the previous method:

$$H\zeta = \mathbf{t} \Leftrightarrow \begin{cases} G\mathbf{z} = \mathbf{t}, \\ G^T\zeta = \mathbf{z}. \end{cases} \tag{3.6}$$

This method is known as the *Cholesky factorization* [57]. The accuracy of the solution given by direct methods is determined by the condition number and the machine precision. An advantage of this method is that its computation time does not depend on the condition number of the matrix H. The number of floating point operations needed is of the order $\mathcal{O}\left(\frac{N^3}{3}\right)$ (N is the dimension of the system). An disadvantage of direct methods is that the matrix H has to be completely stored in memory. This is of course impossible when we go to very large matrices. If one want to apply methods on large matrices one can apply out-of-core strategies. In this case the matrix will not be fully stored in main memory but on an external device. Necessary submatrices will be loaded for the external device into memory when needed. However, this will not be handled in this thesis. Therefore, it is interesting to also consider less computationally intensive methods with smaller memory requirements. For smaller problems, up to say $N = 1000$, our advise is to invoke Cholesky factorization.

3.4 Iterative methods: Jacobi, Gauss-Seidel and SOR

Iterative methods are characterized by the fact that they start from an initial guess $\zeta^{(0)}$. At each iteration step they create a new candidate solution. This sequence of iteration vectors $\left\{\zeta^{(0)}, \zeta^{(1)}, \zeta^{(2)}, \ldots, \zeta^{(l)}, \ldots\right\}$ will converge to the optimal solution ζ^*. In general this iterative nature, as such, is an important advantage. We can terminate the iteration process once an intermediate solution is satisfactory. This can be an advantage in time-restricted on-line processes. We will overview the basic iterative methods ([156],[149]). Hereby we make the algorithms gradually more complex in order to improve the convergence rate and memory efficiency.

One of the simplest methods is the *Jacobi method.* Here the components of the vectors are calculated as:

$$\begin{array}{l} \text{For} \quad i = 1:1:N \\ \qquad \zeta_i^{(l+1)} = \frac{1}{h_{ii}} \left(t_i - \sum\limits_{j=1}^{i-1} h_{ij} \zeta_j^{(l)} - \sum\limits_{j=i+1}^{N} h_{ij} \zeta_j^{(l)} \right) \\ \text{end} \end{array}$$

where the i-th component of ζ after l iterations is written as $\zeta_i^{(l)}$.

The second method is the *Gauss-Seidel* method. In this method the new information about the first $i-1$ of $\zeta^{(l+1)}$ components is used:

$$\begin{array}{l} \text{For} \quad i = 1:1:N \\ \qquad \zeta_i^{(l+1)} = \frac{1}{h_{ii}} \left(t_i - \sum\limits_{j=1}^{i-1} h_{ij} \zeta_j^{(l+1)} - \sum\limits_{j=i+1}^{N} h_{ij} \zeta_j^{(l)} \right) \\ \text{end} \end{array}$$

The third method is the Successive Overrelaxation (SOR) method:

$$\begin{array}{l} \text{For} \quad i = 1:1:N \\ \qquad \zeta_i^{(l+1)} = \frac{\omega}{h_{ii}} \left(t_i - \sum\limits_{j=1}^{i-1} h_{ij} \zeta_j^{(l+1)} - \sum\limits_{j=i+1}^{N} h_{ij} \zeta_j^{(l)} \right) + (1-\omega)\, \zeta_i^{(l)} \\ \text{end} \end{array}$$

This method was already successfully applied for SVM training by Mangasarian and Musicant in [82]. The convergence of this method depends on the value of the *overrelaxation parameter* ω. In general this parameter can be found from the minimization problem

$$\omega_{optimal} = \underset{\omega}{\operatorname{argmin}}\ \rho\left(\left(D - \omega L\right)^{-1}\left(\omega U + \left(1 - \omega\right) D\right)\right), \tag{3.7}$$

where $\rho\left(A\right)$ denotes the spectral radius of a matrix A. The matrices L, D and U are the lower diagonal, diagonal and upper diagonal submatrices of H such that $H = L + D + U$.

Finding the optimal overrelaxation parameter based on this formula is computationally too expensive. When the matrix H is symmetric and positive definite, there exist proofs [156] that the algorithm will converge for all overrelaxation parameters $0 < \omega < 2$. Yet, there may exist large differences in the convergence rate for different overrelaxation parameters.

3.4.1 Acceleration methods for SOR

A first improvement that we investigate to increase the convergence rate of the SOR-algorithm is the use of the Graves-Morris (GM) Acceleration [29]. This is a generalization of Aitken's Δ^2 process [15]. This method constructs a new sequence of solution vectors $\mathbf{t}^{(l)}$ based on previous iterations

$$\mathbf{t}^{(l)} = \zeta^{(l)} - \left(\zeta^{(l)} - \zeta^{(l-1)}\right) \frac{\left(\zeta^{(l-1)} - \zeta^{(l-2)}\right)^T \left(\zeta^{(l-1)} - \zeta^{(l-2)}\right)}{\left(\zeta^{(l-1)} - \zeta^{(l-2)}\right)^T \left(\zeta^{(l)} - 2\zeta^{(l-1)} + \zeta^{(l-2)}\right)}. \tag{3.8}$$

After the first three iterations this new sequence $\mathbf{t}^{(l)}$ will replace the original sequence. The acceleration method not only reduces the number of iterations, the resulting method is also less sensitive to the value of the overrelaxation parameter. The corresponding speed-up is experimentally verified as will be shown further.

3.4.2 Symmetric successive overrelaxation (SSOR)

A alternative SOR algorithm exploits the structure of the matrix. Because we know that the matrix H is always symmetric, we can exploit this fact. Typical for these matrices is that one can develop a backward SOR [57]. This means that we can update our solution vector in reverse direction. This will be combined with the traditional forward SOR in the following two-step procedure:

%Forward Loop

For $\quad i = 1:1:N$
$$\zeta_i^{\left(\frac{l+1}{2}\right)} = \frac{\omega}{h_{ii}}\left(t_i - \sum_{j=1}^{i-1} h_{ij}\zeta_j^{\left(\frac{l+1}{2}\right)} - \sum_{j=i+1}^{N} h_{ij}\zeta_j^{(l)}\right) + (1-\omega)\,\zeta_i^{(l)}$$
end
%Backward Loop
For $\quad i = N:-1:1$
$$\zeta_i^{(l+1)} = \frac{\omega}{h_{ii}}\left(t_i - \sum_{j=1}^{i-1} h_{ij}\zeta_j^{\left(\frac{l+1}{2}\right)} - \sum_{j=i+1}^{N} h_{ij}\zeta_j^{(l+1)}\right) + (1-\omega)\,\zeta_i^{\left(\frac{l+1}{2}\right)}$$
end

where $\zeta^{\left(\frac{l+1}{2}\right)}$ is an intermediate step. These algorithms are very popular for preconditioning conjugate gradient algorithms.

3.4.3 Block successive overrelaxation and block symmetric successive overrelaxation

Next, we will divide the vectors and matrices H, ζ and $\mathbf{t}$ in subvectors and submatrices. In this way we can formulate the block-SOR [149, p78-82]. The linear equation $H\zeta = \mathbf{t}$ is formulated as

$$\begin{bmatrix} H_{11} & H_{12} & \dots & H_{1p} \\ H_{21} & H_{22} & \dots & H_{2p} \\ \dots & \dots & \dots & \dots \\ H_{p1} & H_{p2} & \dots & H_{pp} \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \\ \dots \\ Z_p \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \\ \dots \\ T_p \end{bmatrix}. \tag{3.9}$$

The block-SOR method is given by

For $\quad i = 1:1:N$
$$Z_i^{(l+1)} = \omega H_{ii}^{-1}\left(T_i - \sum_{j=1}^{i-1} H_{ij}Z_j^{(l+1)} - \sum_{j=i+1}^{N} H_{ij}Z_j^{(l)}\right) + (1-\omega)\,Z_i^{(l)}$$
end.

As stated in [149], the use of these block-methods can give some improvements on convergence rate, although it is not guaranteed. However, we experimentally checked this feature and found that it is almost always fulfilled.

Also the SSOR method can be redefined using the block-formulation. The Block Symmetric Overrelaxation is given by:

%Forward Loop
For $\quad i = 1:1:N$

$$Z_i^{\left(\frac{l+1}{2}\right)} = \omega H_{ii}^{-1}\left(T_i - \sum_{j=1}^{i-1} H_{ij} Z_j^{\left(\frac{l+1}{2}\right)} - \sum_{j=i+1}^{N} H_{ij} Z_j^{(l)}\right) + (1-\omega)\, Z_i^{(l)}$$

end

%Backward Loop

For $\quad i = N : -1 : 1$

$$Z_i^{(l+1)} = \omega H_{ii}^{-1}\left(T_i - \sum_{j=1}^{i-1} H_{ij} Z_j^{\left(\frac{l+1}{2}\right)} - \sum_{j=i+1}^{N} H_{ij} Z_j^{(l+1)}\right) + (1-\omega)\, Z_i^{\left(\frac{l+1}{2}\right)}$$

end.

Since for large data sets we cannot store the complete matrix in memory, this method has the advantage of using less I/O operations in the case that we need to store the H matrix in an external storage device. Each loop of the SSOR through the matrix H can reuse the last blocks of the previous loop. This finds its cause in the intelligent switching between forward and backward propagation intrinsically done by the SSOR. Depending on the size of the blocks, this eliminates a big swap between hard disk and main memory of the computer.

For all these methods the convergence is proven as long as the overrelaxation parameter is $0 < \omega < 2$.

3.4.4 Stopping criteria and convergence properties

When one solves a linear system, the exact solution is never reached because of the limited accuracy in the floating point arithmetic. Especially in iterative methods it is important to know what constitutes an acceptable approximation to the solution. Based on this knowledge a stopping criterion can be formulated. Often it is desirable to have an approximate solution for which some standard norm of the error, say the two norm or ∞-norm, is less than a certain tolerance.

It would be nice if we could monitor the evolution of the *true error* $H^{-1}\mathbf{t} - \zeta^{(l)}$ during the iterations. Then it would be easy to know when the desired accuracy is reached. Unfortunately this is impossible, the only information we have is the residual $\mathbf{r}^{(l)} = \mathbf{t} - H\zeta^{(l)}$. But we can estimate the true desired error for iterative methods using the residual by the following rule, which gives the relation between the relative error norm and the relative residual norm [59, p. 107]

$$\frac{1}{\text{cond}_2\left(H\right)} \frac{\left\|\mathbf{t} - H\zeta^{(l)}\right\|_2}{\|\mathbf{t}\|_2} \leq \frac{\left\|H^{-1}\mathbf{t} - \zeta^{(l)}\right\|_2}{\|H^{-1}\mathbf{t}\|_2} \leq \text{cond}_2\left(H\right) \frac{\left\|\mathbf{t} - H\zeta^{(l)}\right\|_2}{\|\mathbf{t}\|_2}, \tag{3.10}$$

where $\text{cond}_2\left(H\right) = \left\|H^{-1}\right\|_2 \|H\|_2$ is the condition number of matrix H. Given a tolerance ε one can always iterate until the desired accuracy on $\frac{\left\|\mathbf{r}^{(l)}\right\|_2}{\left\|\mathbf{r}^{(0)}\right\|_2} = \frac{\left\|\mathbf{t} - H\zeta^{(l)}\right\|_2}{\|\mathbf{t}\|_2}$ is reached[1]. Controlling this residual then gives a lower and upper limit on the true error. A stopping criterion based on this idea will be used in the Krylov methods we will discuss in the next Section 3.5.5.

Another stopping criterion ([149],[156],[139]) is based on the differences $\Delta\zeta_i^{(l)} = \zeta_i^{(l)} - \zeta_i^{(l-1)}$. At the beginning of the process a parameter ε is defined to control the accuracy of the algorithm. The algorithm will stop at iteration step l when $\max_i \left|\Delta\zeta_i^{(l)}\right| < \varepsilon$. This corresponds to the ∞-norm $\left\|\Delta\zeta^{(l)}\right\|_\infty = \max_i \left|\Delta\zeta_i^{(l)}\right|$. Computationally this rule is very easy because it does not need any matrix-vector multiplication as would be the case if we would use the residual $\mathbf{r}^{(l)} = \mathbf{t} - H\zeta^{(l)}$. However, for some iterative methods it is known that this rule can lead to termination before the desired level of accuracy is achieved. Our experiments showed that this rule is sufficient. The experiments showed that demanding a high accuracy takes high computational time for large matrices. For our applications, however, very high accuracy for the solution vector is not always needed. The classification or regression task is not much influenced by small changes of the accuracy of the solution.

3.5 Iterative methods: Krylov methods

An important group of iterative methods are Krylov methods. The *conjugate gradient method* is suitable for matrices with a symmetric positive definite linear system ([57],[122],[48],[59],[97]). Its use was already experimentally verified in [131] and [141]. The idea is that the solution of the linear system is also the minimum of a quadratic convex cost function :

$$\zeta^* = \arg\min_\zeta \frac{1}{2}\zeta^T H\zeta - \mathbf{t}^T\zeta \Rightarrow H\zeta^* = \mathbf{t}. \tag{3.11}$$

[1] In all the experiments that follow, we use $\mathbf{x}^{(0)} = \mathbf{0}$ as the starting point of the iterative procedure.

The search procedure for this minimum is done via the conjugate directions of the matrix H. This leads to the following algorithm to solve $H\zeta = \mathbf{t}$ [57]:

Input $\zeta^{(0)},\ \mathbf{r}^{(0)} = H\zeta^{(0)} - \mathbf{t},\ \mathbf{p}^{(0)} = -\mathbf{r}^{(0)}$
While $\mathbf{r}^{(l)} \neq \mathbf{0}_N$
  $\tau^{(l)} = \frac{\mathbf{r}^{(l)T}\mathbf{r}^{(l)}}{\mathbf{p}^{(l)T}H\mathbf{p}^{(l)}}$
  $\zeta^{(l+1)} = \zeta^{(l)} + \tau^{(l)}\mathbf{p}^{(l)}$
  $\mathbf{r}^{(l+1)} = \mathbf{r}^{(l)} + \tau^{(l)}H\mathbf{p}^{(l)}$
  $\beta^{(l)} = \frac{\mathbf{r}^{(l+1)T}\mathbf{r}^{(l+1)}}{\mathbf{r}^{(l)T}\mathbf{r}^{(l)}}$
  $\mathbf{p}^{(l+1)} = -\mathbf{r}^{(l)} + \beta^{(l+1)}\mathbf{p}^{(l)}$
  $l = l + 1$
end

In this formulation, $\mathbf{r}^{(l)}$ and $\mathbf{p}^{(l)}$ are the residuals and conjugate directions updated at each iteration step respectively. The most demanding part in this algorithm is the matrix-vector product $H\mathbf{p}^{(l)}$ with a computational complexity of $\mathcal{O}\left(N^2\right)$. When the matrix H cannot be stored in the main memory, the matrix-vector computation is done row by row where each time the individual rows of H are recomputed. There are of course more intelligent procedures to do this matrix-vector multiplication by exploiting the symmetric structure of the matrix H.

A basic procedure would be storing those elements of each row that come back in the next rows because of the symmetric property of the matrix. This avoids the recomputation of those elements or the I/O operation of reloading them into the main memory from an external storage device. A second possibility to improve this computationally intensive step is computing the matrix-vector product on a parallel or distributed machine [46]. Remember that these solutions transfer the memory bottleneck to a computational or I/O bottleneck. Hence, it will be always advisable to reduce the number of iteration steps these algorithms need. In general one may conclude that the total computational complexity of the CG is $\mathcal{O}(lN^2)$ where l are the number of iterations before convergence.

The matrix in this matrix-vector product is always based on a kernel function k often with a local structure. This extra structure can also be exploited. A lot of work on how to compute these matrix vector products has been done by Beatson *et al.* ([6],[4],[5],[23]) Most of these methods focus on polynomial approximations of specific Radial Basis functions like the multiquadric RBF and the polyharmonic splines. Unfortunately most of these approximations are limited to low dimensional inputs d.

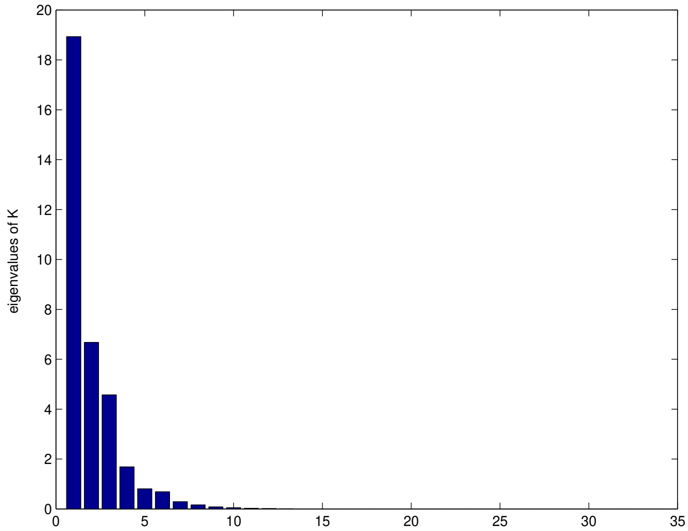

Figure 3.1: This figure shows a typical eigenvalue spectrum of the kernel matrix if one uses a RBF kernel. One can notice a rapid decay of the eigenvalue spectrum. This is the eigenvalue spectrum of the kernelmatrix of a classification problem with two overlapping Gaussian clusters with 35 data points.

3.5.1 Convergence properties of CG

The convergence of the CG algorithm is theoretically guaranteed in at most N steps. This is however a conservative upper limit. The convergence of the CG algorithm is very dependent on the eigenvalue spectrum and condition number of H. In [97, p. 114] a more intuitive explanation is given. If H has only r distinct eigenvalues, then the CG iteration will converge in at most r iterations.

In our application we often notice that only a few eigenvalues are distinct. An example is given in Figure 3.1. This figure shows a typical eigenvalue spectrum of the RBF kernel matrix. One can notice a rapid decay of the eigenvalue spectrum. In this figure we show the eigenvalue spectrum of the kernel matrix of a classification problem of two overlapping Gaussian clusters. The rapid decay of the eigenvalue spectrum was also noticed by Williams [152] for the Gaussian RBF kernel.

Also the condition number $\text{cond}_2(H)$ can give us some more insight in the convergence. As is stated in [139, p. 300], if the condition number

$\mathrm{cond}_2(H)$ is large but not too large, the convergence to a specified convergence can be expected in $\mathcal{O}\left(\sqrt{\mathrm{cond}_2(H)}\right)$ iterations. Convergence will be faster if the eigenvalue spectrum is clustered.

A second important aspect is the dependency of the condition number on the hyperparameters γ and σ. This can be easily seen since $H(\gamma,\sigma) = K(\sigma) + \frac{1}{\gamma} I_N$ (in case of the RBF kernel). These hyperparameters do not only influence the generalization capability of the SVM but also the convergence rate of the training procedure. We can see in Figure 3.2 that there is a large influence of the γ parameter on the condition number. The direct effect of the hyperparameter γ on the condition number is also discussed in Appendix B. One can see the in the case the γ goes to infinity, that the condition number also goes to infinity because it is equal to the condition number of the positive semi definite matrix K.

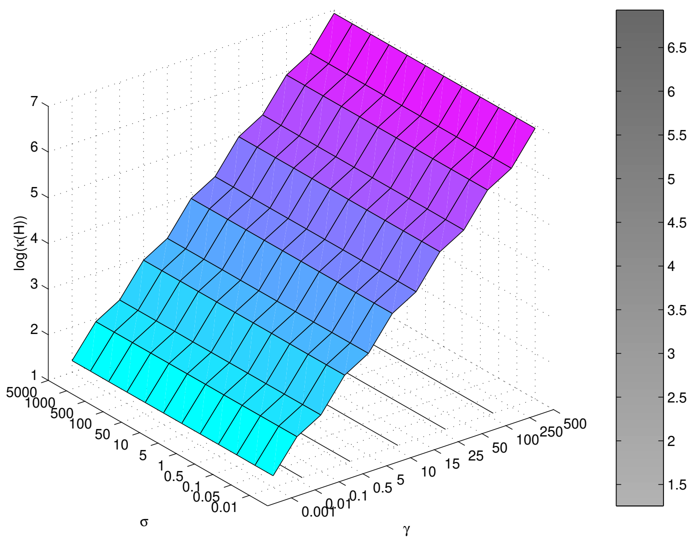

Figure 3.2: This figure shows the dependence of the condition number of the H matrix on the hyperparameters γ and σ. This plot is made by for German Credit data sets (grc) with 666 training data points and the RBF kernel. $\log(cond_2(H))$ is plotted with respect to γ and σ.

3.5.2 Preconditioning

The convergence of most iterative methods can be improved by preconditioning. Many of these preconditioning methods exploit the special structure of the matrix.

A general preconditioning method that has acceptable results on the linear systems involved in kernel models is the *incomplete Cholesky factorization* [77]. Lin *et al.* showed that this preconditioning can save some CG iteration but is only beneficial if the matrix calculations inside the incomplete Cholesky are cheap. Examples are cases where the computation of the whole kernel matrix K takes about the same time as doing the matrix-vector multiplication. Further information about preconditioning of the original linear system (3.1) can be found in [72].

3.5.3 The condition number, regularization parameter and perturbation analysis

In the previous section we have seen how the condition number of the matrix H influences the convergence of the iterative procedures. Here, we show that this condition number also plays an important role in the learning characteristics of the models w.r.t. the sensitivity to perturbations in the data. We study these features based on perturbation analysis of the system. In perturbation analysis one studies how the solution of a model is affected by a small change or disturbance of the system. In our case of learning, the solution of the model depends on the data. These data may be contaminated by noise. In our models noise is only assumed on the y-values. However, also the input data $\mathbf{x}$ might be the result of a measurement and therefore contain errors. Hereby we assume that the error on the input data is much smaller than on the y-values.

Knowing that data is contaminated, one wants to know how the strong the dependency of the models is on the contamination. Are the models stable or do they change completely by a small disturbance? To study this we will use the results of a perturbation analysis of the linear system involved in the training process. Here, we will only focus on the effect it has on the α's of model. This will be done for models without a b-term.

For these models it was shown that if one assumes only a disturbance of the y-values, given by $\Delta\mathbf{y}$, the change in the result of the training process $\Delta\alpha$ is bounded [30]: (assuming that $\mathbf{y} \neq \mathbf{0}_d$)

$$\frac{1}{\mathrm{cond}_2\left(K + \frac{1}{\gamma}I\right)} \frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2} \leq \frac{\|\Delta\alpha\|_2}{\|\alpha\|_2} \leq \mathrm{cond}_2\left(K + \frac{1}{\gamma}I\right) \frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2}. \tag{3.12}$$

Since the condition number $\mathrm{cond}_2\left(K + \frac{1}{\gamma}I\right) = \frac{\lambda_{\max}+\frac{1}{\gamma}}{\lambda_{\min}+\frac{1}{\gamma}}$ (see Appendix B) where $\lambda_{\min}$ and $\lambda_{\max}$ are respectively the minimum and maximum eigenvalue of K, one gets:

$$\frac{\lambda_{\min}+\frac{1}{\gamma}}{\lambda_{\max}+\frac{1}{\gamma}}\frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2}\leq\frac{\|\Delta\alpha\|_2}{\|\alpha\|_2}\leq\frac{\lambda_{\max}+\frac{1}{\gamma}}{\lambda_{\min}+\frac{1}{\gamma}}\frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2}. \tag{3.13}$$

This relation shows that by a possible disturbance of y-values, caused by a changed of the noise level or caused by outlier data, a change in the α's can be controlled directly by the regularization parameter γ. This relation shows us that the bounds on $\frac{\|\Delta\alpha\|_2}{\|\alpha\|_2}$ become monotonically tighter if one increases the regularization (by decreasing γ). In the limit where there is no regularization ($\gamma=\infty$) the bounds are very loose:

$$\frac{\lambda_{\min}}{\lambda_{\max}}\frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2}\leq\frac{\|\Delta\alpha\|_2}{\|\alpha\|_2}\leq\frac{\lambda_{\max}}{\lambda_{\min}}\frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2}. \tag{3.14}$$

In the other case that $\gamma\to 0$ one can see that $\frac{\|\Delta\alpha\|_2}{\|\alpha\|_2}=\frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2}$. But, this situation is not realistic since one only performs a regularization and the data has no influence in the in the model choice.

3.5.4 Block conjugate gradient

Since the conjugate gradient directions for both linear systems only depend on H, both linear systems $H\eta=\mathbf{z}, H\nu=\mathbf{u}$ can be solved simultaneously: $H\begin{bmatrix}\eta & \nu\end{bmatrix}=\begin{bmatrix}\mathbf{z} & \mathbf{u}\end{bmatrix}$. This method is called the *block conjugate gradient method (* [59, p. 113],[99]).

The algorithm to solve $HZ=T$ where $H\in\mathbb{R}^{n\times n}$; $T=\begin{bmatrix}\mathbf{z} & \mathbf{u}\end{bmatrix}$, $Z=\begin{bmatrix}\eta & \nu\end{bmatrix}\in\mathbb{R}^{n\times 2}$ is given by:

Input $Z^{(0)}, R^{(0)}=HZ^{(0)}-T, P^{(l)}=-R^{(l)}$
While $R^{(l)}\neq 0$
  $\tau^{(l)}=\left(P^{(l)T}HP^{(l)}\right)^{-1}R^{(l)T}R^{(l)}$
  $Z^{(l+1)}=Z^{(l)}+P^{(l)}\tau^{(l)}$
  $R^{(l+1)}=R^{(l)}+HP^{(l)}\tau^{(l)}$
  $\beta^{(l)}=\left(R^{(l)T}R^{(l)}\right)^{-1}R^{(l+1)T}R^{(l+1)}$
  $P^{(l+1)}=-R^{(l+1)}+P^{(l)}\beta^{(l+1)}$
  $l=l+1$
end

Note that the algorithm is only well-defined if the matrices $P^{(l)}$ and $R^{(l)}$ posses full rank. Because $\operatorname{span}\left\{P^{(0)},\ldots,P^{(l)}\right\}=\operatorname{span}\left\{R^{(0)},\ldots,R^{(l)}\right\}$

the ranks of both matrices $P^{(l)}$ and $R^{(l)}$ are the same. Therefore we can monitor the stability of the algorithm by calculating the rank of the matrix $P^{(l)T}\ P^{(l)}$. As long as this matrix maintains its full rank, the stability is guaranteed. If not, which means that columns become linearly dependent, the equations corresponding to dependent columns can be treated separately. The algorithm will be continued with the remaining equations.

The block conjugate gradient method is in some occasions to be preferred over CG. The following arguments were given in [99]:

1. To solve the two systems the algorithm will need at most $N/2$ iterations and may involve less work than solving the two systems separately.
2. If the matrix has several eigenvalues widely separated from the others, block CG will converge significantly faster than the CG.
3. If the matrix is stored on a secondary device or needs to be regenerated at every use, block CG can be implemented much more efficiently.

These features were experimentally verified and will be discussed in the next Section.

3.5.5 The starting and stopping criterion for CG and block-CG algorithm

We previously explained for SSOR (Section 3.4.4) that a good stopping criterion is essential. The optimal stopping criterion for our application, described in [141], is a combined set of rules that keep track of the change in the residual vector $\mathbf{r}^{(l)}$ and the evolution of the corresponding cost function $J\left(\zeta\right)$. The CG algorithm will stop the iteration when at least one of the following three criteria has been reached:

- $l \geq l_{\max}$,
- $\left\|\mathbf{r}^{(l)}\right\|_2 \leq \varepsilon_1 \left\|\mathbf{r}^{(0)}\right\|_2$,
- $J\left(\zeta^{(l-1)}\right) - J\left(\zeta^{(l)}\right) \leq \varepsilon_2$.

The last criterion is not often mentioned in literature. In our experiments we often noticed that the evolution of the norm of residual $\mathbf{r}^{(l)}$ is usually not monotonically decreasing. The evolution of $J\left(\zeta\right)$ does monotonically decrease and can be very easily monitored by the following reformulation: $J\left(\zeta^{(l)}\right) = \frac{1}{2}\zeta^{(l)T}\left(H\zeta^{(l)} - \mathbf{t}\right) - \frac{1}{2}\zeta^{(l)T}\mathbf{b} = -\frac{1}{2}\zeta^{(l)}\left(\mathbf{r}^{(l)} - \mathbf{t}\right)$. The constants

$l_{\max}, \varepsilon_1$ and ε_2 are determined by the user according to required numerical accuracy. In most application $\varepsilon_1 = \varepsilon_2 = 10^{-9}$ is sufficient. For the block-CG these criteria are applied to both systems separately.

Based on the work in SVM regression problems by Smola and Schölkopf [125], Keerthi [70] recently developed a stopping criterion based on the *duality gap* between the primal optimization problem and the dual problem formulation as is typically known form interior point methods. This means for classification between (2.59) and (2.65), and for regression between (2.67) and (2.72). Using this duality gap is a common practice in optimization algorithms. For the details we refer to [70].

In [53] yet another stopping criterion was advised. By monitoring a second quadratic function slightly different to (3.11) one is able to compute an upper and lower limit on the convergence process for finding the minimum of (3.11). The relative gap between these two bounds is a measure for the accuracy of the approximation $\zeta^{(l)}$. However, this method needs extra matrix-vector multiplications of monitoring the decrease of this second quadratic function. This makes this stopping criterion computationally expensive.

Until now we have not discussed how to select a good starting point for our iterative procedures. This starting point $\zeta^{(0)}$ or $Z^{(0)}$ can have a significant influence on the required number of iteration steps. However, there are no general applicable rules for choosing these points if one does not have a priori information. We always assume that the starting point is the zero vector (matrix).

One of the rare cases where we have extra information is in hyperparameter selection procedures like cross-validation [7]. In these cases one often trains the kernel models for a whole grid of (γ, σ) combinations. Since each small change of γ or σ can be regarded as a small disturbance of the systems, one may assume that the solution will not change significantly. According to perturbation theory [30] this change is bounded as follows.

In the case of a regression problem with the RBF kernel changing one of both hyperparameters affects the matrix $H(\gamma, \sigma) = K(\sigma) + \frac{1}{\gamma} I_N$. Both changes can be seen as a perturbation of the linear system. In the case that $\mathbf{t} \neq \mathbf{0}_d$ and $\left\| \Delta \left(K + \frac{1}{\gamma} I \right) \right\|_2 < \frac{1}{\left\| \left(\Delta \left(K + \frac{1}{\gamma} I \right) \right)^{-1} \right\|_2}$ the resulting change in the solution is bounded by [30]:

$$\frac{\|\Delta\zeta\|_2}{\|\zeta\|_2} \leq \frac{\mathrm{cond}_2\left(K+\frac{1}{\gamma}I\right)}{1+\mathrm{cond}_2\left(K+\frac{1}{\gamma}I\right)\frac{\left\|\Delta\left(K+\frac{1}{\gamma}I\right)\right\|_2}{\left\|K+\frac{1}{\gamma}I\right\|_2}}\left(\frac{\left\|\Delta\left(K+\frac{1}{\gamma}I\right)\right\|_2}{\left\|K+\frac{1}{\gamma}I\right\|_2}\right). \quad (3.15)$$

Therefore, during the cross-validation process the solution $(\eta,\nu)_{old}$ of the old hyperparameter $(\gamma,\sigma)_{old}$ setting can be used as good starting points in the iterative processes to compute the solutions $(\eta,\nu)_{new}$ for new hyperparameters settings $(\gamma,\sigma)_{new}$. This process is also called *alpha seeding* [31]. A specific form of alpha seeding for the SMO algorithm applied to LS-SVM models can be found in [70].

3.5.6 SMO for LS-SVM algorithms

SMO or Sequential Minimal Optimization was first introduced by Platt [107] for speeding up the quadratic optimization problem in SVMs. This method makes use of the box constraints of the optimization problem and the resulting sparseness of the solution (see also Section 2.9). Consequently the memory usage for solving the QP problem could be reduced drastically because it only optimizes two components α_i and α_j at one iteration step. This process can be done analytically without needing to solve a QP problem.Hence, no Hessian matrix needs to be stored. Until now it is one of the most memory efficient methods for training an SVM.

Models like LS-SVMs, Regularization Networks, Kriging and Gaussian processes however do not have box constrains nor a sparse solution. Therefore Keerthi [70] uses the Karush-Kuhn-Tucker optimality conditions together with a stopping criterion based on the duality gap between the primal and dual formulation to construct a sequential optimization scheme. He shows that the formulation of the SMO, for models without a bias term, have large similarities with the SOR methods previously described (Section 3.4).

This model can be seen as a valid alternative for the described conjugate gradient algorithms. However, one disadvantage is that its performance is highly influenced by the hyperparameters of the systems. For cases where γ is large and therefore the linear system is 'more' ill-conditioned, the computational complexity increases significantly. According to the tests presented in the work of Keerthi the CG algorithms outperform the SMO algorithm.

3.5.7 Numerical results

Implementation details

All the algorithms were implemented and tested in Matlab 6.0. The computation of the matrix H, which is a bottleneck in all the algorithms, was done in C and the communication via the CMEX interface of Matlab. The tests were done on an AMD Athlon 500 MHz processor with 256 MB RAM running MS Windows NT 4.0. The C-code compilation was done with MS Visual C++ compiler. The computationally intensive tests on the adult data set were done on a dual Pentium III 850 MHz processor and 1Gb RAM running Linux. The C-code there was compiled with gcc. In the implementation we reduced the needed memory as much as possible. For the SSOR method this means that we stored only those rows of H in memory that were needed by the block-SSOR. The CG-algorithm did the matrix-vector multiplication in a row by row manner. At each step the matrix H is recomputed.

The hyperparameter selection for the different kernels is done on the basis of the tests performed and reported in [141]. The classification performance of all the tested methods is therefore equal. The data sets that we used are obtained from the UCI benchmark repository [8]. The starting point $\zeta^{(0)}$ of the iterative procedures (SOR, SSOR, CG, Block-CG,...) was always set equal to $\mathbf{0}_d$. The following data sets are used: the Statlog Australian Credit (acr), the Bupa Liver disorder (bld), the Statlog German credit (gcr), the Statlog heart disease (hea), the John Hopkins university ionosphere (ion), the Pima Indians diabetes (pid), the sonar (snr), the tic-tac-toe endgame (ttt), the Wisconsin breast cancer (wbc) and the adult data set (adu). All their characteristics can be found in Appendix C. N stands for the total number of data points in the data set. This data set is divided in N_{train} training points and N_{test} test points. The dimensions of the input are given by d, which can be divided in d_{num} numerical attributes and d_{cat} categorical ones according to the data sets.

Although all the UCI data sets, except the adult data set, are rather small data sets, they reflect very well the ideas that we want to point out in this chapter. Normally we would advise to train the LS-SVM with a Cholesky factorization on these data sets. Here, however, also the other training procedures are used on these data sets to verify the methods.

Experimental results of the SOR methods

Here we describe the experimental results that we performed on the statlog heart disease data set from the UCI benchmark repository [8]. It is a data set with 270 data points and 13 attributes. We used 180 points to train the LS-SVM and the other 90 points are used as test set. We trained the LS-SVM with both the linear kernel and the RBF kernel. In both occasions we used the optimal hyperparameters γ and σ as stated in Appendix C. We trained the LS-SVM with a training algorithm based on the block-SSOR and the block-SSOR with GM-acceleration and this for different block-sizes. The overrelaxation parameter was set to $\omega = 1.0$. Improvements in convergence might still be made by adapting this parameter. Remember that for each training two sets of linear equations have to be solved. To avoid the fact that the matrix H has to be consulted in two separate algorithms, we implemented the SSOR such that the two linear systems $H\eta= \mathbf{y}$ and $H\nu= \mathbf{1}_N$ are computed simultaneously.

In Section 3.1 we already mentioned that as criterion to compare the algorithms we use the number of iteration steps l before convergence upon a certain tolerance. This parameter best reflects the true computational cost that is needed for the algorithms. This parameter is also not dependent on implementation techniques nor hardware. The number of iteration steps is equal to the number of times that the matrix H has to be consulted. For large applications, where the matrix H cannot be stored in the main memory, every consultation of H corresponds to an I/O operation to external storage devices or a recomputation of the matrix. In the latter each of these iterations has a computational complexity of $\mathcal{O}(N^2)$. Because of the forward and backward loop made by the SSOR, each iteration taken by the algorithm corresponds to two consultations of H and therefore has a computational complexity of $\mathcal{O}(2N^2)$. Therefore the total computational complexity of the SSOR methods before convergence is $\mathcal{O}(2lN^2)$.

In Figure 3.3 we see that the block-size reduces the number of iterations needed for convergence which results in a lower computational complexity. This influence is largest for the classic SSOR method. The disadvantage is that the algorithm will need more memory. Notice that in the hypothetical situation where one increases the size of the data sets, the block-size compared to the dimensions of H needs to decrease because of memory limitations. This unfortunately will increases the number of iterations that will needed.

The SSOR with GM-acceleration is less dependent on the block-size and therefore advisable. The speedup in number of iterations accomplished by

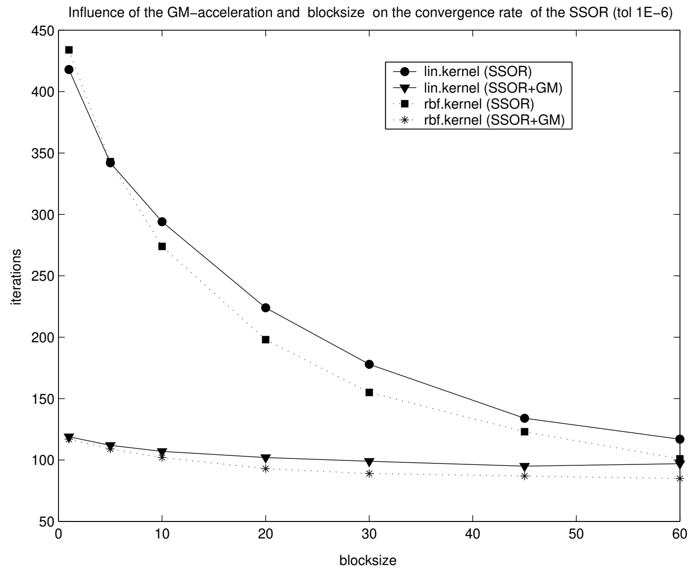

Figure 3.3: The number of iterations that are needed for the training algorithms on the statlog Heart disease data set (hea) with linear and RBF kernel. The training algorithms are based on the Block-SSOR and Block-SSOR algorithm with GM-acceleration. The training is done with variable block sizes. We see that the block-size reduces the number of iterations needed for convergence which results in a lower computational complexity. This influence is largest for the classic SSOR method compared to the SSOR with GM-acceleration.

	SSOR+GM	CG
acr	55	13
bld	21157	8
grc	298	17
hea	169	14
ion	197	20
pid	44339	9
snr	60	19
ttt	16	9
wbc	3183	11

Table 3.1: Comparison of required number of iteration steps l for linear kernel between the conjugate gradient algorithm and the SSOR with GM-acceleration. The CG algorithm always has a lower computational complexity compared to the SSOR on all the tested data sets.

the GM-acceleration is largest for small block-sizes.

The minimum number of iterations needed to converge is 95 in the case of the RBF kernel with a block size of 60. This corresponds to 190 consultations of the H matrix with a block size which is 1/3 of the actual dimensions of the linear system. This is a large number compared to the Krylov methods as we will see in the next section.

Comparison between block-SSOR and CG

To compare the performance of the training routines based on SSOR with GM-acceleration and the one based on the CG, we trained the LS-SVM with both methods on the UCI data sets. We trained the LS-SVM with a linear kernel and the optimal hyperparameters. The overrelaxation parameter was set to $\omega = 1.0$. The block-size was set equal to 1 so that both algorithms needed the same amount of memory. The training algorithms iterated to a tolerance of 10^{-10}. Results of the number of iterations that are needed, are given in Table 3.1.

Since the total computational complexity of the SSOR and CG respectively is $\mathcal{O}(2lN^2)$ and $\mathcal{O}(lN^2)$, we see that the training method based on the CG algorithm always has a lower computational complexity on all the tested data sets. As stated in the previous chapter, better performance for the SSOR can be achieved if we use other overrelaxation parameters. But the automatic choice of this optimal overrelaxation parameter as a function of the data set is an open problem and therefore a serious disadvantage of

	Lin		Pol		RBF	
	CG	BCG	CG	BCG	CG	BCG
arc	13	21	29	22	14	12
bld	8	5	42	19	43	164
gcr	17	47	9	9	66	75
hea	14	26	19	16	11	10
ion	20	39	12	11	33	26
pid	9	74	18	15	16	9
snr	19	21	6	6	20	16
ttt	9	11	35	23	25	36
wbc	11	43	21	15	12	9
adu	11	8	29	22	69	54

Table 3.2: Comparison of needed number of iteration steps l for the linear, polynomial and RBF kernel between conjugate gradient and block conjugate gradient method. The results on the adult data set are based on training with a subset of 10000 data points. We see that block-CG method (BCG) has most of the time a higher convergence rate compared to CG for the non-linear kernels. In case of a linear kernel it is still preferable to use a training algorithm based on CG. The performances on the polynomial kernel show an average decrease of 20 iterations. In the best occasion, the Bupa Liver disorder data set, we can even notice a decrease of 55 iterations. The tests performed on the adult data set show an advantage in number of iterations of BCG compared to CG for every kernel.

these methods.

Comparison between CG and block-CG

In this section we make a comparison of two Krylov methods: the conjugate gradient method (3.5.1) and the block conjugate gradient method (3.5.4). We trained the LS-SVM on the UCI data set. We trained with three different kernels: linear, polynomial and RBF. The needed number of iterations for a tolerance of 10^{-10} are given in Table 3.2. Both algorithms have a similar computational complexity of $\mathcal{O}(lN^2)$ where l is the number of iterations to converge.

In Table 3.2 we show that the block-CG method (BCG) has most of the time a higher convergence rate compared to the CG-method for the non-linear kernels. In case of a linear kernel it is still preferable to use a training algorithm based on CG. The performances on the polynomial kernel show an average decrease of 20% in number of iterations. For, the Bupa Liver disorder data set, we can even notice a decrease of 55%. The tests performed on the adult data set show an advantage in number of iterations of BCG compared to CG for every kernel.

Figure 3.4 and Figure 3.5 show the convergence properties of CG and BCG. The tests were performed on the adult data set with 10000 training points and the RBF kernel. We plotted in Figure 3.4 the evolution of log $J(\zeta)$ for both linear systems individually $H\eta = \mathbf{y}$ and $H\nu = \mathbf{1}_N$ as CG1 and CG2 respectively for the conjugate gradient, BCG1 and BCG2 for the block conjugate gradient. The same is done for the residual of those linear systems in Figure 3.5. These figures clearly show that for the same number of iterations BCG has a smaller residual and a steeper decrease of the $J(\zeta)$, which results in a faster convergence.

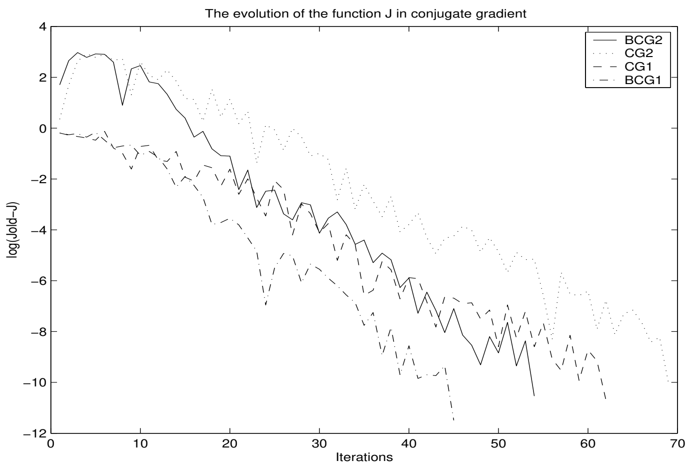

Figure 3.4: Comparison of the convergence properties of the CG and Block-CG tested on the adult-dataset with the RBF-kernel. (We took a subsample of 10000 datapoints of the total dataset.) This Part shows the logarithm of the evolution of the J function used in the CG and Block-CG algorithm with respect to the number of the iterations. We make a distinction between to linear systems that have to be solved for training the LS-SVM. These results clearly show that for the same number of iterations the BCG has a steeper decrease of the $J(\zeta)$, which results in a faster convergence.

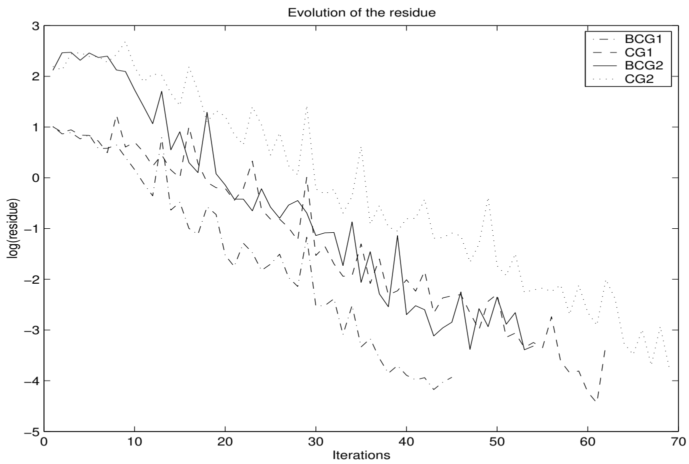

Figure 3.5: Comparison between the convergence properties of CG and Block-CG tested on the adult-dataset with the RBF-kernel. This shows the logarithm of the residual plotted against the number of iterations for all four linear systems. These results show that for the same number of iterations the BCG has a smaller residual.

3.6 Conclusions

In this chapter we have presented different iterative methods for LS-SVM classifiers. The training of the LS-SVM consists of solving a set of linear systems. Because of the symmetric positive definite structure of these linear systems, many efficient numerical algorithms exist. We showed that for large data sets, the iterative methods (SOR, CG,...) have many advantages. A first advantage is that they can be implemented in a memory efficient way. Second, they are in most cases much faster than the direct methods (Cholesky Factorization) for large data sets.

The first iterative method we tested was the successive overrelaxation method. Exploiting the structure in combination with a block implementation of SSOR gave a more memory efficient algorithm. We also showed that the GM-acceleration reduces the required amount of iterations before convergence. The extra advantage is that its dependence on the block size and the overrelaxation parameter is reduced. This GM-acceleration combined with a block-SSOR gives many advantages for large data sets compared with the classical SOR method.

If one compares the block-SSOR (with GM acceleration) with the conjugate gradient (CG) method in terms of computational complexity, the latter outperforms block-SSOR in all cases. A second Krylov method is the block-CG algorithm. This method solves the two linear systems, that are needed to train the LS-SVM, simultaneously. In most tests we performed on the UCI data sets, the block-CG method converges faster for the non-linear classification tasks.

These results make us conclude that for smaller problems $N < 1000$ Cholesky methods are advised. For larger problems, $1000 < N < 5000$, we advise the conjugate gradient and block conjugate gradient methods.

Chapter 4

Kernels for Large Scale Applications

This chapter gives an overview of the different types of kernel functions that are used in the SVM literature. Some general rules for constructing kernels are explained. The kernel functions are classified into three groups: stationary, locally stationary and nonstationary. Attention goes to the computational aspects of these kernels and we will provide general rules of thumb for certain classes of kernels. In this chapter the effects of compactly supported kernels are studied. These kernels can lead to a reduction in computational complexity of the training phase. Correspondingly, they also have an influence on the generalization performance of the kernel model.

4.1 Introduction

In the second chapter we already gave an introduction on kernels and their link with inner products in a feature space. We discussed that the necessary condition for a kernel to represent a mapping in feature space is that it belongs to the class of Mercer kernels. These Mercer kernels have the property to be positive definite. The choice of the kernel offers an enormous freedom in the design of a learning model. It is by applying the *kernel trick* (see Section 2.3) that the kernel model can switch between linear or non-linear models or even between different types of non-linearity. To demonstrate the difference between linear and non-linear models we do the following experiment..

In this example we tested the difference in performance between a linear and non-linear classification. Therefore we trained an LS-SVM classifier model using the LS-SVMlab toolbox [104] on 10 UCI data sets. (For a detailed description of these data sets together with their optimal hyperparameters see Appendix C.) Each data set is divided into a test (1/3) and training (2/3) set. After hyperparameter selection by a cross-validation routine for both the linear and the non-linear Gaussian RBF kernel the generalization performance is measured on the test set. This performance is measured based on the area-under-the-curve (AUC) of the ROC ([136],[135],[35],[80]). This area under the ROC curve should approach 1 for a perfect classification. Using the ROC-routine of the LS-SVMlab, the AUC together with its standard error (SE) are given. Since for both kernels these measures are computed based on the same test set, they are most probably correlated. In [64] a new method was suggested for comparing both ROC measures taking into account the correlation among the two areas. Comparison of the two ROC measures (AUC_1, SE_1) and (AUC_2, SE_2) is done by computing the critical ratio z:

$$z = \frac{AUC_1 - AUC_2}{\sqrt{SE_1^2 + SE_2^2 - 2rSE_1SE_2}}, \tag{4.1}$$

where r represents the estimated correlation between AUC_1 and AUC_2. Assuming the normality of this variable, an evidence of 'true' difference is proven if $z \geq 1.96$. For an exact description of how to compute the correlation coefficient r we advise the methods and tables proposed in [64].

In this experiment we computed the AUC together with its SE for an experimental setup with the linear and Gaussian RBF kernel. The results together with the computed value of r and z are given in Table 4.1

These experiments show that in three (ion, ttt, adu) out of the ten experiments there is a significant improvement in the generalization performance of a kernel model using the non-linear Gaussian RBF kernel in terms of ROC performance. So, it is clear that in some cases a non-linear kernel would be preferred from a learning perspective. In this chapter we will discuss the computational consequences of this choice of the kernel.

In this chapter we will present several classes of kernels based on the work of Genton [52]. We will focus on the computational aspects related choice of the kernel. We first summarize some important properties of kernels.

4.2 General properties of Mercer kernels

The kernel functions used in the support vector literature are symmetric, real-valued functions defined as

$$k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R} : (\mathbf{x}, \mathbf{x}') \mapsto k(\mathbf{x}, \mathbf{x}'), \tag{4.2}$$

data	AUC_{lin}	SE_{lin}	AUC_{RBF}	SE_{RBF}	r	z
acr	0.945	0.015	0.939	0.018	0.70	0.48
bld	0.712	0.048	0.764	0.046	0.66	1.35
gcr	0.813	0.025	0.774	0.028	0.65	1.74
hea	0.919	0.029	0.890	0.034	0.69	1.14
ion	0.914	0.032	0.988	0.074	0.35	2.47
pid	0.819	0.027	0.812	0.028	0.71	0.31
snr	0.844	0.048	0.912	0.036	0.20	1.24
ttt	0.632	0.039	1.000	1e-4	0.18	1.08e1
wbc	0.994	0.004	0.994	0.004	0.82	0.14
adu	0.871	0.003	0.880	0.003	0.57	2.60

Table 4.1: Comparison of the ROC curves for linear and non-linear classification. In this experiment we computed the AUC together with its SE for both an experimental setup with the linear and Gaussian RBF kernel. Comparison of the two ROC measures (AUC1,SE1) and (AUC2,SE2) is done by computing the critical ratio z where r represents the estimated correlation between AUC1 and AUC2. Evidence of 'true' difference is proven if $z = 1.96$. These experiments show that in three (ion, ttt, adu) out of the ten experiments there is a significant improvement in the generalization performance of a kernel model using the non-linear Gaussian RBF kernel in terms of ROC performance.

where $\mathcal{X} \subseteq \mathbb{R}^d$ is the *input space*. In order to represent an inner product in a high dimensional feature space $\mathcal{F}$, these kernel functions need to fulfill the Mercer condition (Theorem 1).

These kernels can be written as $k(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$ where the function $\varphi : \mathcal{X} \subseteq \mathbb{R}^d \to \mathcal{F}$ is a non-linear mapping of the *input space* $\mathcal{X} \subseteq \mathbb{R}^d$ into a $d_{\mathcal{H}}$-dimensional *feature space* $\mathcal{F}$ where $d_{\mathcal{H}} \in \mathbb{N}$ or $d_{\mathcal{H}} = \infty$. A second corollary is that these kernel are positive definite functions. The relation between the positive definiteness and the fact to be a Mercer kernel can also be used in the other direction. The positive definiteness is a necessary and sufficient condition for a symmetric function $k(\mathbf{x}_i, \mathbf{x}_j)$ to be a kernel with a guaranteed existence of a mapping φ [52].

In addition, the use of a positive definite kernel for interpolation will lead to a unique solution of the systems of equations determining the coefficients of the interpolator [95].

Let us first give some general properties of these kernels. A first property of kernels is that they fulfill the Cauchy-Schwarz inequality:

$$(k(\mathbf{x}_i, \mathbf{x}_j))^2 \leq k(\mathbf{x}_i, \mathbf{x}_i)\, k(\mathbf{x}_j, \mathbf{x}_j). \tag{4.3}$$

Let $\mathcal{K}$ be the set of Mercer kernel functions, the following algebraic rules hold :

1. if $k_1, k_2 \in \mathcal{K}$ and $a_1, a_2 \in \mathbb{R}_0^+$ then $a_1 k_1(\mathbf{x}, \mathbf{x}') + a_2 k_2(\mathbf{x}, \mathbf{x}') \in \mathcal{K}$,
2. if $k_1, k_2 \in \mathcal{K}$ then $k_1(\mathbf{x}, \mathbf{x}')\, k_2(\mathbf{x}, \mathbf{x}') \in \mathcal{K}$,
3. if $k_1 \in \mathcal{K}$ then $\exp(k_1(\mathbf{x}, \mathbf{x}')) \in \mathcal{K}$,
4. if $g : \mathbb{R}^d \to \mathbb{R}$ then $k(\mathbf{x}, \mathbf{x}') = g(\mathbf{x})\, g(\mathbf{x}') \in \mathcal{K}$,
5. if $h : \mathbb{R}^d \to \mathbb{R}^d$ and $k \in \mathcal{K}$ then $k(h(\mathbf{x}), h(\mathbf{x}')) \in \mathcal{K}$,
6. if $A \in \mathbb{R}^{d \times d}$ is a positive definite matrix then $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T A \mathbf{x}' \in \mathcal{K}$,
7. if $l : \mathbb{R}^d \to \mathbb{R}^+$ with minimum at 0 then
$$k(\mathbf{x}, \mathbf{x}') = \frac{1}{4}\left[l(\mathbf{x} + \mathbf{x}') - l(\mathbf{x} - \mathbf{x}')\right] \in \mathcal{K}.$$

These rules give the opportunity to design a specific kernel for a specific task. The first two rules define that $\mathcal{K}$ is a convex cone in which every polynomial combination of kernels exist. This property is very interesting if we want to combine information from different data sources and want to avoid cross-correlation among the different data types. For each data type a specific kernel can be applied. Afterwards these kernels can be combined in a polynomial way. An example is given in the article of Pavlidis *et al.* [102] where information from DNA microarray measurements and phylogenetic profiles for gene functional classification is combined. Other examples of specific designed kernels are the *string subsequence kernels* [28] which are examples of rule 4. These kernels are particularly interesting for sequences and string data.

Next, we study some particular classes of kernels proposed by Genton [52].

4.3 Stationary kernels

The first class are the *Stationary Kernels.* This class consists of translation invariant kernels :

$$k\left(\mathbf{x}, \mathbf{x}'\right) = k_S\left(\mathbf{x} - \mathbf{x}'\right). \tag{4.4}$$

These kernels do not depend on the individual input vectors $\mathbf{x}$ and $\mathbf{x}'$ but only on their difference $\mathbf{x} - \mathbf{x}'$. This class can be divided in the *Anisotropic Stationary* kernels that depend on both the direction and length of $(\mathbf{x} - \mathbf{x}')$. Kernels that only depend on the length and not the direction of $(\mathbf{x} - \mathbf{x}')$ are called *Isotropic Stationary* and satisfy

$$k\left(\mathbf{x}, \mathbf{x}'\right) = k_I\left(\left\|\mathbf{x} - \mathbf{x}'\right\|_2\right). \tag{4.5}$$

Some of the most popular non-linear kernels belong to this class. Examples are:

- Gaussian Radial Basis Function: $k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x}-\mathbf{x}'\|_2^2}{\sigma^2}\right)$ where $\sigma \in \mathbb{R}_0^+$,
- Exponential Radial Basis Function: $k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x}-\mathbf{x}'\|_2}{\sigma}\right)$ where $\sigma \in \mathbb{R}_0^+$,
- Multiquadric Radial Basis Function: $k(\mathbf{x}, \mathbf{x}') = \left(\sigma^2 + \|\mathbf{x} - \mathbf{x}'\|_2^2\right)^{\frac{u}{2}}$ where $u \in 2\mathbb{Z}{+}1$ and $\sigma \in \mathbb{R}_0^+$,
- Spline kernel: $k(\mathbf{x}, \mathbf{x}') = \prod_{p=1}^{d} B_{2n+1}\left(x_p - x_p'\right)$, where B_n is a B-spline, $n \in \mathbb{N}$ and x_p and x_p' are the p-th components of the d-dimensional vector $\mathbf{x}$ and $\mathbf{x}'$,

- Matérn kernel: $k(\mathbf{x}, \mathbf{x}') = \frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{2\sqrt{\nu}\|\mathbf{x}-\mathbf{x}'\|_2}{\sigma}\right)^{\nu} H_\nu \left(\frac{2\sqrt{\nu}\|\mathbf{x}-\mathbf{x}'\|_2}{\sigma}\right)$ where $\sigma \in \mathbb{R}_0^+$, Γ is the gamma-function and H_ν is the modified Bessel function of the second kind of order ν. Note that the exponential and the Gaussian kernel are special cases of this Matérn type kernel for respectively $\nu = 0.5$ and $\nu \to \infty$.

One feature that these kernels have in common is their infinite support. This causes the matrix to be dense.

Most of the *isotropic stationary* kernels presented above have a bell-shape. This means that if the points $\mathbf{x}$ and $\mathbf{x}'$ are far form each other the corresponding kernel evaluations will become very small or almost zero. We will study the effect of making this contribution identically zero. This leads us to kernels with compact support.

4.3.1 Compactly supported isotropic stationary kernels

Compactly supported kernels evaluate to zero when the distance $\|\mathbf{x} - \mathbf{x}'\|_2$ is larger than a certain cut-off distance. This leads to a sparse kernel or Gram matrix. This leads to a better memory complexity and can be exploited computationally in the training process. Compactly supported kernels are not new in the statistical literature [56]. We will focus on two types of compactly supported kernels: the spline kernel and the compactly supported radial basis function.

The spline kernel is very well known in interpolation and function estimation. It was first introduced for SVM regression by Vapnik [148], who focussed mainly on kernels based on the B-spline or basic spline [140]. These splines are the shortest possible polynomial splines. Moreover, they have the advantage to have a local support, which can result in more zeros in the kernel matrix. However, as is most often the case for spline kernels, they are only defined for the 1 dimensional case. The *B-splines* are defined in $\mathbb{R}$ as, $\forall x \in \mathbb{R}$:

$$B_n(x) = \sum_{r=0}^{n+1} \frac{(-1)^r}{n!} \frac{(n+1)!}{r!(n+1-r)!} \left(x + \frac{n+1}{2} - r\right)_+^n, \tag{4.6}$$

where $(x)_+ = \max(0, x)$ and the order $n \in \mathbb{N}$.

The kernel function, as used within the SVM theory, based on this B-spline is given by:

$$k(x, x') = B_{2n+1}(x - x'). \tag{4.7}$$

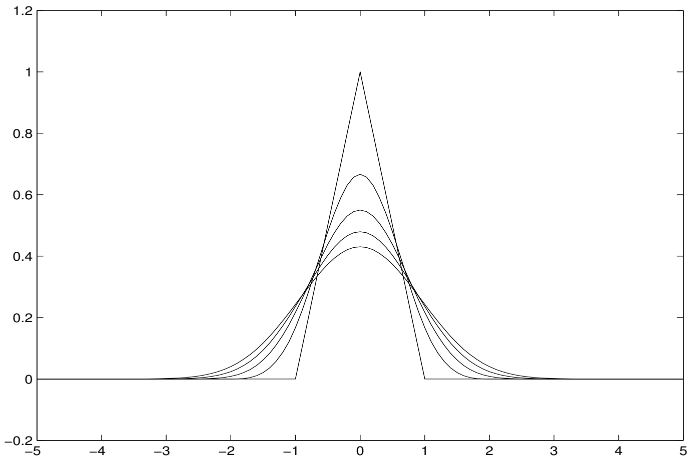

Figure 4.1: The B-spline kernel for different values of n.

Note that only splines of odd order are used. These only are guaranteed to be positive definite. The d-dimensional generalization of these kernels is given by, $\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$:

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \prod_{p=1}^{d} B_{2n+1}\left(x_p - x'_p\right), \tag{4.8}$$

where x_p and x'_p are the p-th components of the d-dimensional vector $\mathbf{x}$ and $\mathbf{x}'$. The shape of the kernel for different values of order n can be seen in Figure 4.1.

The range and shape of the kernel is controlled by the hyperparamter n. The fact that this hyperparameters is discrete, is an advantages for the hyperparamter tunning. If one compares the search space for the crossvalidation routine for the discrete (spline) and continuous case (RBF) one can see the following. The search space in the discrete (spline) case is smaller than in case of a continuous hyperparameter and therefor will be easier to solve.

Although these spline kernels have the compact support that we wanted, they have some important disadvantages:

- the definition of this kernel is not symmetric. One has to apply some tricks to make it symmetric.
- For large n it suffers from numerical and computational problems.

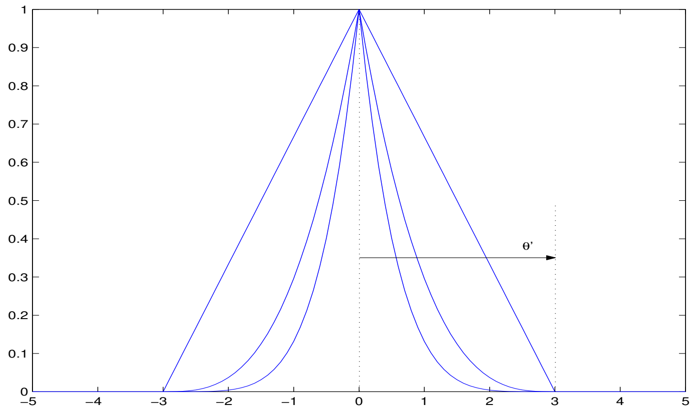

Figure 4.2: The kernel k_c with a cut-off distance of 3 for different values of n (1,3,5).

- It is computationally slow for large dimensional input data d.
- One can not control the cut-off distance since it depends strictly on the order n.

Also in [119] it was mentioned that although this kernel may have computational advantages, it potentially reduces generalization performance.

The second type of compact kernel is based on the *Matérn kernel*. A very important property of Matérn kernels is that they can easily be transformed into compactly supported kernels [52]. This means that the kernel evaluation will be zero if $\|\mathbf{x} - \mathbf{x}'\|_2$ is larger than a cut-off distance θ'. This locality can be constructed by multiplying the kernel by another kernel k_c:

$$k_c(\mathbf{x}, \mathbf{x}') = \max \left\{ 0, \left(1 - \frac{\|\mathbf{x} - \mathbf{x}'\|_2}{\theta'} \right)^{\nu'} \right\}, \tag{4.9}$$

where $\theta' \in \mathbb{R}_0^+$ and $\nu' \geq \frac{(d+1)}{2}$ to ensure positive definiteness. Note that one should take $\nu' \in 2\mathbb{N} + 1$ to have compact support. This kernel is shown in Figure 4.2. The product of both kernels is still a kernel, which guarantees the positive definiteness.

4.3.2 Exploiting the sparse kernel matrix for the LS-SVM

The sparseness that can be achieved by localizing the Matérn kernels will lead to a sparse H matrix. In this section we will explain how this sparseness

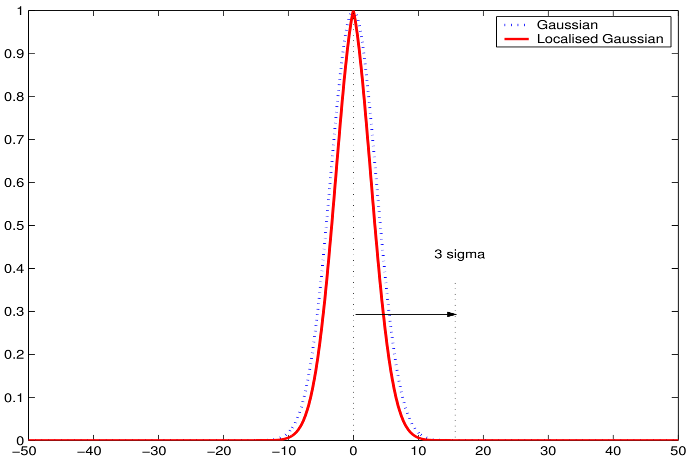

Figure 4.3: The Gaussian RBF kernel with and without compact support. The cut-off distance is $\theta' = 3\sigma$.

can be exploited in the optimization algorithms. Since there is a fundamental difference in the two models, we will handle them separately.

In both cases we will use the *Gaussian RBF kernel with compact support*

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \max\left\{0, \left(1 - \frac{\|\mathbf{x} - \mathbf{x}'\|_2}{3\sigma}\right)^{\nu'}\right\} \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}'\|_2^2}{\sigma^2}\right). \tag{4.10}$$

To avoid too much extra parameters we decided to set $\theta' = 3\sigma$, where σ is the bandwidth of the Gaussian RBF kernel. ν' is chosen equal to the dimension of the input variables for the odd cases. If the dimension is even, we augment it by one. In Figure 4.3 we can see that there is not much difference between the two kernels.

As a result of the compactly supported RBF kernel one gets a sparse kernel matrix and therefore also a sparse matrix H. The memory requirements become proportional to the number of non-zero elements n_z. The computational cost is also reduced by making efficient use of the zero elements in the matrix. This can be computationally exploited when one solves the training process. For direct methods there exist different permutation algorithms (column count permutation, symmetric minimum degree, reverse Cuthill-McKee,...)[54] on the elements of the sparse matrix H that give a higher degree of sparseness in the Cholesky factor G. This obviously will give computational advantages for the forward and backward substitution.

In the case of Krylov methods, the most demanding part in is the matrix-vector product between H and the conjugate directions. This can also be reduced by a compactly supported RBF kernel. The number n_z can be exploited at this point. In further tests we will show that the condition number $\mathrm{cond}_2(H)$, which determines the convergence rate of the algorithm, is almost not affected by the compactly supported kernel. For more information about the use of sparse kernels and their computational advantages we advise [54].

Regression: examples

We compare compactly supported RBF and RBF kernels for a noisy sinc function estimated by LS-SVMs. The sinc function or sine cardinal or sample function is defined as $f^*(x) = \frac{\sin x}{x}$. The tuning parameters are selected by 10-fold cross-validation. The inputs were take between -20 and 20 with an interspacing of 0.03. We added Gaussian noise to the inputs with zero mean and standard deviation 0.1. Figure 4.4 shows that the performance of regression with the RBF and compactly supported RBF are almost the same. A compactly supported RBF gives a slightly larger true bias $f - f^*$ and less smooth results. The pointwise variance of $f(x)$ is larger for the Compactly supported RBF kernel. An advantage of the compactly supported RBF kernel is the sparse kernel matrix. In the example of the sinc function with 1334 training points, the number of non-zero elements decreases from $1334^2 = 1779556$ to $n_z = 850160$. In this one-dimensional problem the Gram matrix also has a very clear band structure as can be seen in Figure 4.5. Note that the H matrix is independent of the y values of the training set. This means that for each regression problem with the same $\mathbf{x}$ values for the training set and hyperparameter set (γ, σ) the H matrix has this sparse band structure. The time needed to solve the two systems is the following: the Cholesky factorization needs 9.75 seconds cpu-time to solve the two linear systems for the Gaussian RBF and 4.33 seconds for the compactly supported RBF. The conjugate gradient method needs respectively 4.15 seconds and 2.65 seconds. Hence, we observe that the locally supported kernel results in a memory reduction and a speed-up of about 50 %.

We also tested the influence of the compact support on the condition number of the matrix H. Figure 4.6 shows that there is only a small difference in the condition number of the matrix H for the different values of (γ, σ). Therefore, the speed of convergence for CG with the RBF and the compactly supported RBF kernel is comparable.

As a second example we used the Boston housing data set. This data

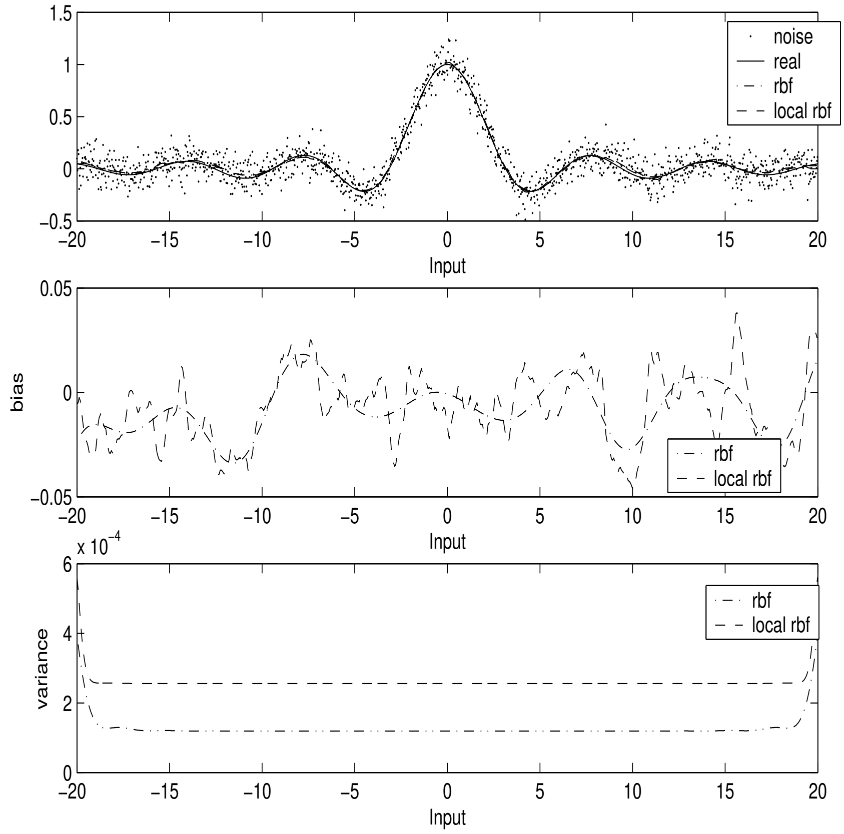

Figure 4.4: LS-SVM results for non-linear function estimation on the sinc function. The middle and bottom part show respectively the true bias $f - f^*$ and the variance of both estimates for the RBF and the compactly supported RBF kernels. A compactly supported RBF gives a slightly larger true bias and less smooth results. The pointwise variance of $f(x)$ is larger for the compactly supported RBF kernel. This shows that the performance of regression with the RBF and compactly supported RBF are almost the same but the algorithm with compactly supported RBF kernel only needs 50% of the memory resources compared to the original RBF kernel.

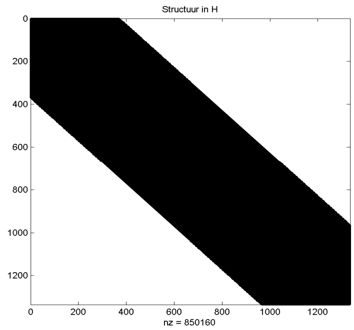

Figure 4.5: Structure of the H-matrix in the sinc regression problem with a compact supported kernel and 1334 training points. This Figure is the result of the *spy*-function in matlab. This produces a template view of the sparsity structure, where each point on the graph represents the location of a nonzero array element. The number of non-zero elements decreases from $1334^2 = 1779556$ to $n_z = 850160$ with is a decrease of the memory complexity of nearly 50%.

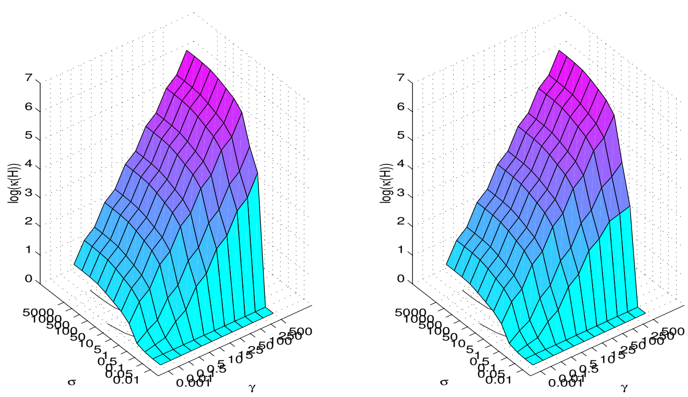

Figure 4.6: This figure shows the logarithm of the condition number for different hyperparameters (γ, σ). (Left) TBF kernel; (Right) compactly supported RBF kernel. Note that the condition number is not significantly larger for the compactly supported RBF. Therefore, the speed of convergence for CG with the RBF and the compactly supported RBF kernel is comparable.

	$\sigma = 1.5$	$\sigma = 2.0$	$\sigma = 10$
MSE_{tr}	5.8e-3	1.3e-2	1.20-1
MSE_{test}	1.1e-1	1.0e-1	8.45e-2
n_z/N^2	0.37	0.84	1

Table 4.2: The performance for different values of bandwidth for the compactly supported RBF kernel. mse_{tr} and mse_{test} are respectively the mean squared error on the training and test set. The ratio n_z/N^2 characterizes the degree of sparseness in the Gram matrix as result of the compact support. This shows that improving the sparseness of the kernel matrix by changing the should be done with care because it also influences the generalization performance of the model.

set consists out of 506 cases in 14 attributes. We trained the LS-SVM on 406 random sampled points and used the other 100 points as test set. We normalized the data except the binary variables. In Table 4.2 we show the performance of the LS-SVM for different values of the hyperparameter σ, where $\gamma = 30$ is kept constant. We computed the mean squared error (MSE) on training and test set for these values of σ. The performances for the normal RBF and compactly supported RBF kernel were comparable on all tests. We see that by decreasing σ, the Gram matrix will become more sparse as a result of the localization of the kernels. This is indicated by the ratio of non-zeros n_z to the number of element in the Gram matrix in Table 4.2. Notice that there is a difference in MSE on training (MSE_{tr}) and test set (MSE_{test}). If we decrease σ the MSE on the training set decreases and the MSE on test set increases. This might be an indication of overfitting. This shows that improving the sparseness of the kernel matrix by changing the σ should be done with care because it also influences the generalization performance of the model. Both should be tuned simultaneously. This is typically done by a cross-validation routine.

Classification: examples

As a second test we use the compactly supported kernels on the UCI classification data tasks. However, the performance on these were not convincing with respect to the introduced sparseness. This can be explained intuitively as follows. The Gram matrix will become sparse if the ratio of the hyperparameter σ to the variance of the distribution of the data points is small. Test on the UCI data sets (Section 4.1) showed that good performance can be achieved by a linear kernel. This indicates that these UCI data sets are 'linear' separable. Consequently, the optimal σ will not be much smaller than the variance of the distribution of the data points. As a result of that the kernel matrix will be less sparse.

Hence, our hypothesis is that only on very non-linear classification data sets, like the Chess board benchmarking data set, an improvement can be achieved with a compactly supported kernel.

Time-series prediction: examples

In a third example we use LS-SVM for time-series prediction on the Santa Fe laser data set. These data were recorded from a Far-Infrared-Laser in a chaotic state. The training set are 1000 measurements over time (see Figure 4.7(a)) and the goal is to predict the next 100 steps. The model that is being used is the *Nonlinear Auto-Regressive with eXogenous inputs scheme* or *NARX [79].* This is defined as:

$$\hat{y}_{i+1} = f(y_i, y_{i-1,\ldots}, y_{i-q}), \tag{4.11}$$

In Figure 4.7(b) we see that a good performance is obtained for the RBF kernel with hyperparameters $(\gamma, \sigma) = (70, 4)$ found by 10-fold cross-validation. For the same hyperparameters the compactly supported RBF kernel has a very bad performance as can be seen in 4.7(b). For almost similar performance either the cut-off point has to be increased $\theta' = 50\sigma$ or the bandwidth of the compactly supported RBF kernel has to increase (Figure 4.7(c) and (d)). Unfortunately both measures reduce the degree of sparseness in the Gram matrix to zero. Remarkably this indicates that the tails of the Gaussian RBF kernel are important and 'cutting' them off should be done with care.

This result is a side-effect of the *curse of dimensionality.* To understand this we should alter our intuition of high dimensional geometrics. In a high dimensional space the majority of the fraction of the volume of a hypersphere is contained in the outer shell of this sphere. Therefore, points that are randomly distributed inside a sphere in a d dimensional space, where d is large, are almost all concentrated in a thin shell close to the surface (see also [7]). The same results hold for a geometrical region described by a RBF kernel. Therefore the tails of an RBF kernel (or other isotropic kernel) play an important role in high dimensions. Cutting off the tails will drastically reshape the kernel with possible decreasing performance of the learning model.

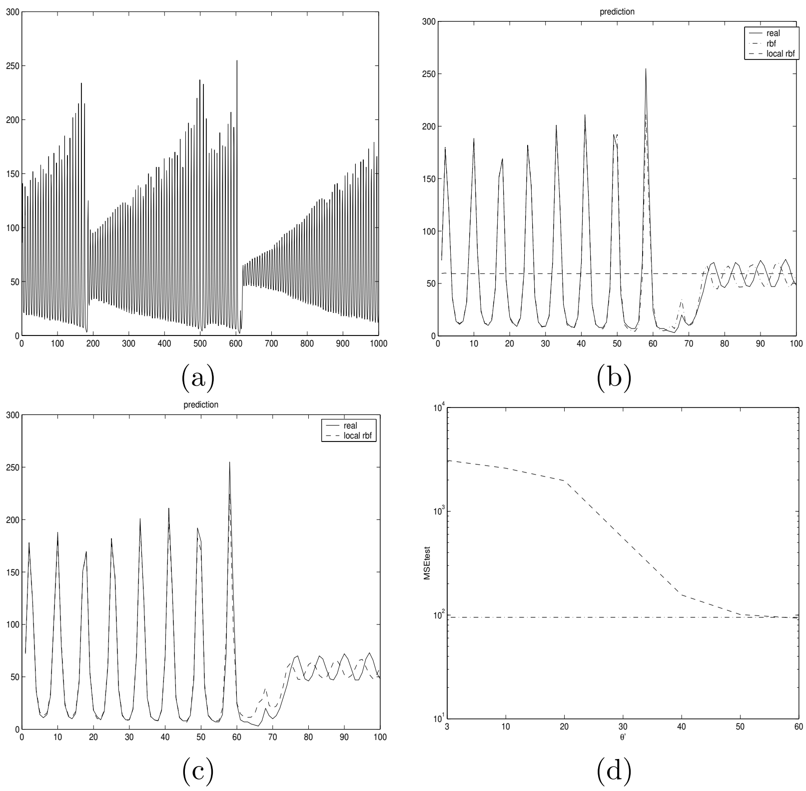

Figure 4.7: The Santa Fe laser data time-series prediction. The given data are indicated by a full line. The subfigures show: (a) input data (b) prediction with Gaussian RBF (.-) and compactly supported Gaussian RBF (- -) $(\gamma, \sigma) = (70, 4)$, (c) prediction with compactly supported RBF (- -) $(\theta' = 50\sigma)$ Gaussian RBF (γ, σ)=(70,4), (d) gives the mse measured on the test set for a prediction based on the compactly supported Gaussian RBF(- -) and this with respect to the cut-off points θ'. This shows a bad mse for smaller θ' that correspond to a sparse Gram matrix. The horizontal (.-) is the mse on the test set for the Gaussian RBF with optimal hyperparameters. Introducing sparseness in the kernel matrix increases the mse performance which indicates a bad generalization performance.

This example brings us to the conclusion that compactly supported kernels can be very useful for reducing the computational bottleneck but should be used with care from a learning perspective, especially in examples with a high dimensional input space and time-series prediction. Therefore, as a rule of thumb we advise to use the compact supported kernel only for low dimensional problems with $d < 5$.

4.4 Locally stationary kernels

A more general class of kernels are the *Locally Stationary kernels.* They are defined as follows:

$$k\left(\mathbf{x}, \mathbf{x}'\right) = k_1\left(\frac{\mathbf{x} + \mathbf{x}'}{2}\right) k_2\left(\mathbf{x} - \mathbf{x}'\right), \tag{4.12}$$

where $k_1 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_0^+$ and k_2 is locally stationary. It should be noticed that stationary kernels are stationary.

4.5 Nonstationary kernels

The most general class of kernels are the *Nonstationary kernels.* These depend on both $\mathbf{x}$ and $\mathbf{x}'$:

$$k(\mathbf{x}, \mathbf{x}').$$

Some examples that belong to this most general class are:

- linear kernel: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$,
- cosine metric kernel: $k(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x}^T\mathbf{x}'}{\|\mathbf{x}\|_2\|\mathbf{x}'\|_2}$,
- polynomial kernels: $k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^T\mathbf{x}'\right)^q$, $k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^T\mathbf{x}' + \mathbf{1}\right)^q$ or $k(\mathbf{x}, \mathbf{x}') = \left(\frac{\mathbf{x}^T\mathbf{x}'}{\|\mathbf{x}\|_2\|\mathbf{x}'\|_2} + \mathbf{1}\right)^q$.

A special and interesting subclass are the *separable nonstationary kernels.* These kernels are defined as

$$k\left(\mathbf{x}, \mathbf{x}'\right) = k_1\left(\mathbf{x}\right) k_2\left(\mathbf{x}'\right) \tag{4.13}$$

where k_1 and k_2 are stationary kernels. These kernels have the very interesting property that the Kernel matrix K over a data set $\{\mathbf{x}_i\}_{i=1}^N$ can be written as $K = \mathbf{a}\mathbf{b}^T$ where $a = [k_1(\mathbf{x}_1)\ k_1(\mathbf{x}_2) \ldots k_1(\mathbf{x}_N)]^T$ and $b =$

$[k_2(\mathbf{x}_1)\ k_2(\mathbf{x}_2)\dots k_2(\mathbf{x}_N)]^T$. This means that the kernel matrix constructed on such a separable kernel always has a rank equal to one. The memory complexity of this kernel matrix reduces from $\mathcal{O}\left(\frac{N^2}{2}\right)$ to only $\mathcal{O}\left(2N\right)$. This feature leads to a very easy training procedure. For regression we explored that one has to solve two linear systems $H\eta = \mathbf{1}$ and $H\xi = \mathbf{y}$. However, since

$$H = K + \frac{1}{\gamma}I = \mathbf{a}\mathbf{b}^T + \frac{1}{\gamma}I, \tag{4.14}$$

one can show by the Sherman-Morrison-Woodbury (see Appendix A.1) formula that

$$H^{-1} = \gamma\left(I - \frac{1}{\frac{1}{\gamma} + \mathbf{b}^T\mathbf{a}}\mathbf{a}\mathbf{b}^T\right), \tag{4.15}$$

which leads to

$$\begin{aligned} H\eta = \mathbf{1} &\Rightarrow \eta = \gamma\left(\mathbf{1} + \tfrac{1}{1+\mathbf{b}^T\mathbf{a}}\mathbf{a}\mathbf{b}^T\mathbf{1}\right), \\ H\xi = \mathbf{y} &\Rightarrow \xi = \gamma\left(\mathbf{y} + \tfrac{1}{1+\mathbf{b}^T\mathbf{a}}\mathbf{a}\mathbf{b}^T\mathbf{y}\right). \end{aligned} \tag{4.16}$$

The complexity and the memory storage needed with these kernels reduces enormously. The computational complexity reduces to $\mathcal{O}(N^2)$ and the memory complexity to $\mathcal{O}(2N)$ for training the kernel model. Similar relations can be show for the classification case. However, we do not know any examples of separable kernels.

A similar strategy as for separable kernels holds for the linear kernel. If one constructs a $N \times d$ matrix X where all the d dimensional input vectors of the data set $\mathcal{D}$ are the rows of this matrix, then the kernel matrix over this matrix is:

$$K = XX^T. \tag{4.17}$$

The rank of the matrix K is equal to the rank of X. In the assumption that no data points are linearly dependent this rank equals d. In most cases there will be more data points N than input dimension d, which means that this matrix is rank deficient. By construction this matrix has an efficient factorization. This factorization can be used in a similar way as the rank one decomposition of the separable kernel. Using the Sherman-Morrison-Woodbury the inversion of the matrix H can be done as follows: $\left(K + \frac{1}{\gamma}I_N\right)[\eta \quad \nu] = [\mathbf{z} \quad \mathbf{u}]$

$$[\eta \quad \nu] = \gamma\left(I_p - X\left(\frac{1}{\gamma}I_p + X^TX\right)^{-1}X^T\right)[\mathbf{z} \quad \mathbf{u}]. \tag{4.18}$$

therefore one first solves the linear system

$$\left(\frac{1}{\gamma}I_p + X^T X\right)[\eta_s \quad \nu_s] = G^T[\mathbf{z} \quad \mathbf{u}], \tag{4.19}$$

from which the solution can be computed as $[\eta \quad \nu] = \gamma\left([\mathbf{z} \quad \mathbf{u}] - X[\eta_s \quad \nu_s]\right)$. One can see that the linear system transforms into a smaller linear system of dimension equal to the input dimension of the learning problem d. Notice now, however, that the computational bottleneck is no longer situated in the matrix inversion of this smaller linear system. The computational complexity is determined by the matrix multiplication of the two factors $X^T X$, which results in $\mathcal{O}(d^2 N)$. So, by using a linear kernel one can reduce the computational complexity involved in training the kernel models from $\mathcal{O}(N^3)$, in the case of the Cholesky factorization, into $\mathcal{O}(d^2 N)$. In the examples that we showed in the introduction the performance of the linear kernel is sufficient in many classification problems. Therefore this method can often be applied. This approach was intensively tested by Chua [24] who pointed out that one can train an LS-SVM model on a million data points in less then a minute using an IBM SP2 computer. Similar approaches for kernel models with inequality constraints (like the SVM) are discussed in the work of Mangasarian [83].

4.6 Conclusions

In this chapter we have studied the role of the kernel function. In an introductory example we showed that on the UCI classification data sets, the choice of the kernel plays an important role in the generalization performance of the model. In some cases the use of a nonlinear kernel has a clear advantage over a linear kernel. To study the computational consequences of using the different types of kernels, we made a classification of the different types of kernels. We discussed the Stationary, Locally stationary and the Non-Stationary kernels based on the work of [52]. For each type we discussed the possible computational or memory advantages that can be achieved by exploiting its properties.

We paid much attention to the use of compactly supported kernels. We showed that the popular RBF kernels can be efficiently transformed into a kernel with compact support. The use of compactly supported kernels can decrease the computational cost and memory requirements. In our study we have seen that for certain problems the generalization performance is comparable as well as the conditioning of the matrices towards iterative methods as the conjugate gradient method. However, on a problem of chaotic time-series prediction the compactly supported RBF kernels fails to produce good results when having a sparse Gram matrix. As a result one may conclude that compactly supported RBF kernels may be useful for some specific applications but one should be careful to use it in a general context. These results are consistent with the work of Smola [124]. He also describes that the computational advantage of the limited support has to be traded for a *possibly* worse performance. Therefore, as a rule of thumb we advise to use the compactly supported kernel only for low dimensional problems with $d < 5$.

In a last part we did discuss some efficient procedures for training LS-SVM models with non-stationary separable or linear kernels.We showed that the intrinsic definition of these kernels results in rank deficient kernel matrix. This rank deficiency leads to good factorizations of the kernel matrix that can reduce the computational and memory complexity of the algorithms.

In the next chapter we will see how this methodology can also be applied to other types of kernels in terms of low rank approximations.

Chapter 5

Low Rank Approximations

The computation time for training kernel models scales quadratically with the number of data points. This bottleneck manifests itself in the training procedure where one needs to solve a set of linear system (3.4) of dimension N. Correspondingly, we advised to use these methods for data sets up to $N < 5000$. In this chapter we will see different approximation methods, on the models and matrix level, that allow us to train on larger data set by circumventing the memory bottleneck. The proposed models are applicable in the range of $5000 < N < 50000$.

5.1 Introduction

When the nonstationary kernel is linear or separable the coefficient matrix of the linear systems can be factorized very easily. This factorization makes it possible to reduce the large linear system by making use of the Sherman-Morrison-Woodbury formula. The question that remains is if this is only limited to this special class of kernels? In Chapter 2 it was already mentioned that the matrix H, and also the kernel matrix K, has an eigenvalue spectrum that decays very rapidly especially in the case of an RBF kernel. Similar behavior was noticed in [152]. Hence the fact that the convergence of the iterative Krylov methods is relatively fast. This might also indicate that the matrix H is (almost) rank deficient. In this case a good factorization might exist such that one can apply the Sherman-Morrison-Woodbury formula to reduce the size of the linear system that has to be solved. Different factorizations will be studied in this chapter.

In the cases of a rank deficient kernel matrix the solution of the linear systems can be constructed in the subspace of the space spanned by the columns (or rows) of the matrix H. In terms of constructing a model in a feature space $\mathcal{F}$, like in our kernel modelling, this rank deficiency is often translated into an interpretation where one also tries to construct models in a subspace of the feature space $\mathcal{F}$ ([126],[152]). Relations of these methods towards kernel PCA, kernel PLS and kernel CCA where shown in ([119],[130],[129],[67]). Some examples of this will also be given in this chapter.

These methods will be explained in a regression setup. All the presented ideas can be applied to classification problems as well.

5.2 The Nyström method

5.2.1 Eigenfunction and eigenvalue approximations

In Chapter 2 we have explained that we want to approximate the function f^*. We choose a model in primal space of the form

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^T \varphi(\mathbf{x}) + b \\ &= \sum_{i=1}^{d_{\mathcal{H}}} w_i \varphi_i(\mathbf{x}) + b, \end{aligned} \tag{5.1}$$

where $d_{\mathcal{H}} \leq \infty$ is the dimension of the high dimensional *feature space* $\mathcal{F}$ and $\varphi = [\varphi_1 \ldots \varphi_{d_{\mathcal{H}}}]$ is the mapping function corresponding to the kernel $k(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x})^T \varphi(\mathbf{x}') = \sum_{i=1}^{d_{\mathcal{H}}} \varphi_i(\mathbf{x}) \varphi_i(\mathbf{x}')$ where $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. Following the Mercer theorem (see Section 2.2) we have seen that

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \sum_{i=1}^{d_{\mathcal{H}}} \lambda_i \phi_i(\mathbf{x}) \phi_i\left(\mathbf{x}'\right). \tag{5.2}$$

Hereby $\lambda_1 \geq \lambda_2 \geq \ldots > 0$ and $\phi_1, \phi_2, \ldots$ are the eigenvalues and eigenfunctions of the integral operator induced by the kernel,

$$\int_{\mathcal{X}} k\left(\mathbf{x}, \mathbf{x}'\right) \phi_i(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} = \lambda_i \phi_i\left(\mathbf{x}'\right), \tag{5.3}$$

with $p(\mathbf{x})$ the probability density[1] of the input points $\mathbf{x}$. Notice that these eigenfunctions are orthonormal to each other with respect to $p(\mathbf{x})$, i.e. $\int \phi_i(\mathbf{x})\,\phi_j(\mathbf{x})\,p(\mathbf{x})\,d\mathbf{x} = \delta_{ij}$. From this equation it follows that we have an expression for the mapping: $\varphi_i = \sqrt{\lambda_i}\phi_i$. But since $d_{\mathcal{H}}$ is possibly infinite, it is impossible to solve this problem in the primal space. To overcome this problem, the Dual Formulation or Wolfe Dual is solved as explained in Chapter 2. However, a second approach to overcome this infinite dimensional model formulation is to make a lower dimensional linear model, $p \ll d_{\mathcal{H}}$, by using a finite dimensional approximation $\tilde{f}$ of the form

$$\tilde{f}(\mathbf{x}) = \sum_{i=1}^{p} w_i \varphi_i(\mathbf{x}) + b. \tag{5.4}$$

It is proven [152] that the best performance is achieved upon the measure $\int (f(\mathbf{x}) - \tilde{g}(\mathbf{x}))^2\, p(\mathbf{x})\,d\mathbf{x}$ when one chooses $\varphi_i = \sqrt{\lambda_i}\phi_i$ corresponding to the p largest eigenvalues λ_i. This is however impossible to solve in an analytic way. Therefore we will have to switch to numerical approximations.

The standard numerical approximation follows from the approximation of the integral operator by applying the simple quadrature rule [153],

$$\int_{\mathcal{X}} k(\mathbf{x}, \mathbf{x}')\,\phi_i(\mathbf{x})\,p(\mathbf{x})\,d\mathbf{x} \simeq \frac{1}{n}\sum_{h=1}^{n} k(\mathbf{x}_h, \mathbf{x}')\,\phi_i(\mathbf{x}_h), \tag{5.5}$$

where $\mathbf{x}_h$ is sampled according to $p(\mathbf{x})$. For finding the eigenfunctions and eigenvalues we again plug in the sample $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ in $\mathbf{x}'$,

$$\frac{1}{n}\sum_{h=1}^{n} k(\mathbf{x}_h, \mathbf{x}_j)\,\phi_i(\mathbf{x}_h) \simeq \lambda_i \phi_i(\mathbf{x}_j), \forall \mathbf{x}_j. \tag{5.6}$$

If we compare this with the eigenvalue problem of the Kernel matrix constructed on this sample $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$,

$$\sum_{h=1}^{n} k(\mathbf{x}_h, \mathbf{x}_j)\,u_{hj}^{(n)} = \lambda_i^{(n)} u_{ij}^{(n)}, \tag{5.7}$$

[1] Notice that in this formulation we use the integral equations with another measure $p(\mathbf{x})\,d\mathbf{x}$. In this case the Mercer theorem (Section 2.2) still holds as long as one stays in $L^2(\mathcal{X})$. This is statisfied for all the kernels we are considering. How this new measure influences the eigenfunctions and eigenvalues of this integral operator and the corresponding kernel model is studied in the work of Williams and Seeger [152].

or in matrix formulation $K_{1:n,1:n}U^{(n)} = \Lambda^{(n)}U^{(n)}$, we can see the following relations:

$$\lambda_i \simeq \frac{1}{n}\lambda_i^{(n)}, \tag{5.8}$$

$$\phi_i(\mathbf{x}_j) \simeq \sqrt{n}u_{ji}^{(n)}. \tag{5.9}$$

Substituting this into equation (5.6) gives the *Nyström approximation* [98] for the eigenfunctions ϕ_i,

$$\phi_i(\mathbf{x}) \simeq \frac{\sqrt{n}}{\lambda_i^{(n)}}\sum_{h=1}^{n} k(\mathbf{x}_h, \mathbf{x})\, u_{hi}^{(n)}. \tag{5.10}$$

Notice that in this formulation we still need to solve an eigenvalue problem of size n. Until know we have not specified which sample will be used to make this approximation. The most straightforward way is of course to use the input data of the data set $\mathcal{D}$. This input data $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ are sampled according to the distribution $p(\mathbf{x})$ and with size $n = N$. This is hard for large N since the computational complexity and memory complexity of these methods are respectively $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$. Therefore a second simplification needs to be made. The approximations of the eigenvalues and eigenfunctions as given in (5.10) and (5.8) do not depend on the size n. Therefore a similar approximation can be made on any random sample of points $\{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_m\}$ with $p \leq m < N$. In this approximation we will assume that this smaller sample is a subset of the original sample of size $N : \{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_m\} \subset \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. According to the previous explanation one can now also make the eigenfunction approximation based on this smaller subsample

$$\begin{cases} \lambda_i \simeq \frac{1}{m}\lambda_i^{(m)}, \\ \phi_i(\mathbf{x}) \simeq \frac{\sqrt{m}}{\lambda_i^{(m)}} \sum\limits_{h=1}^{m} k(\mathbf{x}'_h, \mathbf{x})\, u_{hi}^{(m)}, \end{cases} \tag{5.11}$$

where $\lambda_i^{(m)}$and $\mathbf{u}_i^{(m)}$ are the solutions of the eigenvalue problem $K_{1:m,1:m}U^{(m)} = \Lambda^{(m)}U^{(m)}$. Without loss of generality we may assume that this approximation is based on the first m rows of K. Using Matlab notation for the subscripts, one can write the symmetric kernel matrix as:

$$K = \begin{bmatrix} K_{1:m,1:m} & K_{1:m,N-m:N} \\ K_{N-m:N,1:m} & K_{N-m:N,N-m:N} \end{bmatrix}. \tag{5.12}$$

Both results (5.8),(5.10) and (5.11) are an approximation to the same eigenfunctions and eigenvalues of the kernel integral operator on samples of different sizes.

A side aspect following from both relations is that the eigenvectors $\mathbf{u}_i^{(N)}$ and the eigenvalues $\lambda_i^{(N)}$ of the kernel matrix can be approximated respectively by $\tilde{\mathbf{u}}_i^{(N)}$ and $\tilde{\lambda}_i^{(N)}$ where:

$$\begin{cases} \tilde{\lambda}_i^{(N)} = \frac{N}{m}\lambda_i^{(m)}, \\ \tilde{\mathbf{u}}_i^{(N)} = \sqrt{\frac{N}{m}}\frac{1}{\lambda_i^{(m)}}K_{1:N,1:m}\mathbf{u}_i^{(m)}. \end{cases} \tag{5.13}$$

It is this relation that we will use further to design a new factorization of K.

5.2.2 Factorizations based on the Nyström approximation

For training either Regularization Networks (2.42) or LS-SVM models (3.4) one needs to solve respectively one or two linear systems with the coefficient matrix $H = (K + \frac{1}{\gamma}I)$ (for regression). We will study methods for reducing these linear systems. These methods are all based on the Sherman-Morrison-Woodbury formula (Appendix A.1). One can successfully apply this formula for each factorization of the kernel matrix K. We will present several methods.

The Nyström method was first presented in [153]. The idea is to make a low rank approximation of the matrix K. This approximation is based on the eigenvector decomposition of K. Using the decomposition one knows that $K = U\Lambda U^T$. However, computing this eigenvalue decomposition for large N is in general computationally too expensive. Therefore we will use the approximations, which resulted out of the Nyström approximation. Taking the p largest eigenvalues, this results in the approximation $\tilde{K} = \sum_{i=1}^{p}\tilde{\lambda}_i^{(n)}\tilde{\mathbf{u}}_i^{(n)}\tilde{\mathbf{u}}_i^{(n)T}$. In (5.13) we have demonstrated that the eigenvectors and eigenvalues can be computed based on a subsample of size m. If we fill in these approximations for $p = m$ it is easy to see that

$$\begin{aligned} \tilde{K} &= K_{1:N,1:m}K_{1:m,1:m}^{-1}K_{1:m,1:N} \qquad (5.14) \\ &= \begin{bmatrix} K_{1:m,1:m} & K_{1:m,N-m:N} \\ K_{N-m:N,1:m} & K_{N-m:N,1:m}K_{1:m,1:m}^{-1}K_{1:m,N-m:N} \end{bmatrix}. \end{aligned}$$

Notice that since the matrix K is symmetric $K_{N-m:N,1:m}^T = K_{1:m,N-m:N}$, the difference between the approximation and the real kernel is given by

$$\Delta K_{N-m:N,N-m:N} = K_{N-m:N,N-m:N} - K_{N-m:N,1:m} K_{1:m,1:m}^{-1} K_{1:m,N-m:N}, \tag{5.15}$$

which is the Schur complement of $K_{1:m,1:m}$. According to [153] the computational complexity of this method is $\mathcal{O}(m^2 N)$. This approximation is exact if the true rank of K is m and m linearly independent columns are chosen. This approximation gives us a very cheap method to compute a factorization of the original matrix K.

Since our goal was to compute the solution of a linear system $\left(K + \frac{1}{\gamma} I_N\right) [\eta \quad \nu] = [\mathbf{z} \quad \mathbf{u}]$, this factorization is very useful. If we use the Sherman-Morrison-Woodbury formula and change the notation so that $A = K_{1:N,1:m}, B = K_{1:m,1:m}$, we get the following double linear system for a regression algorithm $H[\eta \quad \nu] = \left(K + \frac{1}{\gamma} I_N\right) [\eta \quad \nu] = [\mathbf{z} \quad \mathbf{u}]$, then

$$\begin{aligned} [\eta \quad \nu] &= \gamma \left(I_m - AB^{-1} \left(\frac{1}{\gamma} BB^{-1} + A^T AB^{-1} \right)^{-1} A^T \right) [\mathbf{z} \quad \mathbf{u}] \qquad (5.16) \\ &= \gamma \left(I_m - AB^{-1} B \left(\frac{1}{\gamma} B + A^T A \right)^{-1} A^T \right) [\mathbf{z} \quad \mathbf{u}] \\ &= \gamma \left(I_m - A \left(\frac{1}{\gamma} B + A^T A \right)^{-1} A^T \right) [\mathbf{z} \quad \mathbf{u}]. \end{aligned}$$

(As shown in the previous chapters, a similar formulation can be found for classification by changing $A = \mathrm{diag}(\mathbf{y}) K_{1:N,1:m}$, where $\mathbf{y}$ is a column vector with the y-labels of the training points). This solution can numerically best be solved by first solving the double linear system:

$$\left(\frac{1}{\gamma} B + A^T A \right) [\eta_s \quad \nu_s] = A^T [\mathbf{z} \quad \mathbf{u}], \tag{5.17}$$

from which $[\eta \quad \nu] = \gamma \left([z \quad d] - A[\eta_s \quad \nu_s] \right)$. This formulation gives a more clearer view on the complexity of this algorithm. The memory complexity is $\mathcal{O}(mN)$, corresponding to the size of the matrix A. Computing the solution of this smaller linear system has a computational complexity of $\mathcal{O}\left(m^3 + m^2 N\right)$. The matrix inversion involves $\mathcal{O}\left(m^3\right)$ but the major computational bottleneck is the multiplication of the two matrices $A^T A$. Since $N > m$ the memory complexity can be summarized as $\mathcal{O}\left(m^2 N\right)$, which still can be significantly smaller then the $\mathcal{O}(l N^2)$, where l is the number of iteration steps before convergence, which are needed for solving the large linear system with Krylov methods.

We will focus now on solving this smaller double linear system of dimension m. Notice that this matrix formulation is a generalization of the regularized Least Squares problem in its *normal equation* form [10]. However, there might be some numerical drawbacks. It is not guaranteed that the coefficient matrix $\left(\frac{1}{\gamma}B + A^T A\right)$ in this smaller linear system is positive definite. It is possible that the matrix is positive semi-definite, which means that the matrix is singular and the linear system has no unique solution.

This can be understood by analyzing the two terms of the matrix. The sum of $\left(\frac{1}{\gamma}B + A^T A\right)$ is positive definite only if one of them is positive definite. The first term B, being a submatrix of the original Gram matrix K, inherits its properties. In the most general case it is positive semi-definite because of the fact that K has this property. Only in the case that the Gram matrix is constructed on a sample $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^d$ and the Gaussian RBF kernel, it can be proven that this matrix has full rank [119].

For the second term it holds that $A^T A$ is positive definite in the case that A (which is equal to $K_{1:N,1:m}$) has full column rank. But, because of the limited machine precision it still becomes positive semi-definite. It is known in the literature that computing this term $A^T A$ explicitly may cause loss of significant digits towards results in a matrix that is far from positive definite. It might even become singular. This can be seen by studying the condition number $\mathrm{cond}_p(A^T A) = (\mathrm{cond}_p(A))^2$. Hence, in a worst case scenario, we need to assume that $\left(\frac{1}{\gamma}B + A^T A\right)$ is positive semi-definite. How can we now overcome this problem?

The best suited numerical methods for solving positive semi-definite linear system are based on LU factorization with symmetric pivoting. This is a direct method with a computational complexity of $\mathcal{O}(m^3)$. Iterative methods based on the Lanczos process in most case have a lower computational complexity of $\mathcal{O}(lm^2)$ with l the number of iteration steps needed. For symmetric possible indefinite systems the Minimum Residual (MinRes) and Symmetric LQ (SymmLQ) are advised [2]. According to [48] the most computationally expensive step in all these iterative methods is the matrix-vector multiplication, similar as in the CG algorithm (Chapter 3). Therefore we will compare the algorithms on the basis of the number of iterations they need before convergence.

To check the difference between both iterative methods we have tested them in 10 classification tasks. For each of the UCI (Appendix C) classifica-

	N	SymmLQ	MinRes
acr	460	21/42	20/34
bld	230	13/40	10/37
gcr	666	28/31	21/30
hea	180	6/6	6/6
ion	234	25/31	24/26
pid	512	38/50	30/40
snr	138	7/8	7/8
ttt	638	30/65	18/54
wbc	455	7/19	5/13
adult	2000	55/141	37/72

Table 5.1: Number of iteration steps before convergence needed by the SymmLQ and the MinRes procedure in ten classification tasks. Since two linear systems have to be solved for in the training of the Nyström factorization procedure, two numbers are indicated in each column. This shows that in all performed classification tasks the MinRes procedure needed less iteration steps before convergence compared to the SymmLQ. Each iteration has a computational complexity of $\mathcal{O}(m^2)$. So the number of iteration steps l gives an idea about the total computational complexity of these methods: $\mathcal{O}(lm^2)$. Based on the results the MinRes procedure is more advisable then the SymmLQ.

tion tasks we used a non-linear Gaussian RBF kernel with the corresponding optimal hyperparameters for best generalization performance found for 10 fold cross-validation. We used the Nyström factorization based on a 10 % sample ($m \simeq \frac{N}{10}$) as stated above and solved the resulting m-dimensional linear system with SymmLQ and MinRes. The number of iteration steps l needed before convergence for both linear systems is counted. The results are summarized in Table 5.1.

Table 5.1 shows that in all performed classification tasks the MinRes procedure needed less iteration steps before convergence compared to the SymmLQ. According to [48] both methods need in each iteration step a matrix-vector multiplication with a computational complexity of $\mathcal{O}(m^2)$. So the number of iteration steps l gives an idea about the total computational complexity of these methods: $\mathcal{O}(lm^2)$. Based on the results the MinRes procedure is more advisable than the SymmLQ.

There are also several ways to avoid this indefinite matrix. We will discuss some of the possibilities: A first and simple method is by adding an extra jitter factor. By adding a small term εI the whole matrix $\left(\frac{1}{\gamma}B + A^T A + \varepsilon I\right)$ becomes positive definite. This is a simple and well-known way of solving the problem. The only drawback is that the solution of this linear system will be different from the original. The difference will depend on the weighting factor ε and the condition number of $\frac{1}{\gamma}B + A^T A$. (See also Appendix B)

A second way to solve these numerical problems is by using an SVD of $\tilde{K} = \tilde{U}\Lambda\tilde{U}^T$ ([153],[151]). As described in [49] the SVD of $\tilde{K}$ can be computed based on the SVD of the submatrix $K_{1:m,1:m}$ of K. If $K_{1:m,1:m} = U\Lambda U^T$ then $\tilde{U}$ is given as

$$\tilde{U} = \begin{bmatrix} U \\ C^T U \Lambda^{-1} \end{bmatrix}, \tag{5.18}$$

with $C = K_{1:m,N-m:N}$. A drawback is however that the approximated eigenvectors of $\tilde{U}$ are not necessarily orthogonal (solutions for this are formulated in [49]). Making use of this factorization on $\tilde{K}$ the solution of $\left(K + \frac{1}{\gamma}I_N\right)[\eta \quad \nu] = [\mathbf{z} \quad \mathbf{u}]$ can be found by again applying the Sherman-Morrison-Woodbury formula. This results into

$$[\eta \quad \nu] = \gamma\left(I_m - \tilde{U}\left(\frac{1}{\gamma}I + \Lambda\tilde{U}^T\tilde{U}\right)^{-1}\Lambda\tilde{U}^T\right)[\mathbf{z} \quad \mathbf{u}]. \tag{5.19}$$

This can be computed by solving the smaller linear systems

$$\left(\frac{1}{\gamma}I + \Lambda\tilde{U}^T\tilde{U}\right)[\eta_s \quad \nu_s] = \Lambda\tilde{U}^T[\mathbf{z} \quad \mathbf{u}] \tag{5.20}$$

or

$$\left(\frac{1}{\gamma}I + \Lambda + U^T C C^T U \Lambda^{-1}\right)[\eta_s \quad \nu_s] = [\Lambda U \quad U^T C][\mathbf{z} \quad \mathbf{u}], \tag{5.21}$$

out of which follows that $[\eta \quad \nu] = \gamma\left([\mathbf{z} \quad \mathbf{u}] - \tilde{U}[\eta_s \quad \nu_s]\right)$. The computational complexity of this method formulated as in (5.21) is $\mathcal{O}(m^2(N-m))$ corresponding to the matrix multiplication CC^T but this method requires an extra SVD. This linear system now is strictly positive definite. This then gives us the opportunity to use numerical methods as Cholesky Factorization and Conjugate Gradient.

A third method to overcome this problem is to ensure that the subsample, from the Nyström approximation, is well chosen in the following terms.

Until now the selection of the subsample $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$, corresponding to a selection of columns/rows of K, was done randomly. One way is to choose the sample such that the columns of the matrix B are as orthogonal as possible. However, finding an optimal subset of m out of N is a combinatorial problem since there exist $\frac{N!}{(N-m)!m!}$ combinations. Different heuristics of how to sample the m columns/rows out of K are given by Smola and Schölkopf in [126]. This last method introduces a set of other algorithms to reduce the original problem formulation. These use different strategies to span basis vectors in the feature space. However, since these methods are computationally expensive and their advantages from learning perspective are rather small, we advise to take the sample randomly. We will come back to this in Section 5.4.

In the case of a random selection procedure of the sample of size m and under the assumption of using a direct method to solve the smaller linear system, the computational complexity of the total Nyström factorization method becomes $\mathcal{O}(m^2N + m^3)$.

5.3 Cholesky factorization

A standard numerical method for factorizing the kernel matrix is a Cholesky Factorization. An unique Cholesky factorization is normally only guaranteed for positive definite matrix. But Golub and Van Loan [57] also describe that one can also make Cholesky factorization of a positive semi-definite matrix. As a result of such a factorization one gets a Cholesky factor $G \in \mathbb{R}^{N \times p}$ such that $K = GG^T$. Here, $p \leq N$ is an approximation of the true rank of K.

The Cholesky factorization of the positive semi-definite kernel matrix is based on the *outer product Cholesky factorization* [57]. This uses the property that if R is a positive definite matrix then the following relation holds:

$$
\begin{aligned}
R &= \begin{bmatrix} a & v^T \\ v & T \end{bmatrix} \\
&= \begin{bmatrix} \sqrt{a} & 0 \\ \frac{v}{\sqrt{a}} & I_{N-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & T - \frac{v^T v}{a} \end{bmatrix} \begin{bmatrix} \sqrt{a} & \frac{v^T}{\sqrt{a}} \\ 0 & I_{N-1} \end{bmatrix},
\end{aligned} \tag{5.22}
$$

where $a > 0$ and $T - \frac{v^T v}{a}$ is positive definite. If the Cholesky factor of

$T - \frac{v^T v}{a}$ is G_1, the Cholesky factor of the matrix R is given by

$$G = \begin{bmatrix} \sqrt{a} & 0 \\ \frac{v}{\sqrt{a}} & G_1 \end{bmatrix}. \tag{5.23}$$

This gives the opportunity to compute the Cholesky factorization of a matrix in a recursive way.

The fact that K, the matrix to which we apply this factorization, is positive semi-definite needs some special attention. The recursive updating of the Cholesky factorization only takes place as long as the diagonal elements of the intermediate matrix are positive. To avoid numerical instabilities, caused by small diagonal elements during the updating process, Golub and Van Loan suggest also a symmetric pivoting scheme of the rows and columns during the process of the factorization. These two aspects are formulated in an algorithm presented in [57], which computes the Cholesky factorization of a positive semi-definite matrix such that $PKP^T = \tilde{G}\tilde{G}^T$ where P is a permutation matrix. The true Cholesky factor of K then can be found via $G = P^T\tilde{G}$, which is still of dimension N by p. As a result this method gives a low-rank approximation of the kernel matrix, which is better than just taking a subsample of an arbitrary size m as in the Nyström method. Intuitively one could interpret the monitoring of the diagonal elements as an alternative method for choosing an optimal basis or subspace. The disadvantage is that the factorization might end up with a higher computational complexity of $\mathcal{O}(Np^2)$ if the true rank of the matrix is almost N.

Once this factorization is made one can again use Sherman-Morrison-Woodbury formula to exploit the structure of $\left(K + \frac{1}{\gamma} I_N\right)[\eta \quad \nu] = [\mathbf{z} \quad \mathbf{u}]$:

$$[\eta \quad \nu] = \gamma \left(I_p - G \left(\frac{1}{\gamma} I_p + G^T G \right)^{-1} G^T \right) [\mathbf{z} \quad \mathbf{u}]. \tag{5.24}$$

To avoid numerical instabilities one first solves the linear system

$$\left(\frac{1}{\gamma} I_p + G^T G \right) [\eta_s \quad \nu_s] = G^T [\mathbf{z} \quad \mathbf{u}] \tag{5.25}$$

from which the solution again can be computed by $[\eta \quad \nu] = \gamma \left([\mathbf{z} \quad \mathbf{u}] - G[\eta_s \quad \nu_s] \right)$. One can see that the linear system transforms into a smaller linear system of dimension p. The computational complexity is also here determined by the matrix multiplication of the two factors $G^T G$, which results in $\mathcal{O}(p^2 N)$. Compared to the Nyström method, the linear system however is a positive definite system for which excellent methods exist, like Cholesky factorization or conjugate gradient.

This method was already successfully applied to the positive semi-definite kernel matrix K by Scheinberg and Fine [47] for SVMs. They developed an algorithm to compute the factorization with a computational complexity $\mathcal{O}(p^2N)$ and a memory complexity of $\mathcal{O}(pN)$.

If one uses a direct method to solve the smaller linear system, the computational complexity of the total method becomes $\mathcal{O}(p^2N + p^3)$. A disadvantage of this method is that one can not directly control the size of p.

5.4 Constructing a basis in feature space

In order to reduce the number of parameters of our model one can also try to choose a smaller number of basis vectors in feature space. Remember (Section 2.6.2) that in order to represent a model $f(\mathbf{x}) = \mathbf{w}^T\varphi(\mathbf{x}) + b$ in Primal or feature space $\mathbf{w}$ is expanded over an N dimensional basis $\{\varphi(\mathbf{x}_i)\}_{i=1}^N$:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \varphi(\mathbf{x}_i), \tag{5.26}$$

where both $\mathbf{w}$ and each of the basis vectors $\varphi(\mathbf{x}_i)$ have dimension $d_{\mathcal{H}}$. But the number of free-parameters α_i is equal to N. Instead of using all basis vectors one can also take a smaller set of size $m \ll N$ based on a subset of the training points $\{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_m\} \subset \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. Since $\mathbf{w}$ is then approximated by $\tilde{\mathbf{w}}$:

$$\tilde{\mathbf{w}} = \sum_{i=1}^{m} \tilde{\alpha}_i \varphi\left(\mathbf{x}'_i\right), \tag{5.27}$$

the approximation of the model f is

$$\begin{aligned}
\tilde{f}(\mathbf{x}) &= \tilde{\mathbf{w}}^T\varphi(\mathbf{x}) + b \\
&= \sum_{i=1}^{m} \tilde{\alpha}_i \varphi\left(\mathbf{x}'_i\right)^T \varphi(\mathbf{x}) + b \\
&= \sum_{i=1}^{m} \tilde{\alpha}_i k\left(\mathbf{x}'_i, \mathbf{x}\right) + b.
\end{aligned} \tag{5.28}$$

The parameters $\tilde{\alpha}_i$ of the model are now reduced to m instead of N, hence this model can be seen as a *sparse representation* of the original. To find the optimal parameters the following cost function is minimized in

Primal Space [20]:

$$\begin{cases} \min\limits_{\mathbf{w},\mathbf{e},b} & \frac{1}{2}\|\tilde{\mathbf{w}}\|_2^2 + \frac{\gamma}{2}\|\mathbf{e}\|_2^2 \\ \text{s.t. } y_i = \tilde{\mathbf{w}}^T\varphi(\mathbf{x}_i) + b + e_i, \ \forall i \in \{1,\ldots,N\}. \end{cases} \tag{5.29}$$

By filling in the expression for $\tilde{\mathbf{w}}$ and eliminating both $\tilde{\mathbf{w}}$ and $\mathbf{e}$ one gets:

$$\min_{\tilde{\alpha},b} \quad \frac{1}{2}\sum_{i,j=1}^{m}\tilde{\alpha}_i\tilde{\alpha}_j k\left(\mathbf{x}'_i,\mathbf{x}'_j\right) + \frac{\gamma}{2}\sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{m}\tilde{\alpha}_j k\left(\mathbf{x}'_j,\mathbf{x}_i\right) + b\right)^2. \tag{5.30}$$

This equation can be simplified into a matrix equation by using the notation introduced in the previous section, $A = K_{1:N,1:m}, B = K_{1:m,1:m}$:

$$\min_{\tilde{\alpha},b} \quad \frac{1}{2}\tilde{\alpha}^T B\tilde{\alpha} + \frac{\gamma}{2}\left(\mathbf{y} - A\tilde{\alpha} - b1_N\right)^2. \tag{5.31}$$

The solution of this minimization problem can be found by setting the partial derivatives with respect to $\tilde{\alpha}$ and b equal to zero. The resulting relations can be summarized into the $(m+1)\times(m+1)$ linear system:

$$\begin{bmatrix} \frac{1}{2\gamma}B + \frac{1}{2}A^TA & A^T1_N \\ 1_N^TA & 1 \end{bmatrix}\begin{bmatrix} \tilde{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} A^T\mathbf{y} \\ 1_N^T\mathbf{y} \end{bmatrix}. \tag{5.32}$$

Following the same reasoning as in the previous section we may conclude that $\frac{1}{2\gamma}B + \frac{1}{2}A^TA$ is in general positive semi-definite and therefore the whole coefficient matrix of (5.32) inherits this property. Note that the largest computational part in this procedure is the computation of A^TA which has a computational complexity $\mathcal{O}(Nm^2)$ We already mentioned that this linear system can best be solved with LU factorization as a direct method with a computational complexity of $\mathcal{O}((m+1)^3)$ or MinRes and SymmLQ as iterative solvers with an computational complexity of $\mathcal{O}(l(m+1)^2)$. So, the total computational complexity for computing the matrix multiplication and solving the linear system in this methods is $\mathcal{O}(Nm^2 + (m+1)^3)$.

The question remains how to choose the subset $\{\mathbf{x}'_1,\mathbf{x}'_2,\ldots,\mathbf{x}'_m\}$ on which the approximation of $\tilde{\mathbf{w}}$ is based. Each choice of subset $\{\mathbf{x}'_1,\mathbf{x}'_2,\ldots,\mathbf{x}'_m\} \subset \{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N\}$ corresponds a subspace span $\{\varphi(\mathbf{x}'_1),\varphi(\mathbf{x}'_2),\ldots,\varphi(\mathbf{x}'_m)\} \subseteq \mathcal{F}$. Similar as in the Nyström approximation one can take a random subset of the training points corresponding to a subspace spanned by a random subset of $\{\varphi(\mathbf{x}_1),\varphi(\mathbf{x}_2),\ldots,\varphi(\mathbf{x}_N)\}$. However, in several recent articles ([126], [3], [19]) different methodologies have been suggested that use different heuristics to form an optimal subspace of which the individual basis vectors are orthogonalized in feature space. This 'rank reduction' can often be computed more efficiently that the optimal row/column subspace of the kernel matrix K since the inner product between two vectors in feature space can be simplified by making use of the properties of the kernels: $\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle_{\mathcal{H}} = k(\mathbf{x}_i, \mathbf{x}_j)$.

In [129] another suggestion was made by Suykens *et al.* for sampling the subset $\{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_m\}$. This method makes use of an unsupervised step to make a density estimation of the sample based on a Parzen Window density estimation. Here one proposes to choose a sample that maximizes the *Quadratic Renyi Entropy,* which is defined as

$$H_R = -\log \int p(\mathbf{x})^2 \, d\mathbf{x}. \tag{5.33}$$

It was shown in [55] that this *Quadratic Renyi Entropy* over a sample $\{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_m\}$ can be computed by

$$H_R\left(\{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_m\}\right) = -\log \frac{1}{m^2} \sum_{i,j=1}^{m} k_p(\mathbf{x}'_i, \mathbf{x}'_j), \tag{5.34}$$

where $k_p(\cdot, \cdot)$ is the normalized kernel on which the Parzen Window density estimation is based. Choosing a sample of size m out of the whole data set $\mathcal{D}$ that maximizes this entropy will lead to a more uniform spread of the data sample over the input space X. Similar as in the *Radial Basis Networks* literature this subset of data points will be regarded as *prototype* vectors that will be used to build the model on. Notice that this unsupervised sampling step is separate form the training of the kernel model. Therefore one may also use extra information, like test points, to choose the optimal subsample. This leads to a transductive learning methodology ([51],[147],[21],[69]).

Lin *et al.* suggests in [78] a further simplification of the cost function (5.32). Based on the idea of the Reduced Support Vector Machines [76] they proposed modifying the regularization term into $\frac{1}{2}\sum_{i,j=1}^{m} \tilde{\alpha}_i \tilde{\alpha}_j$ instead of $\frac{1}{2}\sum_{i,j=1}^{m} \tilde{\alpha}_i \tilde{\alpha}_j k\left(\mathbf{x}'_i, \mathbf{x}'_j\right)$. The cost function hereby becomes[2]

$$\min_{\tilde{\alpha}, b} \quad \frac{1}{2}\sum_{i,j=1}^{m} \tilde{\alpha}_i \tilde{\alpha}_j + \frac{\gamma}{2}\sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{m} \tilde{\alpha}_j k\left(\mathbf{x}'_j, \mathbf{x}_i\right) + b\right)^2. \tag{5.35}$$

[2] In the original formulation of Lin et al. [78] also the bias term b is regularized. Although this leads to a simpler optimization problem, it is not advisable from theoretical perspective [110].

As was already remarked by the authors this modification of the LS-SVM formulation does lead to a similar formulation as is known from *Radial Basis Networks* literature [7]. Also here many different strategies exist for finding the optimal subset of data points such as *orthogonal least squares* ([22]), which are related to the orthogonalization procedures previously mentioned. Also many unsupervised learning methods, based on density estimation techniques, are often used to choose a good set of radial basis functions. (Examples are clustering algorithms like K-means and Gaussian Mixture models by using Expectation-Maximization.)

5.5 Fixed size LS-SVM

In the previous section we have seen how one can reduce the dimensionality of the optimization process involved in training by reducing the number of dual parameters α. A second way to achieve a lower dimensional problem formulation is to reduce dimensionality of the problem formulation in primal space [129]. In order to exploit this lower dimensional problem formulation one needs to find the model formulation in primal space $f(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x})$. Therefore one needs to the explicit formulation of the mapping function $\varphi(\mathbf{x})$. In Chapter 2 we have seen that this mapping can be expressed in by the eigenfunctions and eigenvalues of the integral operator (2.19): $\varphi_i(\mathbf{x}) = \sqrt{\lambda_i}\phi_i(\mathbf{x}), \forall i = 1 \ldots d_{\mathcal{H}}$. These eigenfunctions and eigenvalues can be constructed based on a finite sample $\{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_m\} \subset \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ of size $m \leq N$ by making use of the Nyström approximation (5.11). This results the following m-dimensional approximation of the mapping $\varphi_i(\mathbf{x}) \simeq \tilde{\varphi}_i(\mathbf{x})$ where

$$\begin{aligned} \tilde{\varphi}_i(\mathbf{x}) &= \sqrt{\lambda_i^{(m)}}\phi_i^{(m)}(\mathbf{x}) \\ &= \frac{1}{\sqrt{\lambda_i^{(m)}}}\sum_{k=1}^{m} k\left(\mathbf{x}'_k, \mathbf{x}\right) u_{ki}^{(m)}. \end{aligned} \tag{5.36}$$

The resulting model that will be used is an m-dimensional linear model $\tilde{f}(\mathbf{x}) = \sum_{i=1}^{m} w_i \tilde{\varphi}_i(\mathbf{x})$ in primal space. The model takes the form

$$\tilde{f}(\mathbf{x}) = \sum_{i=1}^{m} w_i \frac{1}{\sqrt{\lambda_i^{(m)}}}\sum_{k=1}^{m} k\left(\mathbf{x}'_k, \mathbf{x}\right) u_{ki}^{(m)}. \tag{5.37}$$

Since one now has a finite dimensional expression of the model in Primal Space the parameters or unknowns w_i can be estimated directly. The estimation of the parameters in this model is done similar to the previous given formulation of the LS-SVM. Since we are able to find a meaningful estimate of the φ_i we can now find the parameters by solving the *ridge regression* problem in $\mathbf{w}$ as a parametric model,

$$\min_{\mathbf{w}\in\mathbb{R}^m, b\in\mathbb{R}} \quad \frac{1}{2}\sum_{i=1}^{m} w_i^2 + \frac{\gamma}{2}\sum_{j=1}^{N}\left(y_j - \left(\sum_{i=1}^{m} w_i \tilde{\varphi}_i\left(\mathbf{x}_j\right)\right)\right)^2. \tag{5.38}$$

therefore we fill in the elements of the training set $\mathcal{D}$ in the model (5.37) which leads to the overdetermined linear system

$$\begin{aligned}\mathbf{y} &= \left(AU\mathrm{diag}([\frac{1}{\sqrt{\lambda_1^{(m)}}}\cdots\frac{1}{\sqrt{\lambda_m^{(m)}}}])\right)\mathbf{w} \\ &= Z\mathbf{w},\end{aligned} \tag{5.39}$$

where similar to the previous sections $A = K_{1:N,1:m}$ and the eigenvectors $U^{(m)}$ and eigenvalues $\lambda_1^{(m)}$ follow from the eigenvalue decomposition of $K_{1:m,1:m}U^{(m)} = \Lambda^{(m)}U^{(m)}$. We already discussed previously that this involves operations with a computational complexity of $\mathcal{O}(m^3)$. The optimal parameter $\mathbf{w}$ corresponding to the ridge regression cost function (5.38) can be computed by

$$\mathbf{w} = \left(Z^T Z + \frac{1}{\gamma}I_m\right)^{-1} Z^T\mathbf{y}, \tag{5.40}$$

by making use of direct methods. But again the computationally most intensive step is the matrix multiplication $Z^T Z$ with a computational complexity of $\mathcal{O}(Nm^2)$. So, if we sum the major computational steps one gets a computational complexity of $\mathcal{O}(Nm^2 + m^3 + m^3)$. However, in practise this is rarely doen because of numerical instabilities. A more efficient way, with the same computational complexity, is by computing a QR-factorization of the matrix:

$$\begin{bmatrix} Z \\ \frac{1}{\sqrt{\gamma}}I_m \end{bmatrix}[\mathbf{w}] = \begin{bmatrix}\mathbf{y} \\ \mathbf{0}\end{bmatrix}. \tag{5.41}$$

A second aspect of the Fixed Size LS-SVM is that the points on which the approximations are made, are not randomly chosen. In [129] it was proposed to chose a sample that maximizes the *Quadratic Renyi Entropy* (see previous section). This leads to a sparse formulation of the model based on prototype vectors. For more details and examples on classification, regression and time-series problems see ([39],[38],[129]).

5.6 Model evaluation

Until now we have discussed only the training time of the kernel models. Another aspect is the time needed to evaluate new points once the training is done. In Chapter 1 we have seen that the original SVM formulation as introduced by Vapnik has a sparse solution by nature of its training process. This means that many of the components of the vector α are zero. We saw that with each α_i corresponds an element of $\mathcal{D}$. Those training points that have a non-zero α_i are called the support vectors. Let us assume that their are $n_s \ll N$ support vectors, then the evaluation of a new point by the kernel model needs n_s kernel evaluations. Therefore we will say that the computational complexity for evaluating a new point is $\mathcal{O}(n_s)$.

Models like LS-SVM, RN, Kriging, kernel ridge regression and Gaussian processes do not have a sparse solution. Therefore an evaluation of a new point has a computational complexity of $\mathcal{O}(N)$. This might be a disadvantage for large data sets or if one needs fast evaluation in on-line processes. The Nyström and Cholesky factorization training algorithms reduce the computational cost of the training of the models. But they do not alter the computational complexity of the evaluations. This problem can be overcome by pruning the α-values a posteriori. After training the α_i that are below a certain threshold are put to zero. This idea of *sparse approximation* was proposed in [128] and [129].

The models based on a reduced feature space basis and the Fixed Size LS-SVM (Section 5.4 and 5.5) have a sparse solution. By limiting the number of parameters in the models, their computational complexity reduces to $\mathcal{O}(m)$. In this way, one controls the time that is needed to evaluate new points.

5.7 Experiments

5.7.1 Regression

To test some of the low rank models on regression tasks we will study the following example. We make a regression of the sinc-function based on a training sample of 3334 points disturbed by Gaussian noise. The results of the incomplete Cholesky and the Nyström low rank models in comparison to the original LS-SVM are shown in Figure 5.1. One can see that the

	LS-SVM	Incomp. Chol.	Nyström
bias	1.02e-002	1.02e-002	2.67e-002
MSE	1.45e-004	1.48e-004	1.02e-003

Table 5.2: The normalised bias and the mse for the sinc regression task for the LS-SVM, the incomplete Cholesky and the Nyström factorization models. The differences in performance are small.

performance of the three models is almost equivalent. In the bottom part of this figure can we see the true bias $=\frac{\sin x}{x} - f(x)$ measure for all three methods in a test set. One can see that the bias for the incomplete Cholesky and the original LS-SVM formulation are almost identical. The true bias of the model based on the Nyström factorization is slightly higher. This can also be seen in the normalized bias and the MSE computed over a test set as can bee seen in Table 5.2.

A remarkable result of this example is that the true rank of the kernel matrix is only 24. The rank as approximated by the incomplete Cholesky factorization is 13. Since the Nyström approximation was again based on 10% of the data, it is surprising that the model based on the incomplete Cholesky has a better performance although the training is based on a smaller part of the total kernel matrix.

5.7.2 Classification

In this experiment we compare two low rank approximations: the Incomplete Cholesky and the Nyström factorization on the UCI benchmarking data sets. For the Nyström factorization we use the computationally simplest method as formulated in Equation (5.16) where the MinRes method is used to solve the smaller linear system. We make a comparison of the number of data points in the data set (which is equal to the size of the original linear set to be inverted), the true rank of the kernel matrix as a result of an SVD and the estimated rank as given by incomplete Cholesky factorization. This last one is equal to the size of the linear system to be inverted by the reduced formulation. The stopping criterion of the incomplete Cholesky factorization monitors the diagonal elements during factorization. As soon as this approaches zero the iteration is terminated because it indicates that the full rank is covered. In our experiment the stopping criterion was very rough. As soon as the diagonal elements are smaller than $1e-3$ the factorization process is terminated. One can see in Table 5.3 that the incomplete Cholesky rarely leads to a significant reduction in computational complexity

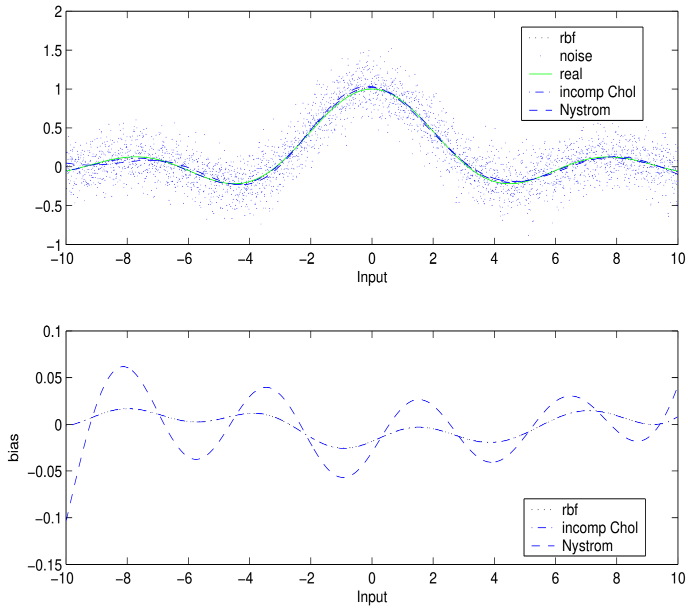

Figure 5.1: Performance of original LS-SVM model, a low rank approximation with incomplete Cholesky and a low rank based on the Nyström approximation is applied. The top figure gives the prediction based on the training data as shown. One can see that the performance of the three models is almost equivalent. In the bottom part of this figure can we see the true bias $=\frac{\sin x}{x} - f(x)$ measure for all three methods in a test set. One can see that the true bias for the incomplete Cholesky and the original LS-SVM formulation are almost identical. The true bias of the model based on the Nyström factorization is slightly higher.

	N	r.r.	a.r.	AUC_{chol}	SE_{chol}	AUC_{Nyst}	SE_{Nyst}
acr	460	460	451	0.939	0.018	0.940	0.018
bld	230	228	100	0.764	0.046	0.762	0.047
gcr	666	666	666	0.774	0.028	0.698	0.032
hea	180	180	180	0.890	0.034	0.878	0.039
ion	234	234	227	0.998	0.007	0.941	0.030
pid	512	512	500	0.812	0.028	0.798	0.030
snr	138	138	138	0.912	0.036	0.766	0.057
ttt	638	638	638	1.000	1e-4	0.648	0.032
wbc	455	313	174	0.994	0.004	0.994	0.004
adult	2000	1984	1540	0.880	0.003	0.861	0.004

Table 5.3: Results of the low rank approximations based on incomplete Cholesky and the Nyström factorization. This table shows the real rank (r.r.) and the approximated rank (a.r.) of the kernel matrix together with the test set performance measure by the AUC with its SE for the Incomplete Cholesky and Nyström approximation. From these results we may conclude that the low rank approximation based on the incomplete Cholesky has a very good performance but as we have seen almost no computational advantage. The model based on the Nystrm approximation has a significant computational improvement since it is based on a subsample of $m = (N/(10))$. If we compare the classification performances we notice that in only 2 out of 10 (the tic-tac-toe and Sonar data sets) there is a larger decrease in performance. In all the other tests the decrease in performance in small compared to the computational advantages.

because the kernel matrix still has a rank that is only a bit smaller then N.

As a second aspect of this test we compared the classification performance on a test set of the low rank models. This performance is measured as the area-under-the-curve (AUC) of an ROC curve together with the standard error (SE). The results are also shown in Table 5.3. One can compare these results with the results shown in Table 4.1. From these results we may conclude that the low rank approximation based on the incomplete Cholesky has a very good performance but as we have seen almost no computational advantage. The model based on the Nyström approximation has a significant computational improvement since it is based on a subsample of $m = \frac{N}{10}$. If we compare the classification performances we notice that in only 2 out of 10 (the tic-tac-toe and Sonar data sets) there is a larger decrease in performance. In all the other tests the decrease in performance in small to the computational advantages.

5.8 Conclusions

In this chapter we have shown that the computational complexity of the kernel models can be reduced significantly by using low rank approximations. We discussed two types of approximations.

The first one made use of a factorization of the kernel matrix. By making use of the Sherman-Morrison-Woodbury formula this factorization of the kernel matrix can be exploited for efficiently solving the linear systems. The Nyström approximation makes a factorization of the kernel matrix at a fixed but low computational cost. We showed different types of Nyström factorization and their corresponding computational results. Furthermore we did also study the incomplete Cholesky factorization as possible alternative for the Nyström factorization. This method fully exploits the rank of the kernel matrix in an iterative way. For the regression task we have shown, an impressive decrease in computational complexity. However, in some classification tests this same method gave only small computational advantages. On the same classification experiment the Nyström factorization gave a good performance. Only in some rare examples a decrease in performance, caused by the approximations, were observed.

A second class of methods to reduce the computational complexity are based on different methodologies to reducing the number of parameters in the models. This can be achieved by either reduce the number of parameters in the dual space by using an optimal basis representation in feature space, or by reducing the number of parameters in primal space by making use of the Nyström approximation, which leads to the *fixed size approximation* with a sparse representation. Hereby one constructs a basis or subsample by using different selection criteria.

Table 5.4 summarizes the computational complexity of training and model evaluation for different proposed low rank methods. We assume that one does no active selection of the subsample of size m and the linear systems and eigenvalue problems are computed by direct methods. We may conclude that the computational advantages resulting from these different low rank models are almost similar. Depending on the task one method gives a better computational advantage compared to the other. More intensive benchmarking is necessary to decide which model has the best overall performance. From the learning performance perspective a significant reduction in generalization performance is rare. To make a comparison with

	Training	**Model Evaluation**
Nyström factorization	$\mathcal{O}(m^2N + m^3)$	$\mathcal{O}(N)$
Cholesky factorization	$\mathcal{O}(p^2N + p^3)$	$\mathcal{O}(N)$
Reduced basis	$\mathcal{O}(m^2N + (m+1)^3)$	$\mathcal{O}(m)$
Fixed Size LS-SVM	$\mathcal{O}(m^2N + 2m^3)$	$\mathcal{O}(m)$

Table 5.4: Computational Complexity of training and model evaluation for different proposed low rank methods. We may conclude that the computational advantages for training resulting from these different low rank models are almost similar.

the models described in Chapter 3 one can make the following rule of thumb. For the models described in Chapter 3 the maximum advised computational complexity was $\mathcal{O}(N^3) = \mathcal{O}(5000^3) \approx 10^{11}$ in a worst case scenario. A similar restriction limits the low rank models to $N < 50000$ since the maximum computational complexity is $\mathcal{O}(m^2N) = \mathcal{O}(m^2 50000) \approx 10^{12}$ where we advise a sample size of $m = \frac{N}{10}$. Therefore these low rank approximation are promising and advisable in most large scale tasks with $5000 < N < 50000$.

Chapter 6

Ensemble Learning

The Whole Is More than the Sum of Its Parts.

We saw in the previous chapters that data set where the number of data points exceeds 50000 using a non-linear kernel are difficult to train by only one kernel model. In all the previously explained methods the goal was to train *one* model on the whole data set. A different methodology is based on the idea of 'divide and conquer'. This is done based on *ensemble learning*. In ensemble learning one combines different models trained on a data set $\mathcal{D}$. Most of the literature on ensemble learning uses this strategy to improve the performance of the individual models where each of them is trained on almost the whole original data set $\mathcal{D}$. In this chapter our objective will be to study this methodology for tackling the memory and computational problems kernel models face. Instead of training one kernel model on the whole data set, we will train a collection or ensemble of models each on a *subset* of the original data set $\mathcal{D}$. These submodels, trained on subsets, will be combined afterwards to have a new ensemble model estimator which is trained on the whole data set.

First we will give a general introduction on different ensemble learning strategies as found in the literature. We will explain these different strategies from a new viewpoint in which we make a clear distinction between the phase of the ensemble creation (Section 6.1.1) and the combination phase (Section 6.1.2). After this introductory part, we will overview different ensemble learning strategies, focussing on kernel models and explain there use towards training large data set. We will zoom in on recent theoretical results on ensembles of kernel models based on the theory of stability of learning algorithms. A comparison of the computational complexity of some of the combined models will be made.

In a last part we will introduce a new ensemble learning methodology where the training of the individual elements of the ensemble is done in a coupled way. Use of this *coupled learning* will lead to models which are more robust towards so called 'outlier' submodels. This is achieved by an information exchange during training of the individual submodel caused by the coupling. The idea of coupled learning will be motivated based on the idea of coupled local minimizer and multitask learning. We will also show how this coupled learning can be used for transductive learning and how it may introduce a sparse solution. All the results are experimentally verified.

6.1 Introduction

It is well-known in the literature that combining different models is a fruitful way for creating new improved estimators. Instead of using one learning algorithm ensemble learning uses a set or *ensemble* of models. This ensemble is indicated by $\{f^{(j)}\}_{j=1}^{q}$ where each model is trained on the data set $\mathcal{D}$. Each of the models $f^{(j)}, j = 1, \ldots, q$ is called a *submodel*. The *ensemble model* f then combines these q *submodels* to make new predictions. Ensemble Learning can be done both for classification and regression problems. For time-series prediction (sub)sampling techniques are more complicated because of the ordered data. Therefore it will not be handled in this work.

According to the literature the main goal for a good ensemble model is that the submodels should be both *accurate* and *diverse* ([65],[74],[127]). The fact that they should be accurate means that their individual predictive power should be 'fairly good'. For classification this would mean that the misclassification rate should be less than 50%. The diversity of the models, as a criterium, is more subtle. The divergence or non-agreement of the individual algorithms is intuitively plausible. If all the submodels would be identical, combining them would not improve the performance. However, the diversity between the models also should not be emphasized too much. It somethimes deteriorates the accuracy of the ensemble models. We will see this in the sequel.

The reasons why ensemble models work well in many practical applications can be explained by the following three fundamental reasons [33]:

- the *statistical reason*: every learning algorithm tries to find a good hypothesis f that predicts the data based on a data set $\mathcal{D}$. Since we only dispose of a limited amount of data $\mathcal{D}$, it is is perfectly possible that there are several hypotheses $f_1, f_2, \ldots, f_q$ that have the same overall accuracy. Assume each of the hypotheses f_i has a performance which is fairly good over the input domain $\mathcal{X}$, but in some regions some hypothesis f_j performs significantly better than others. By constructing an ensemble model, one takes the 'average' over all predictions thereby reducing the risk of taking a wrong hypothesis in a certain region of $\mathcal{X}$.

- the *computational reason*: Every kernel model depends on its hyperparameters. For a kernel model using a radial basis kernel these are both the regularization parameter γ and the bandwidth σ. The optimal hyperparameters are typically found by procedures like cross-validation ([150],[11]). These non-convex optimization problems are computationally very intensive and have many local minima. To overcome this problem of non-uniqueness of the solution, a combination is made from the different 'suboptimal' (local minima) solutions. Each of the individual solutions can be the end-result of a local optimization started form different initial points. The combination of these results may provide a better approximation of the true unknown function. A second computational advantage will be studied in this chapter. We use ensemble methods to tackle the problem of learning with large data sets. Instead of training one model on the whole data set, one can train an ensemble on subsets of the original data set.

- the *representational reason*: every learning algorithm tries to find the optimal hypothesis f to approximate the true function f^* out of a limited set of candidate hypotheses $\mathcal{F}$. So, each learning algorithm is limited to its own hypothesis space $\mathcal{F}$. If we now take a set of learning algorithms each working on its individual hypotheses space $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_q$ it is possible to expand the hypotheses space by taking weighted combinations of the individual learning algorithms. So, by combining the individual hypotheses it is possible to find a new hypothesis $f \notin \mathcal{F}_1 \cup \mathcal{F}_2 \cup \ldots \cup \mathcal{F}_q$. Notice that it is not only the choice of the learning algorithms (for example the kernel) that limits the hypothesis space, but also the limited amount of data. Also this last reason makes that the individual hypotheses spaces $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_q$ are limited in the number of different hypotheses.

These are three intuitive reasons that explain why ensemble learning can work.

In order to make a classification of all the different ways to do ensemble learning, we will make a subdivision based on the two steps needed to design an ensemble model. The first step is to create a meaningful set *ensemble* of submodels $\left\{f^{(j)}\right\}_{j=1}^{q}$. This will be handled in Section 6.1.1. Once the ensemble is created, the second step is to combine those into an *ensemble model* f. Different combination strategies will be discussed in 6.1.2.

6.1.1 Creating the ensemble

As a first step in creating the ensemble model, the set of different submodels has to be constructed. Obviously that means that one has to choose from which class the individual models are taken: neural networks, decision trees, kernel models,... Here, we will focus on kernel models, but all the different strategies that are considered in this section are applicable to every class of learning models. Once this choice is made one can construct ensembles according to four different strategies.

A first way is by *manipulating the training examples.* By training each of the learning algorithms on a different data set, different hypotheses $\left\{f^{(j)}\right\}_{j=1}^{q}$ are created. This technique is most suited for *unstable* algorithms, which are algorithms whose predictions change significantly in response to small changes in the training data $\mathcal{D}$. The most straightforward way of manipulating the data is by making *bootstrap samples* of the original data set. Combinations or aggregations of different algorithms trained on bootstrap samples are called *bootstrap aggregation or bagging* [12]. A second method are *crossvalidated committees* [101]. Here, the data set $\mathcal{D}$ is randomly divided into q disjoint subsets for training q separate learning algorithms. With these q disjoint subsets q overlapping data sets are created by leaving out a different subset. On each of these overlapping data sets an algorithm is trained. A third way of using the input data is by iterative refinement or adaptive resampling. The methods make combinations of individual algorithms in a similar way done by the bagging methods. The difference is that this sampling is done adaptively. One starts with training the first learning algorithm on a subset of the whole training set $\mathcal{D}$ sampled according to a uniform sampling distribution. After the first iteration, the performance is measured on the whole data set $\mathcal{D}$. According to this performance the sampling distribution is modified. A new data set is sampled out of $\mathcal{D}$ according to the new sampling distribution. Based on this new subset a new learning algorithm is trained, which will be combined with the previous one. This procedure is repeated several times. A lot of work has been done in this area for classification by Breiman ([13],[14]) known as *arcing (Adaptive Resample and Combine)* and by Schapire [117] via *boosting.* Also recently this work was extended towards regression. An overview with links to learning theory, optimization theory and robustness is given in the work of Rätsch [111].

A second class for building ensemble models is by working on the *input features*. If one has many input features d it may happen that these are highly correlated or even that some are redundant. Ensemble models exploiting this property may improve the generalization performance. This can be done by training a set of algorithms each working on a subset of the input values. One can group the input features together based on prior or expert knowledge or based on subselection criteria like principal component analysis [33].

A third class works on the *output features*. This class was already discussed in the first chapter without mentioning the relations with ensemble learning. Many multiclass classification problems are solved by combining multiple binary classification problems. So in this sense, multiclass classification models can be seen as ensembles of binary classification problems. Many different combination schemes exist for combining the binary classification algorithms, based on different types of output coding. The simplest way is by majority voting; more advanced methods use error correcting codes or information criteria. For an overview of these methods see [132],[144] and [123].

A last method for generating an ensemble of algorithms is by *injecting randomness* in the training process. This can be done on different levels. If the training process is non-convex, randomness can be introduced by using different starting points. Each of the different starting points will evolve to its local minimum. These different solutions result in different algorithms with possibly different predictions which then will be combined in the ensemble model. Other ways of introducing randomness are done by adding artificial noise to the input or output data. This improves the robustness of the algorithms. This is perhaps the most direct way to test and improve the stability of an algorithm. For more information about this subject we advise [101], [33] and [34].

The above strategies generate a set of submodels $\left\{f^{(j)}\right\}_{j=1}^{q}$. The following step is to combine these into an ensemble model.

6.1.2 Combined models in ensemble learning

Assume that one has a set of q submodels $\left\{f^{(j)}\right\}_{j=1}^{q}$. The next task is to combine those. The explanation will be given for regression but can be generalized towards classification tasks. The combination methods can be divided in four different classes of increasing complexity.

The first method averages the individual predictions of the subalgorithms. The ensemble models f takes the form

$$f(\mathbf{x}) = \frac{1}{q} \sum_{j=1}^{q} f^{(j)}(\mathbf{x}). \tag{6.1}$$

In classification, where $f(\mathbf{x}) = \text{sign}(\frac{1}{q} \sum_{j=1}^{q} f^{(j)}(\mathbf{x}))$, this comes down to a majority voting system. This method is very simpleand performs in a very robust way in many applications ([32],[25]).

A second way is a weighted averaging. The individual weights $\beta^{(j)}$ can be appointed to the individual models $f^{(j)}$ according to the trust one has in this model:

$$f(\mathbf{x}) = \sum_{j=1}^{q} \beta^{(j)} f^{(j)}(\mathbf{x}). \tag{6.2}$$

There exist many different methods for computing there weights. Popular techniques are boosting [117], committee networks ([106],[105]) or based on Bayesian methods ([7], [120],[155]).

A third way of combining the set of models makes use of the input for deciding which subalgorithms recieves most attention. These models are called *mixture of experts* [7]. Their structure can be written as:

$$f(\mathbf{x}) = \sum_{j=1}^{q} \beta^{(j)}(\mathbf{x}) f^{(j)}(\mathbf{x}). \tag{6.3}$$

The weights $\beta^{(j)}(\mathbf{x})$ are functions defined on the input domain $\mathcal{X}$. These weight functions are often called *gating networks.*

A last model makes use of an extra learning model, which acts on top of the other models. In this sense one can see it as a learning structure in different layers, 'stacked' on top of each other. The first layer is the set of models $\{f^{(j)}\}_{j=1}^{q}$, the second layer is a new algorithm or function $h : \mathbb{R}^q \to \mathbb{R}$ such that:

$$f(\mathbf{x}) = h\left(f^{(1)}(\mathbf{x}), \ldots, f^{(q)}(\mathbf{x})\right). \tag{6.4}$$

This method is called *stacking.* There exist of course many ways of constructing such a multi-layer learning scheme. For an overview see the work of Wolpert [154].

6.1.3 Ensembles and large scale applications

In the previous section we have seen many different methodologies to design an ensemble model. Originally the reason for this new way of constructing learning algorithms, was to build a better algorithm with better generalization performance. In this way ensemble learning is a way of *meta-learning*. It is a way of using the already existing classes of learning models, like neural networks, kernel model and decision trees, to create even more powerful learning structures.

In this work we have a very specific reason for introducing these ensemble models. In the previous chapters we have explained that training kernel models on large data sets is not straightforward because of the memory requirements. One can of course parallellize many of the numerical procedures introduced in the previous chapters. However, the approach we propose is to use ensemble models to tackle this problem. Instead of training one model on all the data $\mathcal{D}$ we subdivide this task among a set of kernel models each working on a subset of the data set $\mathcal{D}$. With these submodels the ensemble models f can then be created. With this strategy we try to divide the computational burden among the submodels.

In the following section we will highlight some of the already described methods, but now focussing on ensembles of kernel models. The main objective is training ensembles on large data sets.

6.2 Bagging

Firts we give an overview of the theoretical and experimental characteristics that have been proven recently in literature. We will focus on ensembles where the submodels are based on subsets of size N. Later the generalization for models based on subsets of size $g \leq N$ will be shown. This will create a framework in which we will motivate our further studies.

Bagging or Bootstrap Aggregation, as first introduced by Breiman [12], makes combinations of different submodels by simply averaging them (6.1). Hereby the individual submodels are trained on *bootstrap samples* of the original data set. These bootstrap samples are made by random sampling N elements with replacement of the original data set of size N. It was proven by Breiman that this strategy can improve the performance of *unstable* algorithms. It can be seen as follows. All the ensemble models that we used were based on q submodels. These ensemble models $f = \frac{1}{q}\sum_{j=1}^{q} f^{(j)}$ can be seen as a finite approximation. Therefore we use the following notation in this section: an ensemble model is given by $E_{\mathcal{D}}[f_{\mathcal{D}}]$ where $f_{\mathcal{D}}$ is a general model trained on the data set $\mathcal{D}$ and $E_{\mathcal{D}}[\centerdot]$ is the expected value over all data sets $\mathcal{D}$. As an example $f_{\mathcal{D}}$ can be either decision trees, neural networks or some other learning model. $E_{\mathcal{D}}[f_{\mathcal{D}}]$ is then a combination of these learning models. The generalization performance of the ensemble model $E_{\mathcal{D}}[f_{\mathcal{D}}]$ compared to the single model $f_{\mathcal{D}}$ is better if

$$E_{\mathcal{D}}\left[(f_{\mathcal{D}}(\mathbf{x}) - E[y|\,\mathbf{x}])^2\right] \geq E_{\mathcal{D}}\left[(E_{\mathcal{D}}[f_{\mathcal{D}}(\mathbf{x})] - E[y|\,\mathbf{x}])^2\right], \tag{6.5}$$

which states that the mean squared error over all data sets $\mathcal{D}$ is smaller. But since for the single model $f_{\mathcal{D}}$ holds that

$$\begin{aligned} E_{\mathcal{D}}\left[(f_{\mathcal{D}}(\mathbf{x}) - E[y|\,\mathbf{x}])^2\right] &= \\ & E_{\mathcal{D}}\left[f_{\mathcal{D}}(\mathbf{x})^2\right] - 2E_{\mathcal{D}}[f_{\mathcal{D}}(\mathbf{x})]\,E[y|\,\mathbf{x}] + E[y|\,\mathbf{x}]^2 \end{aligned} \tag{6.6}$$

and for the ensemble model $E_{\mathcal{D}}[f_{\mathcal{D}}]$:

$$\begin{aligned} E_{\mathcal{D}}\left[(E_{\mathcal{D}}[f_{\mathcal{D}}(\mathbf{x})] - E[y|\,\mathbf{x}])^2\right] &= \\ & E_{\mathcal{D}}[f_{\mathcal{D}}(\mathbf{x})]^2 - 2E_{\mathcal{D}}[f_{\mathcal{D}}(\mathbf{x})]\,E[y|\,\mathbf{x}] + E[y|\,\mathbf{x}]^2 . \end{aligned} \tag{6.7}$$

This generalization performance is only better if $E_{\mathcal{D}}\left[f(\mathbf{x})^2\right] \geq E_{\mathcal{D}}[f(\mathbf{x})]^2$. A hypothesis that obeys this property is said to be *unstable* according to Breiman [12]. These are hypotheses whose predictions change significantly in response to small changes in the training data $\mathcal{D}$. Note that by rewriting this inequality one gets

$$E_{\mathcal{D}}\left[f_{\mathcal{D}}(\mathbf{x})^2\right] - E_{\mathcal{D}}[f_{\mathcal{D}}(\mathbf{x})]^2 = E_{\mathcal{D}}\left[(f_{\mathcal{D}}(\mathbf{x}) - E_{\mathcal{D}}[f_{\mathcal{D}}(\mathbf{x})])^2\right] \geq 0, \tag{6.8}$$

which is recognizable as the variance of the random variable $f_{\mathcal{D}}(\mathbf{x})$. It is often said that, in correspondence to other bootstrap methods, bagging reduces the variance but it does not reduce the bias since for bagging models $f_{\mathcal{D}} \approx E_{\mathcal{D}}[f_{\mathcal{D}}]$.

A second proof of the working of bagging was recently given based on the concept of stability of algorithms. Many different definitions of stability exist: uniform stability [9], α-stability [112], hypothesis stability [36],... Each of them leads to a different bound on the generalization error in terms of the leave-one-out estimate. For an overview we advise [36]. In the pioneering work of Bousquet and Elisseeff [9] it was formally proven that there exists a strong relation between the uniform stability and both the generalization and the leave-one-out error of a learning algorithm. Some preliminary work in the case of noisy labels was done in [115]. The concept of stability they used, is defined as:

Definition 2 ***Uniform stability [9]:*** *Let the model $f_{\mathcal{D}}$ be the outcome of a symmetric and deterministic algorithm trained on a data set $\mathcal{D}$. The uniform stability $\beta_{\mathcal{D}}$ with respect to the loss function V is defined by*

$$\beta_{\mathcal{D}} = \sup_{\mathcal{D}} \left\| V(f_{\mathcal{D}}, .) - V(f_{\mathcal{D}\backslash i}, .) \right\|_{\infty}, \tag{6.9}$$

where $\mathcal{D}\backslash i$ indicates the data set $\mathcal{D}$ with the i-th point removed.

This means that the loss measured by the loss function V of a model $f_{\mathcal{D}}$ (see Section 2.1) will never be bigger than $\beta_{\mathcal{D}}$ if one removes one point from the training set. The smaller $\beta_{\mathcal{D}}$, the more stable the algorithm is. An algorithm is said to be *(strongly) stable* if $\beta_{\mathcal{D}} = \mathcal{O}\left(\frac{1}{N}\right)$. From this definition they proved the following bound on the expected risk (see Section 2.1):

Theorem 3 ***Bousquet and Elisseeff [9]:*** *Let $f_{\mathcal{D}}$ be the outcome of an algorithm with stability $\beta_{\mathcal{D}}$ with respect to the loss function V such that $M \in \mathbb{R}$ is defined as $0 \leq V(f_{\mathcal{D}}, y) \leq M$, for all $y \in \mathcal{Y}$ and all sets $\mathcal{D}$ of size N. Then for any $N \geq 1$ and any η between $0 \leq \eta \leq 1$ the following bound holds with probability at least $1 - \eta$ over the random draw of the sample $\mathcal{D}$:*

$$I\left[f_{\mathcal{D}}\right] \leq I_{emp}\left[f_{\mathcal{D}}, \mathcal{D}\right] + 2\beta_{\mathcal{D}} + \left(4N\beta_{\mathcal{D}} + M\right)\sqrt{\frac{\ln 1/\eta}{2N}}. \tag{6.10}$$

This shows that it is best to create stable algorithms. However, to formally apply the concept of stability for proving the benefits of bagging as described by Breiman, Elisseeff *et al.* [36] needed to use the closely related but weaker concept of *hypothesis stability* β_h. They proved that:

$$\begin{aligned} E_{\mathcal{D}}\left[\left(E_{\mathcal{D}}\left[f_{\mathcal{D}}\left(\mathbf{x}\right)\right] - E\left[y|\,\mathbf{x}\right]\right)^2\right] &\leq E_{\mathcal{D}}\left[\left(f_{\mathcal{D}}(\mathbf{x}) - E\left[y|\,\mathbf{x}\right]\right)^2\right] \qquad (6.11) \\ &\leq E_{\mathcal{D}}\left[\left(E_{\mathcal{D}}\left[f_{\mathcal{D}}\left(\mathbf{x}\right)\right] - E\left[y|\,\mathbf{x}\right]\right)^2\right] + \frac{N\beta_h}{4} \\ &\leq E_{\mathcal{D}}\left[\left(E_{\mathcal{D}}\left[f_{\mathcal{D}}\left(\mathbf{x}\right)\right] - E\left[y|\,\mathbf{x}\right]\right)^2\right] + \frac{N\beta_{\mathcal{D}}}{4}. \end{aligned}$$

The last step in this bound holds since the uniform stability is an upper bound on the hypothesis stability, $\beta_h \leq \beta_{\mathcal{D}}$. This shows that constructing ensemble models, by a bagging procedure, theoretically has advantages for unstable (sub)models $f_{\mathcal{D}}$. Additionally, it shows that for stable algorithms improvement by bagging might be very small.

Literature is not always very clear about this bagging with kernel models. In general, kernel models are regarded as stable algorithms. In [9] Bousquet and Elisseeff derived a very clear expressions for the uniform stability of both SVMs and Regularization Networks. Since the upper limit on the stability depends on the regularization parameter γ, the stability depends on how this regularization parameter scales with the number of data points N. If this regularization parameter is a constant, then $\beta_{\mathcal{D}} = \mathcal{O}\left(\frac{1}{N}\right)$, which indicates a stable algorithm. However, in [37] it is mentioned that the regularization parameters γ often increases with N and therefore the algorithm is not always stable. In these cases averaging helps to improve the stability. Therefore it makes sense to use methods like bagging on kernel models.

The influence of the regularization parameter on the stability can also be seen from a numerical point of view. To make the explanation easier we will show this for Regularization Networks in a regression setup[1]. Relations between the condition number and the stability were already mentioned in [90]. If $f_{\mathcal{D}}(\mathbf{x})$ is the evaluation in $\mathbf{x}$ of a model trained on a data set $\mathcal{D}$ and $f_{\mathcal{D}\backslash i}(\mathbf{x})$ is a model trained on $\mathcal{D}\backslash i$, indicating the data set $\mathcal{D}$ with the i-th point replaced by a new point. The stability depends on the difference between the models:

$$\begin{aligned}\Delta f(\mathbf{x}) &= f_{\mathcal{D}}(\mathbf{x}) - f_{\mathcal{D}\backslash i}(\mathbf{x}) \\ &= \sum_{p=1}^{N} \alpha_p k(\mathbf{x}_p, \mathbf{x}) - \sum_{p=1}^{N} \alpha'_p k(\mathbf{x}'_p, \mathbf{x}),\end{aligned} \tag{6.12}$$

where $\mathbf{x}_p \in \mathcal{D}$ and $\mathbf{x}'_p \in \mathcal{D}\backslash i$. Note that the vectors α and α' are the solutions of the linear systems of the form (2.42) respectively over the data sets $\mathcal{D}$ and $\mathcal{D}\backslash i$. Since there is only a small difference in the kernel values, we will assume that $\Delta f(\mathbf{x}) \approx \mathbf{k}^T \Delta\alpha = \mathbf{k}^T(\alpha - \alpha')$ where $\mathbf{k} = [k(\mathbf{x_1}, \mathbf{x}) \ldots k(\mathbf{x_N}, \mathbf{x})]^T$ is the vector of the kernel values. Let us now see what the influence is on α if there is a change in the data set. Training of a regularization network on a data set $\mathcal{D}$ consists in solving the linear system $(K + \frac{1}{\gamma}I)\alpha = \mathbf{y}$. If the i-th data point is replaced by another point, this affects both the matrix K and the vector $\mathbf{y}$. Both changes can be seen as a perturbation of the linear system. In the case that $\mathbf{y} \neq \mathbf{0}_d$ and $\left\|\Delta\left(K + \frac{1}{\gamma}I\right)\right\|_2 < \left\|\left(\Delta\left(K + \frac{1}{\gamma}I\right)\right)^{-1}\right\|_2^{-1}$ the

[1] For LS-SVM models it is not straighforward to show how a change in the data set affects the solution because of the influence of the b-term.

resulting change in the solution is bounded by [30]:

$$\frac{\|\Delta\alpha\|_2}{\|\alpha\|_2} \leq \frac{\mathrm{cond}_2\left(K+\frac{1}{\gamma}I\right)}{1+\mathrm{cond}_2\left(K+\frac{1}{\gamma}I\right)\frac{\left\|\Delta\left(K+\frac{1}{\gamma}I\right)\right\|_2}{\left\|K+\frac{1}{\gamma}I\right\|_2}}\left(\frac{\left\|\Delta\left(K+\frac{1}{\gamma}I\right)\right\|_2}{\left\|K+\frac{1}{\gamma}I\right\|_2}+\frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2}\right)$$

$$= \frac{\frac{1}{\gamma}+\lambda_{\max}}{\frac{1}{\gamma}+\lambda_{\min}+(\frac{1}{\gamma}+\lambda_{\max})\frac{\left\|\Delta\left(K+\frac{1}{\gamma}I\right)\right\|_2}{\left\|K+\frac{1}{\gamma}I\right\|_2}}\left(\frac{\left\|\Delta\left(K+\frac{1}{\gamma}I\right)\right\|_2}{\left\|K+\frac{1}{\gamma}I\right\|_2}+\frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2}\right), \tag{6.13}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are respectively the minimum and maximum eigenvalues of K. (See also Appendix B.) This intuitively shows how a change of the models caused by a small difference in the data set is dependent on the condition number and therefore also the regularization parameter γ. The right part of this bound is monotone decreasing with respect to $\frac{1}{\gamma}$ which can easily be proven by taking the first derivative. This proves that stability of a RN, towards the removal of points form the data set, increases for higher regularization. In the limiting case where there is no regularization ($\frac{1}{\gamma} \to 0$) this bound is very conservative:

$$\frac{\|\Delta\alpha\|_2}{\|\alpha\|_2} \leq \frac{1}{\frac{\lambda_{\min}}{\lambda_{\max}}+\frac{\left\|\Delta\left(K+\frac{1}{\gamma}I\right)\right\|_2}{\left\|K+\frac{1}{\gamma}I\right\|_2}}\left(\frac{\left\|\Delta\left(K+\frac{1}{\gamma}I\right)\right\|_2}{\left\|K+\frac{1}{\gamma}I\right\|_2}+\frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{y}\|_2}\right). \tag{6.14}$$

Notice however, that until now, the methods that were discussed assumed that the averaging was done on bootstrap samples of size N. Since these samples are of equal size as the original data set, this does not decrease the computational complexity. Hence this method has no computational advantages towards large data sets. Therefore we are interested in averaging methods that work on subsets of $\mathcal{D}$ of size $g \leq N$. These models are sometimes referred to as *subagging models* [43]. Do the theoretical stability bounds still hold in that case?

These subagging algorithms, similar to bagging, can be seen as finite approximations of $E_{\mathcal{S}}\left[f_{\mathcal{S}}\right]$ where $f_{\mathcal{S}}$ is a model trained on a subset $\mathcal{S}$ and $E_{\mathcal{S}}\left[.\right]$ is the expectation with respect to a subset $\mathcal{S}$. One gets $\mathcal{S}$ by sampling $g \leq N$ points uniformly without replacement from $\mathcal{D}$. In [43] it was shown that this subagging model $E_{\mathcal{S}}\left[f_{\mathcal{S}}\right]$ is always stable.

Theorem 4 ***(Evgeniou* et al. *[43])*** *Assume that $f_{\mathcal{D}}$ has a uniform stability $\beta_{\mathcal{D}}$ when trained on $\mathcal{D}$. Then the uniform stability $\hat{\beta}_{\mathcal{S}}$ of the averaging model $E_{\mathcal{S}}[f_{\mathcal{S}}]$ is bounded as*

$$\hat{\beta}_{\mathcal{S}} \leq \frac{g}{N}\beta_{\mathcal{D}}. \tag{6.15}$$

In practice the average combination is replaced by a finite combination. Tests with ensembles of SVMs trained on subsets, show that the generalization performance of the models is good ([41],[43]). These ensembles of kernel models improve the leave-one-out bound both theoretically and experimentally. But, because of the randomization of the solution the former theorem (6.15) does not hold anymore. The theory of the stability of bagging and subagging algorithms was extended in [36] towards these finite combinations by making use of the concept of *randomized hypothesis stability.* This extension takes into account that the sampling/subsampling from the data set $\mathcal{D}$ is a random process. Remarkably, the results of these theories prove similar conclusions both on bagging and subagging as discussed previously. *Bagging as well as subagging increases the stability of unstable algorithms.*

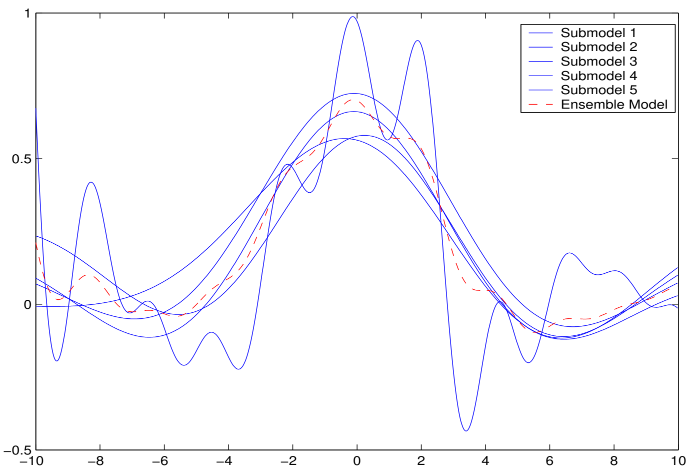

Figure 6.1: The ensemble of 5 submodels each trained on 1% of a large data set of 6667 training points with equal hyperparameters on all the submodels. The dashed line is the ensemble model Each of submodels is indicated by a full line. The majority of submodels gives an acceptable performance. However, one can see in this example that one particular submodel, which we define as an ‘outlier model’ gives a very bad performance.

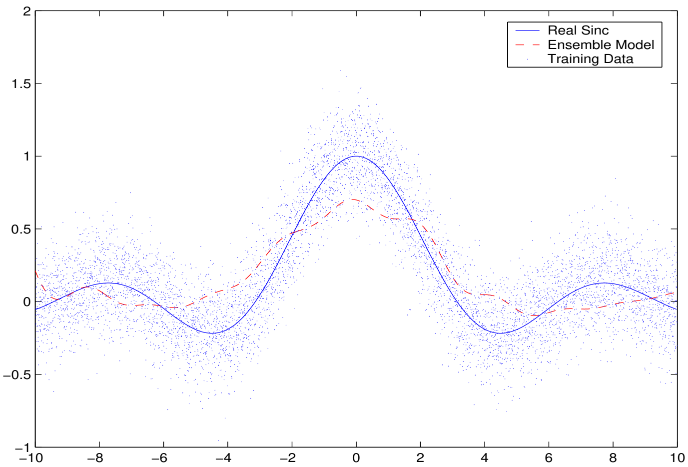

Figure 6.2: The ensemble model created on submodels shown in 6.1 with equal hyperparameters for all models. Here we see both the true sinc function (full line) and the training points where Gaussian noise is added. The ensemble model is given by the dashed line. We can see a clear bad performance of the bagging models which indicates the sensitivity of bagging toward 'outlier' models.

The experiments show that also for subagging a lot depends on the choice of the hyperparameters. A real improvement in performance of the subagging applied to SVMs depends on the choice of the regularization parameters C. But for a good choice of hyperparameters there is almost no difference in performance between the single SVM trained on the whole data set and the subagging models ([41],[43]). Because of its structure the computation involved in training the submodels on smaller subsets can be done more efficiently. The training of these models can be parallellized very easily.

A second conclusion that was drawn in these papers ([41],[43]) was that the increase in performance for bagging and subagging models saturates by increasing the amount of submodels q. This means that a limited amount of submodels is enough to capture all the generalization power of the ensemble model. The fewer models are necessary, the less computation is needed. This second advantage proves that subagging with kernel models is a valid alternative for handling large data sets.

To show the behavior of an ensemble of kernel models in practical 'large scale' application we did the following test. We will show the use of subagging on a subset for training large data sets. Hence, we want to mimic the situation where one has a very large data set where training on the whole data set is impossible because of its size. We will do a regression of the sinc function with a data set of 6667 data points where random normally distributed noise is added. Subsets of 1% of the data are sampled out of this large data set and on each of these subsets a submodel is trained. After training these submodels the ensemble model f is created by unweighted averaging of the submodels. All the submodels are LS-SVM models with the Gaussian radial basis function.

To check the influence of the hyperparameters selection we did the following. In a first test we find the optimal hyperparameters (γ, σ) for only one of the submodels by cross-validation based on a leave-one-out criterion These optimal hyperparameters are kept constant for all the other models of the ensemble.

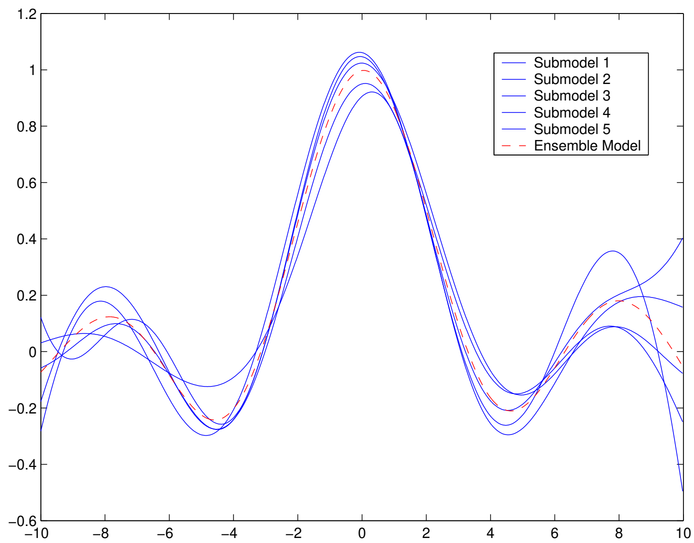

Figure 6.3: The ensemble of 5 submodels each trained on 1% of a large data set of 6667 training points. For each of the submodels a hyperparameter tuning was done based on a cross-validation routine. The dashed line is the ensemble model and the each of submodels are indicated by a full line. One can see that all of the submodels have a good performance.

In Figure 6.1 we can see that the performance of the different submodels is very different. The majority of models gives an acceptable performance when only trained in 1% of the data set. One particular submodel gives

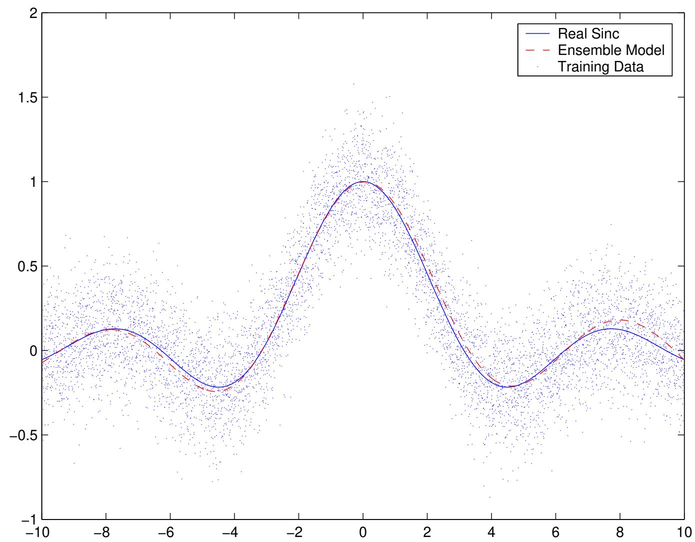

Figure 6.4: The ensemble model created on submodels with hyperparameter tuning as shown in Figure 6.3. Here we see both the true sinc function (full line) and the training points where Gaussian noise is added. The ensemble model is given by the dashed line. We can see that the performance of the ensemble models is better that the performance of the individual submodels.

a very bad performance. It is this one bad submodel that deteriorates the performance of the whole ensemble model as we see in the resulting ensemble model in Figure 6.2. This is a typical example of the unrobust behavior of an averaging towards 'outliers'.

In a second test we do a hyperparameter tuning by leave-one-out cross-validation for each of the submodels. These are again trained on only 1% of the data. The ensemble model is created based on these tuned submodels by unweighted averaging.

In Figure 6.3 we can see the performance of the different submodels with individual hyperparameter tuning. One can now see that performance of the individual submodels is almost similar. Combining these submodels results in an ensemble model as given in Figure 6.4. We can see that the performance of the ensemble model compared to the real sinc function is very good. This example shows that tuning the hyperparameters of the individual submodels is a necessity to avoid 'outlier' models, which disturb the total performance of the ensemble. A too high variance among the models does not guarantee a good generalization performance of the combined models. It is this problem that we later on try to solve by a new methodology that makes use of a coupling between the submodels during training (see Section 6.7).

For another regression test, we compared the performance on the abalone data [8] set. Here the goal is to predict the age of abalone from physical measurements. The age of abalone is normally determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope – a time-consuming task. Other measurements, which are easier to obtain, are used to predict the age. This gives a data set of 4177 instances with 8 attributes. The first 2800 are used as training set and the remaining 1377 are used as test set. To make a comparison with we first trained a single LS-SVM on the whole abalone data set. This is then compared to a bagging ensemble of 10 LS-SVM submodels each trained on 5% of the training set. Notice that for each of the individual submodels a hyperparameter tuning is done. The results are shown in Figure 6.5.

This figure shows that by using a bagging model trained on subsets, a performance can be achieved that is only slightly worse than the performance of a single kernel model trained on the whole data set. However, the bagging model has a much lower computational cost. The performance of this bagging model is better than the average performance of the individual submodels. The 'outlier' models in this example has a better performance than the average. This is of course not guaranteed in other examples.

Note that in these tests the size of the subsets can be tuned according to the physical memory available. This strategy clearly has a computational advantage. The time for training the single LS-SVM with hyperparameter tuning by using the cross-validation tool in LS-SVMlab took about 3.5 days on a Pentium 1Ghz with 512MB Ram. The bagging model was trained and evaluated, also with a hyperparameter tuning, in less than one hour. Hence this shows that using a bagging structure on subset is a valid way for tackling training on large data set with a limited amount of memory.

6.3 Boosting

One of the first attempts for combining kernel models trained on subsets for classification was done by Pavlov *et al.* [103], who used a Boosting approach to overcome the computational bottleneck of SVM training. The methodology of boosting is different from the other ensembles strategies already discussed because it is an iterative procedure. The class of *boosting* algo-

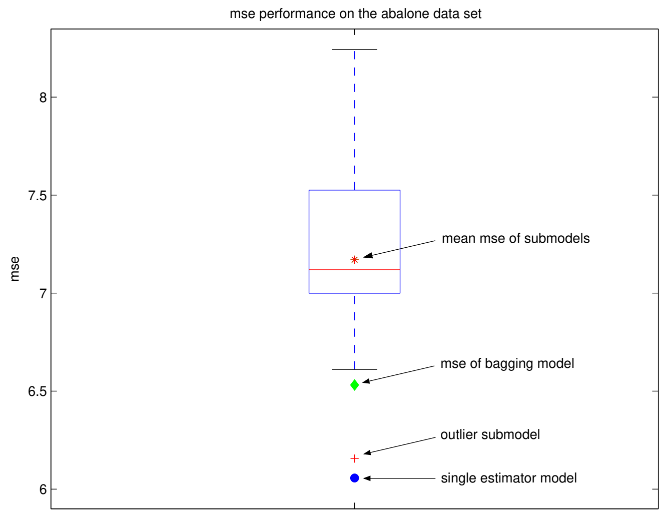

Figure 6.5: Here we show the mse performance on a test set of the Abalone data set. The boxplot shows the difference in mse performance of the submodels. Further the performances of the bagging model and the single LS-SVM are shown. This demonstrates that by using a bagging model trained on subsets of the training data, a performance can be achieved that is only slightly worse that the performance of a single kernel model trained on the whole data set and this with a much lower computational cost. One can see that the performance of this bagging model is better then the average performance of the of the individual submodels. The 'outlier' models in this example have a better performance than the average.

rithms ([117],[118],[111]) similar to methods as *arcing (Adaptive Resampling and Combine)* ([13],[14]) build the ensemble or set of submodels based on an iterative refinement of the learning algorithm.

During each step of the iterative proces new submodels are trained on a different training set. We will explain this for classification. The basic step in the boosting procedure is the construction of the training set or *boosting set*. This boosting set is found by sampling according to a predefined distribution over the whole data set $\mathcal{D}$. Typically, one starts with a uniform distribution where all points are equally weighted. After the sampling, a submodel will be trained in this boosting set. For the next iteration this sampling distribution is modified according to the performance submodel. The weight of those points that are misclassified most often, will be increased. In this way one creates a set of classifiers where one increasingly pays more attention to the difficult examples. In the end all the submodels are combined and weighted according to their individual performance on the training set.

Boosting can effectively convert a weak classifier (which does little better than random guessing) into a strong classifier (which can achieve an arbitrarily low error rate given sufficient training data). The main effect, similar to bagging, is a reduced variance. According to Breiman [13] arcing and boosting methods usually perform better than bagging.

Pavlov [103] showed experimentally that *Boosting* can be used to create ensembles of SVMs. Making ensembles of 10 to 15 SVMs each trained on subsets between 2 and 4 percent of the original data set, gives algorithms of which the learning performances are comparable to a single SVM trained on the whole data set. Additionally a computational improvement of a factor 10 was measured. Towards training on large data sets, this obviously is a good strategy. These experiments showed that the size of the subsample sets does not influence the performance of the boosted kernel model. However, for larger boosting sets, less iterations in the boosting procedure are needed. Therefore it is best to tune the size of the boosting set according to the available physical memory.

6.4 Mixture of experts

A third method where the individual learning algorithms are trained on *subsamples* of the original data set is *mixture of experts* [7]. In this ensemble model every submodel or ‘expert’ specializes in a certain task induced by its training set. As was explained in Section 6.1.2, parallel to the submodels a *gater* is trained. This gating network also uses the input and is trained to judge which of the trained submodels is best suited to determine the output. The gating network modifies the weights of the submodels according to the new input point that is given to the network. This gating network is on itself a learning algorithm, which is trained most often on a separate validation set. The validation set can be a part of the training set on which the submodels are trained or it can be a completely new set of training points that is kept aside during the training of the submodels.

This approach was successfully used with kernel models by Collobert *et al.* [26] for classification. They proposed to make a mixture of SVMs each trained on a subset of the training set. The combination of the individual outputs of the SVMs is then evaluated by a MLP Neural Network, which acts as *gater*. A crucial step in their approach is that the assignment of the data points to the individual SVM is done in several loops. The reassignment is done according to the confidence the gater has in an individual experts. An important advantage of this method is that it can be parallelized easily. According to their results, much depends on the complexity of the gater. The number of neurons and hidden layers of the MLP has a big influence on the performance. In this way, the computational bottleneck is now partially transferred to the training of the gater.

6.5 Stacking

The fourth method for combining the submodels uses a separate learning stage for finding the weights of the individual submodels $\left\{f^{(j)}\right\}_{j=1}^{q}$. Training of these ensemble models can be divided into two separate stages. In a first stage the individual submodels are trained on their corresponding subsets. The subsets can be disjoint or overlapping. Disjoint subsets can be constructed dividing the data set $\mathcal{D}$ into q sets. The overlapping subsets can be the result of random subsampling with replacement. This selection and training of the submodels can be seen as the training of the the *first layer* of the network structure.

Once the submodels are trained, the *second layer* is trained. This second layer is again a learning algorithm h such that $f(\mathbf{x}) = h\left(f^{(1)}(\mathbf{x}), \ldots, f^{(q)}(\mathbf{x})\right).$ This new learning algorithm will give a 'weight' to the each of the individual submodels according to the 'trust' it has in this model. This second layer is separately trained on a different training set. Again as in the case of mixture of experts, this can be the whole training $\mathcal{D}$ set on which the submodels were trained, a subset of this set $\mathcal{D}$ or a subset of $\mathcal{D}$ that has been kept aside.

We will overview some stacking strategies and study their computational complexity. The perhaps simplest way is by making a weighted average $f(\mathbf{x}) = \sum_{j=1}^{q} \beta^{(j)} f^{(j)}(\mathbf{x})$ of the submodels $\left\{f^{(j)}\right\}_{j=1}^{q}$ each trained on a subset $\mathcal{D}$. Once the submodels are trained, the weights $\left\{\beta^{(j)}\right\}_{j=1}^{q}$ are trained based on a least squares criterion. One often demands that $\sum_{j=1}^{q} \beta^{(j)} = 1$. In this way one makes an affine combination of the submodels. If $\beta^{(j)} \geq 0, \forall j$, one constructs convex combinations of the submodels $\left\{\beta^{(j)}\right\}_{j=1}^{q}$. These methods are called *committee networks* and were first introduced by Perrone ([106],[105],[7]). In Section 6.7.4 we will show that training the second layer results into solving a quadratic programming problem of dimension q. As an interesting property this solution of the QP problem is often sparse, which can have computational advantages.

Another way to design a stacking ensemble model is to train kernel models to combine the different estimations of the submodels. Hereby this 'second layer' kernel model is trained on data set $\mathcal{G}$ with

$$\mathcal{G} = \left\{ \left([f^{(1)}(\mathbf{x}_i), \ldots, f^{(q)}(\mathbf{x}_i)]^T, y_i \right); \forall (\mathbf{x}_i, y_i) \in \mathcal{D} \right\} \tag{6.16}$$

is based on the outputs of the first layer models. One has two possibilities for this 'second layer' kernel model: a linear or a non-linear model.

In this case of a non-linear model the computational complexity scales up with the number of data points N. If training of the second layer is done on the whole data set the computational bottleneck transfers it to this second layer. Therefore, on large data sets it is better to use a linear kernel in the second layer. We have seen in Chapter 2 that the training of an LS-SVM or RN kernel model with a linear kernel scales with the input dimension of this models. In this case it is equal to q, the number of submodels we are using in the first layer. Additionally in [154] it was shown that a linear second layer gives also the best performances for stacking models. This shows similarities with the neural network literature. Also in MLP's the architecture often consists of a first hidden layer with non-linear transfer function which are combined in a second layer by a linear output neuron. These stacking models were already applied in the literature on kernel models in ([75],[41],[43]). In these last two papers of Evgeniou *et al.* the idea was launched to use stacking with support vector machines trained on subsets and with a linear output layer. They experimentally showed that this approach is beneficial in the case of data sets with outliers. In general it was also more robust for different kernels used in the first layer or towards the hyperparameters tuning. We already showed that the hyperparameter tuning was also a difficulty in the case of bagging.

Of course many other ways of stacking are possible. In ([120],[155],[7]) they showed different ways to combine the ensemble of submodels in a Bayesian setup and in [129] neural networks were used.

6.6 The bias-variance for ensemble methods

In a similar way as for a single estimator (Section 2.8), one can compute the bias-variance trade-off for the combined estimator or ensemble model. In a statistical setup we are assuming that the relation between the input $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ is given by $y = f^*(\mathbf{x}) + e$ where $e \sim \mathcal{N}(0, \sigma)$ normally distributed noise and f^* is the true function to be estimated. Based on a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ we will estimate the optimal function $f(\mathbf{x})$ which approximates $f^*(\mathbf{x})$. Let us also assume that $f(\mathbf{x})$ is the ensemble model found by combining the submodels $\left\{f^{(j)}\right\}_{j=1}^{q}$ where each of the individual submodels $f^{(j)}$ is trained on a data set $\mathcal{D}^{(j)} \subseteq \mathcal{D}$. Then it holds that for the weighted combined models of the form $f(\mathbf{x}) = \sum_{j=1}^{q} \beta^{(j)} f^{(j)}(\mathbf{x})$:

$$E_{\mathcal{D}}\left[(f(\mathbf{x}) - E[y|\,\mathbf{x}])^2\right] = (\text{bias})^2 + \text{variance}. \tag{6.17}$$

The bias for this combined estimator is [85]:

$$\begin{aligned} \text{bias} &= E_{\mathcal{D}}\left[f(\mathbf{x}) - E[y|\,x]\right] \\ &= \sum_{j=1}^{q} \beta^{(j)} E_{\mathcal{D}^{(j)}}\left[f^{(j)}(\mathbf{x})\right] - E[y|\,\mathbf{x}]. \end{aligned} \tag{6.18}$$

It is easy to see that since $\sum_{j=1}^{q} \beta^{(j)} = 1$, the bias of the combined estimator is equal to the weighted average of the biases of the individual submodels.

$$\begin{aligned} \text{bias} &= \sum_{j=1}^{q} \beta^{(j)} E_{\mathcal{D}^{(j)}}\left[f^{(j)}(\mathbf{x}) - E[y|\,\mathbf{x}]\right] \\ &= \sum_{j=1}^{q} \beta^{(j)} \text{bias}^{(j)}, \end{aligned} \tag{6.19}$$

where $\text{bias}^{(j)}$ is the bias of submodel $f^{(j)}$. So, the bias depends only on the single submodel properties.

The variance of the ensemble model is [85]:

$$\begin{aligned}
\text{variance} &= E_{\mathcal{D}}\left[\left(f(\mathbf{x}) - E_{\mathcal{D}}\left[f(\mathbf{x})\right]\right)^2\right] \\
&= E_{\mathcal{D}}\left[\left(\sum_{j=1}^{q}\beta^{(j)}f^{(j)}(\mathbf{x}) - \sum_{j=1}^{q}\beta^{(j)}E_{\mathcal{D}}\left[f^{(j)}(\mathbf{x})\right]\right)^2\right] \\
&= \sum_{j=1}^{q}\left(\beta^{(j)}\right)^2\left\{E_{\mathcal{D}^{(j)}}\left[\left(f^{(j)}(\mathbf{x})\right)^2\right] - \left(E_{\mathcal{D}^{(j)}}\left[f^{(j)}(\mathbf{x})\right]\right)^2\right\}
\end{aligned}$$

$$+\sum_{j\neq l}^{q}\beta^{(j)}\beta^{(l)}\left(E_{\mathcal{D}}\left[f^{(j)}(\mathbf{x})\,f^{(l)}(\mathbf{x})\right] - E_{\mathcal{D}^{(j)}}\left[f^{(j)}(\mathbf{x})\right]E_{\mathcal{D}^{(l)}}\left[f^{(l)}(\mathbf{x})\right]\right). \tag{6.20}$$

In this formulation we see that the variance consists of two terms. The first term of the variance equals the weighted average of the submodel variances. The second term measures the average covariance between the different submodels. In the limit, this term vanishes when the models are totally independent of each other. In normal circumstances, however, the second term explicitly depends on correlation between the single estimators. Therefore it is beneficial to attempt making the submodels as weakly correlated as possible in order to decrease the variance. In the literature this is often referred to as that the models should *disagree* [74]. This, however, should be done with care because one wants to avoid increasing either the bias or the variances of individual submodels.

6.7 Coupled ensemble learning

6.7.1 Introduction

In the previous sections we showed that constructing ensemble models is beneficial for improving the generalization performance. We explained that this was caused by a better stability of the ensemble models (see Section 6.2). Stability in this terms could be explained as the sensitivity towards small changes in the data set. In case of unstable algorithms this leads to a big variance between the submodels which can be reduced by bagging. Further we discussed that variance among the submodels is not only caused by a change in data sets. Experimentally we showed that ensembles where the individual submodels did not have properly tuned hyperparameters, could also have a large variance among the submodels. Outlier models could even lead to a bad performance of the whole ensemble model.

Furthermore we explained that based on the bias-variance trade-off of ensemble models in Section 6.6 it was advised that the correlation between the submodels has a direct impact on the variance of the generalization performance of the ensemble model. Submodels that are highly uncorrelated are better candidates to construct ensemble models. To study this behavior and to control the variance among the submodels we designed the following *coupled learning* setup.

In all previously mentioned methods the training of the individual submodels is done independently from each other. In [62] we presented the technique where we *couple* the learning processes of the individual submodels. In this section we will explain this idea in more detail. The idea behind this method is that the different estimators of the ensemble are each trained on disjoint training sets, but have to agree on a predefined set of chosen data points. This *coupling set* can be chosen freely. It can be a part of the training set, chosen randomly or via more intelligent sampling criteria. But notice that it can also be the test set. In this way the coupling can be done in a *transductive* way ([51],[21],[69],[121]).

Another related ensemble technique that offers transductive inference is the Bayesian Committee Support Vector Machine presented by Schwaighofer [120]. This technique also partitions the training set into several randomly picked subsets. On each of the these subsets an SVM is trained. Using a probabilistic output of the SVM, the combination scheme is based on the covariance of the test data. In this way this method can be seen as a *transductive* method. The disadvantage of this is that it cannot operate on a single data point and the whole test set should be known in advance.

The idea originates from the method of coupled local minimizers and coupled training processes that has been proposed by Suykens *et al.* in [134]. There the coupling is done by means of coupling constraints. In this section the outputs of individual models are coupled on a chosen set of data points, the coupling set. Further links and motivations based on multitask learning [18] will be shown. This technique is then applied to the training of kernel models that have a similar form as Gaussian Processes [81], Kriging ([73],[27]), RBF networks or Regularization Networks [44],[109], but consider the model and training in a parameterized way. The method is illustrated on a number of examples.

6.7.2 Parameterized kernel methods

In this part we consider true models of the form $\mathbf{y} = f^*(\mathbf{x}) + \mathbf{e}$ where $E[\mathbf{e}] = 0$, $E[\mathbf{e}\mathbf{e}^T] = I\sigma^2$ and $\sigma < \infty$. Based on a training set $\mathcal{D} =$

$\{(\mathbf{x}_i,y_i)\}_{i=1}^{N} \subset \mathbb{R}^d \times \mathbb{R}$ a model $f(\mathbf{x}) = f(\mathbf{x};\mathcal{D})$ is estimated. Since we will focus on *ensemble learning*, this estimate $f(\mathbf{x})$ results from combining the elements of a population of q estimators $\{f^{(j)}\}_{j=1}^{q}$.

In order to construct our population of submodels, *parameterized kernel models* are considered. Let us take ensembles of estimators $\{f^{(j)}\}_{j=1}^{q}$ where the individual models have the form

$$f^{(j)}(\mathbf{x}) = \sum_{p=1}^{g_j} \alpha_p^{(j)} k(\mathbf{x}_p^{(j)}, \mathbf{x}), \tag{6.21}$$

where $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} : (\mathbf{x}, \mathbf{x}') \mapsto k(\mathbf{x}, \mathbf{x}')$. Unlike standard SVMs where k should be a Mercer kernel (positive definite kernel), this is not the case here as we consider parameterized kernel models. The kernel (or Gram) matrix constructed over the data set $\mathcal{D}^{(j)}$ of size g_j will be indicated by $K^{(j)}$. Each of the elements of these matrices are denoted by $K_{lm}^{(j)} = k(\mathbf{x}_l^{(j)}, \mathbf{x}_m^{(j)})$, $l, m \in \{1, \ldots, g_j\}$ where $(\mathbf{x}_l^{(j)}, y_l^{(j)}) \in \mathcal{D}^{(j)}$. The unknown parameters of each model $f^{(j)}$ are $\alpha^{(j)} = [\alpha_1^{(j)} \ldots \alpha_{g_j}^{(j)}]^T$.

The estimation of these parameters is based on the least squares loss function with regularization (ridge regression). This leads to the following cost function $U^{(j)}(\alpha^{(j)})$ corresponding to the training of the submodels ([109])

$$\min_{\alpha^{(j)}} U^{(j)}\left(\alpha^{(j)}\right) = \min_{\alpha^{(j)}} \quad \frac{1}{2}\left\|K^{(j)}\alpha^{(j)} - y^{(j)}\right\|_2^2 + \frac{1}{2\gamma}\left\|\alpha^{(j)}\right\|_2^2. \tag{6.22}$$

The influence of the regularization term is controlled by the hyperparameter γ. The solution of this convex cost function can be found by setting $\frac{\partial U^{(j)}(\alpha^{(j)})}{\partial \alpha^{(j)}} = 0$. This results into a linear system for each individual submodel:

$$(K^{(j)T}K^{(j)} + \frac{1}{\gamma} I_g)\alpha^{(j)} = K^{(j)T}\mathbf{y}^{(j)}, \tag{6.23}$$

where the output variables of the training data points of subset $\mathcal{D}^{(j)}$ are contained in the vector $\mathbf{y}^{(j)}$. Although the kernel used is not necessarily positive definite, the solution of this linear system is unique as long as the matrix is full rank, which can be ensured through the regularization term (note that $K^{(j)T}K^{(j)}$ is positive semi definite).

6.7.3 Uncoupled ensembles and committee networks

To combine the elements of the ensemble we will use *weighted averaging methods* (or *committee networks* as discussed e.g. in [7])

$$f(\mathbf{x},\mathcal{D}) = \sum_{j=1}^{q} \beta^{(j)} f^{(j)}(\mathbf{x}). \tag{6.24}$$

We simplify the notation as $f^{(j)}(\mathbf{x}) = f^{(j)}(\mathbf{x};\mathcal{D}^{(j)})$. To avoid that the whole data set $\mathcal{D}$ has to be learned by only one single model, we consider training the different submodels on separate subsets of the original data set. Therefore we divide $\mathcal{D}$ into q subsets, each of size g_j. Each of the submodels $f^{(j)}$ is then trained on one of the mutually disjoint subsets $\mathcal{D}^{(j)} = \{(\mathbf{x}_p^{(j)}, y_p^{(j)})\}_{p=1}^{g_j} \subset \mathcal{D}$, for $j \in \{1,\ldots,q\}$. This training on the subsets introduces a variance on the performance of these different models, which is reduced by taking a weighted average of the individual models. This variance reduction capability was already proven by [12] on neural networks and decision tree models.

In the previous section we explained a method for constructing the individual learning algorithms of our ensemble. But how should we interpret this for the whole ensemble? The training of the ensemble corresponds to minimizing a cost function U_e, which is the sum of the individual cost functions

$$U_e(\alpha^{(1)},\ldots,\alpha^{(q)}) = \sum_{j=1}^{q} U^{(j)}(\alpha^{(j)}). \tag{6.25}$$

Setting $\frac{\partial U_e}{\partial \alpha^{(j)}} = 0, \forall j$ gives the solution

$$\begin{bmatrix} H^{(1)} & 0 & \cdots & 0 \\ 0 & H^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & H^{(q)} \end{bmatrix} \begin{bmatrix} \alpha^{(1)} \\ \alpha^{(2)} \\ \vdots \\ \alpha^{(q)} \end{bmatrix} = \begin{bmatrix} K^{(1)T} y^{(1)} \\ K^{(2)T} y^{(2)} \\ \vdots \\ K^{(q)T} y^{(q)} \end{bmatrix}, \tag{6.26}$$

where $H^{(j)} = K^{(j)T} K^{(j)} + \frac{1}{\gamma} I_{g_j}$ for $j = 1,\ldots,q$. Note that the matrix of this linear system is a block-diagonal matrix where each individual subsystem is of the form (6.23). Hence there is no coupling between the learning processes for the parameter vectors $\alpha^{(j)}$. The training of an individual model on a data set $\mathcal{D}^{(j)}$ can be seen as the learning of one single task. Therefore, we call them *Single Task Learners*. This concept of Single Task Learners originates

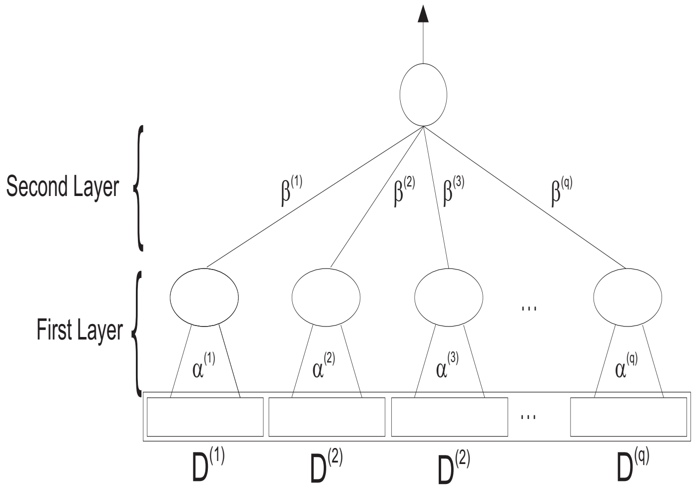

Figure 6.6: Architecture of the Uncoupled Ensemble Model.

from [18]. Ensembles of Single Task Learners found via this training are called here *Uncoupled Ensembles*.

After finding the optimal parameter vectors of the individual submodels, the overall committee model, based on (6.24) becomes

$$\begin{aligned} f(\mathbf{x}) &= \sum_{j=1}^{q} \beta^{(j)} f^{(j)}(\mathbf{x}) \\ &= \sum_{j=1}^{q} \beta^{(j)} \sum_{p=1}^{g_j} \alpha_p^{(j)} k(\mathbf{x}_p^{(j)}, \mathbf{x}). \end{aligned} \tag{6.27}$$

From this expression one can clearly see that the overall models consist of two layers, as shown in Figure 6.6. The first layer is parameterized in terms of $\{\alpha^{(j)}\}_{j=1}^{q}$. The second layer is parameterized by $\{\beta^{(j)}\}_{j=1}^{q}$.

The optimal weights of the second layer can be determined e.g. according to committee networks ([106], [7]) where the individual error functions are defined as $\epsilon^{(j)}(\mathbf{x}) = f^{(j)}(\mathbf{x}) - f^*(\mathbf{x})$ and the committee network takes the form

$$f(\mathbf{x}) = f^*(\mathbf{x}) + \sum_{j=1}^{q} \beta^{(j)} \epsilon^{(j)}(\mathbf{x}), \tag{6.28}$$

where one assumes that $\sum_{j=1}^{q} \beta^{(j)} = 1$. This leads to an ensemble model that is an affine combination of the submodels. The cost function considered for the committee network is

$$\begin{aligned} J &= E\left[\left(f\left(\mathbf{x}\right) - f^*\left(\mathbf{x}\right)\right)^2\right] \\ &= E\left[\left(\sum_{j=1}^{q} \beta^{(j)} \epsilon^{(j)}\left(\mathbf{x}\right)\right)\left(\sum_{j=1}^{q} \beta^{(j)} \epsilon^{(j)}\left(\mathbf{x}\right)\right)\right] \\ &\simeq \beta^T S \beta, \end{aligned} \tag{6.29}$$

where S denotes the error covariance matrix with $s_{ij} = E\left[\epsilon^{(i)}\left(\mathbf{x}\right)\epsilon^{(j)}\left(\mathbf{x}\right)\right]$, $\beta = [\beta^{(1)}; \ldots; \beta^{(q)}]$ and $E[\cdot]$ the expected value. This error covariance matrix can be computed by making use of a finite-sample approximation of $S = [s_{ij}] \in \mathbb{R}^{q \times q}$ based on a data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{p}$ of size p such that

$$s_{ij} \approx \frac{1}{p+1} \sum_{j=1}^{p} \left(f\left(\mathbf{x}_i\right) - y_i\right)\left(f\left(\mathbf{x}_j\right) - y_j\right). \tag{6.30}$$

This data set on which the error covariance matrix is built is typically a subset of the original training set.

An optimal choice of the parameters β can be found by minimizing

$$\left\{ \begin{array}{ll} \min\limits_{\beta} & \beta^T S \beta \\ & \text{s.t.} \sum_{j=1}^{q} \beta^{(j)} = 1. \end{array} \right. \tag{6.31}$$

The solution follows from the Lagrangian

$$\mathcal{L}\left(\beta, \lambda\right) = \frac{1}{2}\beta^T S \beta - \lambda \left(\sum_{j=1}^{q} \beta^{(j)} - 1\right) \tag{6.32}$$

with Lagrange multiplier λ. The conditions for optimality are given by

$$\left\{ \begin{array}{l} \frac{\partial \mathcal{L}(\beta,\lambda)}{\partial \beta} = S\beta - \lambda \mathbf{1}_q = 0, \\ \frac{\partial \mathcal{L}(\beta,\lambda)}{\partial \lambda} = \mathbf{1}_q^T \beta - 1 = 0 \end{array} \right. \tag{6.33}$$

with optimal solution

$$\beta = \frac{S^{-1}\mathbf{1}_q}{\mathbf{1}_q^T S^{-1} \mathbf{1}_q} \tag{6.34}$$

and corresponding error $J = \left(\mathbf{1}_q^T S^{-1} \mathbf{1}_q\right)^{-1}$.

6.7.4 Ensemble learning using a coupling set

In relation to the literature on multitask learning ([18]), one may interpret the learning of the different submodels on disjoint data sets as different tasks that have to be learned. The fact that each model $f^{(j)}$ is trained on its individual data set $\mathcal{D}^{(j)}$ can be interpreted as if it is trained to perform a specific task induced by this data set. Therefore it is called *Single Task Learning* ([18]).

Multitask learning was first introduced by [18], who showed that when learning algorithms are trained on different but related tasks, they share knowledge and are able to learn from each other. The central idea of multitask learning is sharing what is learned by different tasks while these tasks are trained in *parallel*. This is explained via the concept of *inductive bias*.

An inductive bias is anything that causes an inductive learning algorithm to prefer some set of parameters, or hypotheses, over another hypothesis. When the training of a model on a certain task is influenced by tasks other than the main task, these other tasks may serve as bias. This information exchange between the different models results into an improved generalization performance. Caruana tested this idea on neural networks (MLPs) that share the hidden layer for different tasks. But in the end, the goal was to test the algorithm on its main task.

The idea to use this approach on ensemble models was investigated by [1] where the goal is to train different models on the same task and the knowledge sharing is achieved through model clustering. An ensemble model is created on top of these submodels found by the multitask learning scheme.

The central aspect in the multitask learning is the information exchange between the different models. This causes that the generalization performance increases. In our approach we induce this information exchange by coupling of the different submodels. This idea is inspired on the idea of Coupled Local Minimizers (CLM) introduced in ([134]), where the strategy has been proposed as a new way for solving non-convex optimization problems. By coupling of the different trajectories of a multi-start optimization method, very good local minima of the non-convex optimization problem are found in an efficient way. In certain ill-conditioned problems it has been illustrated that the coupling mechanism may serve as a regularization mechanism by itself (without having any regularization term in the cost function). During the optimization process there is a continuous information exchange between the different multi-start optimizers through the coupling (as illustrated for state vector synchronization of search and learning processes). This methodology was applied to the training of neural networks. The CLM approach improved the generalization performances of the neural networks even without regularization. The coupling between the models acts as an information exchanging procedure. In this way the coupling acts as a collective intelligence among the different models. Notice that the training of the different models is done in parallel. It is this idea of coupled learning models that is further studied in the following subsections.

Ensemble learning through coupling

As explained in the previous section, it is our goal to increase the generalization performance of the ensemble model through coupling of the individual learning processes of the submodels. Since an ensemble model is constructed on top of the collection of submodels $\{f^{(j)}\}_{j=1}^{q}$ the performance of the ensemble model may increase if the individual submodels have a better performance on their individual task. Instead of constructing an ensemble model out of Single Task Learners, we now construct an ensemble with coupling of the training processes of the submodels. Since each of the individual models $f^{(j)}$ is trained on its individual data set $\mathcal{D}^{(j)}$, this can be understood as different tasks that have to be learned. The coupling between the individual models will result into a multitask learning scheme. How can we couple the models?

During the parallel training of the individual models, a distance measure between the models will be monitored. Based on this distance measure, corrections on the minimization of the individual cost functions are made. Our experience about a good distance measure between the models is in correspondence with the results of [1]. The 'natural' distance function based on the model outputs rather than a measure based on the model parameters should be used to couple the submodels. Each of the q submodels is trained by minimizing a cost function. Since training sets $\mathcal{D}^{(j)}$ are all sampled from the same distribution, we expect the submodels to give similar results on the same input $\mathbf{x}$. This can be used as an extra coupling between the submodels. We will use this constraint as an extra *output coupling* between the models on a predefined *coupling set* $\mathcal{D}_c = \{\mathbf{x}_i^c\}_{i=1}^{N_c}$ of size N_c. This means that during training we will demand that the neighboring models are found with as additional objective

$$\min_{\mathbf{x}_i^c \in \mathcal{D}_c} \quad \sum_{i=1}^{Nc} \left\| f^{(j)}\left(\mathbf{x}_i^c\right) - f^{(j+1)}\left(\mathbf{x}_i^c\right) \right\|_2^2 \; \forall j \in \{1 \ldots q\}. \tag{6.35}$$

Notice that we don't need the output values of the coupling points. This means that we have the freedom to chose the coupling set points. We consider three schemes of coupling sets $\mathcal{D}_c$:

- the coupling set $\mathcal{D}_c$ is a subset of the training set,
- the coupling set $\mathcal{D}_c$ is a randomly generated in the input space,
- the coupling set $\mathcal{D}_c$ is the test set. In some applications we have the test set in advance. We can use this information during the training without actually knowing the output value. In this way the coupling set will act as Transductive Learning set. By doing this we achieve a way of *Transductive Learning* ([51],[147],[21],[69],[121]).

Parameterized kernel models with coupled learning processes

When we tested the idea of coupled learning on the parameterized kernel models, it turned out the models as defined in (6.7.2) are too stringent. In this way the coupling between the models only has minor effects. In order to construct models with more degrees of freedom, we consider an *overparameterization* of our individual submodels. Each submodel j is constructed based on a Gram matrix on the corresponding training set $\mathcal{D}^{(j)}$ and on the *adjoining subsets* $\mathcal{D}^{(j-1)}$ and $\mathcal{D}^{(j+1)}$. The submodels have the form

$$f^{(j)}\left(\mathbf{x};\tilde{\alpha}^{(j)}\right)=\tilde{\alpha}^{(j)T}\tilde{\mathbf{k}}^{(j)}\left(\mathbf{x}\right), \tag{6.36}$$

where

$$\begin{aligned}\tilde{\mathbf{k}}^{(j)}(x)=[&k(x_1^{(j-1)},x)\dots k(x_g^{(j-1)},x)\\&k(x_1^{(j)},x)\dots k(x_g^{(j)},x)k(x_1^{(j+1)},x)\dots k(x_g^{(j+1)},x)]^T\end{aligned} \tag{6.37}$$

and $\tilde{\alpha}^{(j)}\in\mathbb{R}^{3g\times 1}$ (without loss of generality we consider here $g=g_1=\ldots=g_q$). This is done e.g. in a circular way, such that $\mathbf{x}_i^{(0)}=\mathbf{x}_i^{(q)}$ and $\mathbf{x}_i^{(q+1)}=\mathbf{x}_i^{(1)}$.

For each subset $\mathcal{D}^{(j)}$ we consider now minimization of

$$\min_{\alpha^{(j)}}\ \frac{1}{2}\left\|\tilde{K}^{(j)}\tilde{\alpha}^{(j)}-y^{(j)}\right\|_2^2+\frac{1}{2\gamma}\left\|\tilde{\alpha}^{(j)}\right\|_2^2, \tag{6.38}$$

where $\tilde{K}^{(j)}=[K^{(j,j-1)}\ K^{(j,j)}\ K^{(j,j+1)}]$ and $K^{(i,j)}\in\mathbb{R}^{g\times g}$ is a kernel matrix built on the subsets $\mathcal{D}^{(i)}$ and $\mathcal{D}^{(j)}$ defined by $K_{kl}^{(i,j)}=k(\mathbf{x}_k^{(i)},\mathbf{x}_l^{(j)})$, $k,l\in\{1,\ldots,g\}$ and $(\mathbf{x}_k^{(i)},y_k^{(i)})\in\mathcal{D}^{(i)}$ and $(\mathbf{x}_l^{(j)},y_l^{(j)})\in\mathcal{D}^{(j)}$. We see that similar

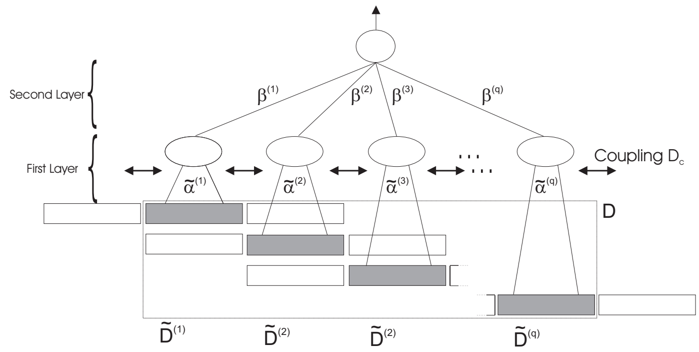

Figure 6.7: Architecture of the Coupled Ensemble Model.

to (6.22), this cost function is also a combination of a squared loss error minimization and a regularization term, which can be controlled by the parameter γ. The final architecture has a similar two layered structure and is shown in Figure 6.7.

The training process

As explained in the previous section we will use the idea of Coupled Local Minimizers (CLM) to achieve multitask learning via coupling. The coupling of the different submodels is done by imposing an extra constraint in the optimization process. In Section 6.7.3 we explained how an ensemble of Single Task Learners can be trained by minimizing the sum of the individual cost functions. The CLM method suggests now to couple these individual models by imposing additional coupling constraints in the optimization process. An important point hereby is that we don't need the output values at the coupling points. In the same way as given in Section 6.7.3, we now have the ability to create an ensemble of these models with a circular coupling via

$$\left\{\begin{array}{l} \min\limits_{\tilde{\alpha}^{(j)},e_i^{(j)}} \frac{1}{2}\sum_{j=1}^{q}\left\|\tilde{K}^{(j)}\tilde{\alpha}^{(j)}-y^{(j)}\right\|_2^2+\frac{1}{2\gamma}\left\|\tilde{\alpha}^{(j)}\right\|_2^2+\frac{\nu}{2}\sum_{j=1}^{q}\sum_{i=1}^{N_c}\left(e_i^{(j)}\right)^2 \\ \quad\text{s.t. } e_i^{(j)}=\tilde{\alpha}^{(j)T}\tilde{\mathbf{k}}^{(j)}\left(\mathbf{x}_i^c\right)-\tilde{\alpha}^{(j+1)T}\tilde{\mathbf{k}}^{(j+1)}\left(\mathbf{x}_i^c\right), \\ \qquad x_i^c\in\mathcal{D}_c;\ \forall i\in\{1,\ldots,N_c\};\ \forall j\in\{1,\ldots,q\}. \end{array}\right. \tag{6.39}$$

The coupling strength can be adjusted by the coupling parameter ν. The constraints are added to the problem in a similar fashion as in least squares SVMs (see Chapter 2). If we put $\nu = 0$ we end up with a model similar to (6.7.3). When $\nu > 0$ a synchronization between the models on the predefined coupling set is achieved. Note that if $\nu < 0$ the models will be de-synchronized. This last option should be used with care since it can violate the convexity of the cost function. Depending on the value of ν, the matrix involved in the resulting linear system may become singular.

As mentioned, in the case that $\nu > 0$ the optimization problem has a unique solution due to its convexity. This is an important property compared to the results on multitask learning with Neural Networks where one typically has a non-convex optimization problem. The solution of the convex problem can be found via the Karush-Kuhn-Tucker optimality conditions. Therefore we take the Lagrangian

$$\begin{aligned}\mathcal{L}\left(\tilde{\alpha}^{(j)}, e_i^{(j)}, \mu_i^{(j)}\right) &= \frac{1}{2}\sum_{j=1}^{q}\left\|\tilde{K}^{(j)}\tilde{\alpha}^{(j)} - y^{(j)}\right\|_2^2 + \frac{1}{2\gamma}\left\|\tilde{\alpha}^{(j)}\right\|_2^2 \\ &+ \sum_{j=1}^{q}\sum_{i=1}^{N_c}\left(\frac{\nu}{2}\left(e_i^{(j)}\right)^2 - \mu_i^{(j)}\left\{\tilde{\alpha}^{(j)^T}\tilde{\mathbf{k}}^{(j)}(\mathbf{x}_i^c) - \tilde{\alpha}^{(j+1)^T}\tilde{\mathbf{k}}^{(j)}(\mathbf{x}_i^c) - e_i^j\right\}\right).\end{aligned} \tag{6.40}$$

The Karush-Kuhn-Tucker conditions are given by

$$\left\{\begin{array}{l} \frac{\partial\mathcal{L}}{\partial\tilde{\alpha}^{(j)}} = \left(\tilde{K}^{(j)T}\tilde{K}^{(j)} + \frac{1}{\gamma}I_{3g}\right)\tilde{\alpha}^{(j)} - 2y^{(j)}\tilde{K}^{(j)} \\ \qquad + \sum_{i=1}^{N_c}\mu_i^{(j-1)}\mathbf{k}^{(j)}\left(\mathbf{x}_i^c\right) - \sum_{i=1}^{N_c}\mu_i^{(j)}\mathbf{k}^{(j)}\left(\mathbf{x}_i^c\right) = 0,\ \forall j \in \{1,\ldots,q\}, \\ \frac{\partial\mathcal{L}}{\partial e_i^{(j)}} = \nu e_i^{(j)} + \mu_i^{(j)} = 0,\ \forall j \in \{1,\ldots,q\},\ \forall i \in \{1,\ldots,N_c\}, \\ \frac{\partial\mathcal{L}}{\partial\mu_i^{(j)}} = \alpha^{(j)^T}\tilde{\mathbf{k}}^{(j)}\left(\mathbf{x}_i^c\right) - \alpha^{(j+1)^T}\tilde{\mathbf{k}}^{(j+1)}\left(\mathbf{x}_i^c\right) - e_i^{(j)} = 0,\ \forall i \in \{1,\ldots,N_c\}. \end{array}\right. \tag{6.41}$$

After some simple matrix algebra this can be rewritten as

$$\begin{aligned}\left(\tilde{K}^{(j)T}\tilde{K}^{(j)} + \frac{1}{\gamma}I_{3g} + \nu G^{(j,j)}\right)\tilde{\alpha}^{(j)} - \nu G^{(j,j-1)}\tilde{\alpha}^{(j-1)} - \nu G^{(j,j+1)}\tilde{\alpha}^{(j+1)} \\ = 2\tilde{K}^{(j)T}y^{(j)},\ \forall j \in \{1,\ldots,q\},\end{aligned} \tag{6.42}$$

with $G^{(q,w)} = \sum_{i=1}^{N_c}\tilde{\mathbf{k}}^{(q)}(\mathbf{x}_i^c)\tilde{\mathbf{k}}^{(w)}(\mathbf{x}_i^c)^T$.

In order to show the links with (6.26) we rewrite this as

$$\begin{bmatrix} H^{(1)} & -\nu G^{(1,2)} & 0 & \cdots & 0 & -\nu G^{(1,q)} \\ -\nu G^{(2,1)} & H^{(2)} & -\nu G^{(2,3)} & & & 0 \\ 0 & -\nu G^{(3,2)} & H^{(3)} & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & & & \ddots & \ddots & -\nu G^{(q-1,q)} \\ -\nu G^{(q,1)} & 0 & \cdots & 0 & -\nu G^{(q,q-1)} & H^{(q)} \end{bmatrix} \begin{bmatrix} \tilde{\alpha}^{(1)} \\ \tilde{\alpha}^{(2)} \\ \vdots \\ \\ \tilde{\alpha}^{(q)} \end{bmatrix} \tag{6.43}$$

$$= \begin{bmatrix} 2\tilde{K}^{(1)T} y^{(1)} \\ 2\tilde{K}^{(2)T} y^{(2)} \\ \vdots \\ \\ 2\tilde{K}^{(q)T} y^{(q)} \end{bmatrix},$$

where $H^{(j)} = \tilde{K}^{(j)T}\tilde{K}^{(j)} + \frac{1}{\gamma} I_{3g} + \nu G^{(j,j)}$ for $\forall j \in \{1, \ldots, q\}$. The effect of the coupling manifests itself in two ways. The most noticeable are the off-diagonal blocks. These represent the coupling between the neighboring submodels. A second effect is the extra term added to the diagonal elements of the matrix. This term acts as an additional regularization term. In this way we can see that $H^{(j)}$ consists of two regularization terms. The first term I_{3g}/γ is an individual regularization for the submodel. This was also present in the uncoupled ensemble. The second term $\nu G^{(j,j)}$ is an effect of the coupling and can be seen as a *group regularization* caused by the coupling.

Towards large data sets

In the introduction we already mentioned that one of the goals of ensemble methods is that one can reduce the computational and memory complexity of the algorithms for large data sets. Since both models consist of two separate layers that are computed separately, will we handle them apart.

The memory complexity for the training of the first layer of the uncoupled ensemble model is $\mathcal{O}(N^2/q)$, which is clear from (6.26). But since each of the parameters $\alpha^{(j)}$ can be computed separately, the memory complexity drops to $\mathcal{O}(N^2/q^2)$.

For the first layer of the coupled ensemble models, the memory complexity increased due to the overparameterization and the coupling. The memory complexity of this model is $\mathcal{O}(27N^2/q)$. This can be seen as follows. Each of the block matrices in the coupled ensemble formulation has a memory storage $\mathcal{O}((3N/q)^2)$ where N is the number of data points and q the number of submodels. The factor 3 originates from the overparameterization of the models. Since we have q block diagonal elements and $2q$ block off-diagonal elements, this adds up to a memory complexity of $\mathcal{O}(3q(3N/q)^2) = \mathcal{O}(27N^2/q)$. For large data sets the structure and sparseness of this large system (which is positive definite) can be exploited. One sees that if $q \geq 27$ the memory complexity of the coupled ensemble model is lower than other kernel model like RBF, SVM or LS-SVM, which have a memory complexity [2] of $\mathcal{O}(N^2)$.

A second solution is to transform the linear system into a block tridiagonal matrix. For this purpose, we eliminate here the coupling between the outer models. On tests this effect did not influence on the performance. The solution of the block tridiagonal system can be found by a *block LU factorization*. Since the matrices $H^{(i)}$ are all non-singular[3], this LU factorization is proven to converge ([57]). As will be shown, this divides the original problem into smaller sub-problems.

In general the *block LU factorization* is defined as follows. Let a linear system $Ax = b$ to be solved have the following block tridiagonal form:

$$\begin{bmatrix} A_1 & C_1 & 0 & \cdots & 0 \\ B_1 & A_2 & C_2 & 0 & \vdots \\ 0 & B_2 & A_3 & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & C_{q-1} \\ 0 & \cdots & 0 & B_{q-1} & A_q \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \\ x_q \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \\ b_q \end{bmatrix}. \tag{6.44}$$

According to [57] the LU-factorization of this matrix A is given by

$$A = \begin{bmatrix} I & 0 & & \cdots & 0 \\ L_1 & I & 0 & & \vdots \\ 0 & L_2 & I & \ddots & \\ \vdots & 0 & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & L_{q-1} & I \end{bmatrix} \begin{bmatrix} U_1 & C_1 & 0 & \cdots & 0 \\ 0 & U_2 & C_2 & 0 & \vdots \\ & 0 & U_3 & \ddots & 0 \\ \vdots & & \ddots & \ddots & C_{q-1} \\ 0 & \cdots & & 0 & U_q \end{bmatrix}. \tag{6.45}$$

[2]This memory complexity holds for direct methods. Better memory efficiency can be achieved by using iterative methods.

[3]If $\nu > 0$ then the matrix $H^{(j)}$ is a positive definite matrix since $\tilde{K}^{(j)^T}\tilde{K}^{(j)} + \frac{1}{\gamma}I$ is positive definite and $G^{(j,j)}$ which is a sum of rank one matrices, is also positive definite.

The unknown submatrices L_i and U_i can be computed via the following iterative process:

$U_1 = A_1$
for $i = 2 : q$
$\quad L_{i-1} = B_{i-1}U_{i-1}^{-1}$
$\quad U_i = A_i - L_{i-1}C_{i-1}$
end

The procedure is defined as long as the matrices U_i are non-singular. After this factorization, the solution of the linear system follows from forward and backward substitution:

$y_1 = b_1$
for $i = 2 : 1 : q$
$\quad y_i = b_i - L_{i-1}y_{i-1}$
end
$x_q = U_q^{-1}y_q$
for $i = q-1 : -1 : 1$
$\quad x_i = U_i^{-1}(y_i - C_i x_{i+1})$
end

Note that all the matrix inversions can be avoided by rewriting them into linear systems that can be solved by appropriate methods.

At the level of the second layer also improvements for large data sets are possible. As explained in Section 6.7.3, the optimal weights $\{\beta^{(j)}\}_{j=1}^{q}$ can be found by solving (6.31). The solution to this problem is in most cases non-sparse which means that all the values of $\beta^{(j)}$ are non-zero. However, by imposing extra constraints $\beta^{(j)} \geq 0, \forall j$, one obtains ensemble models that are *convex* combinations of the submodels and that are trained by ([7], [106]):

$$\left\{\begin{array}{ll} \min\limits_{\beta} & \beta^T S \beta \\ \text{s.t.} & \sum_{j=1}^{q} \beta^{(j)} = 1, \\ & \beta^{(j)} \geq 0, \, \forall j \in \{1, \ldots, q\}. \end{array}\right. \tag{6.46}$$

This is a quadratic optimization problem of dimension q. Since the error covariance matrix is positive definite, the solution of this QP problem is global and unique. The advantage is that the solution now becomes sparse which simplifies the evaluation at new points

$$f(\mathbf{x}) = \sum_{j=1}^{\mathrm{SM}} \beta^{(j)} \sum_{p=1}^{g} \alpha_p^{(j)} k(\mathbf{x}_p^{(j)}, \mathbf{x}), \tag{6.47}$$

where SM denotes the number of $\beta^{(j)}$ output weights that are different from zero. Similar to the support vector machine literature ([147], [28]) these models can be viewed as *support models*. This sparseness sometimes also leads to a better performance as is illustrated in the next section.

6.7.5 Experiments

We illustrate coupled learning of Regularization Networks on examples of regression and classification.

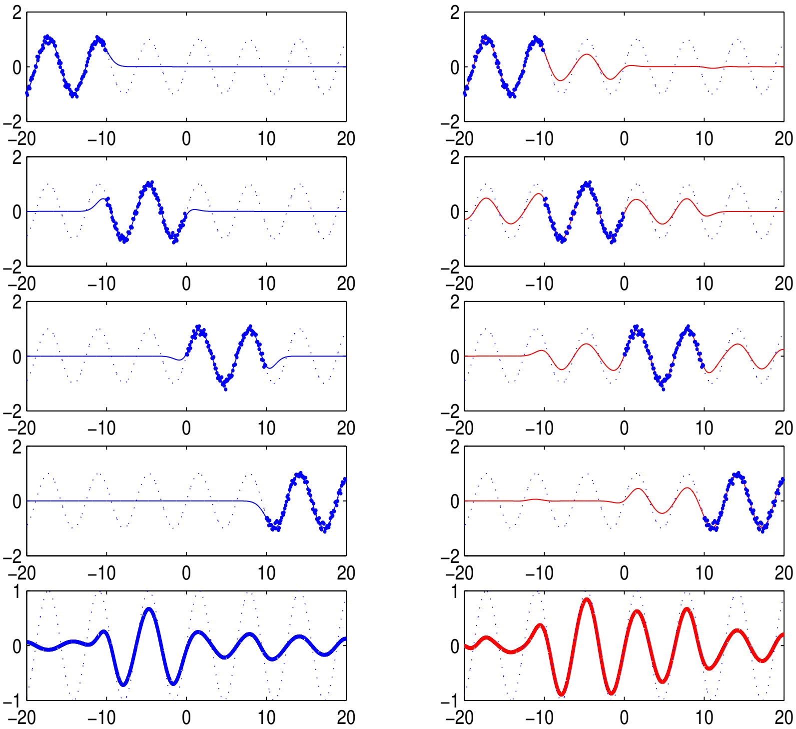

Figure 6.8: This figure shows 4 submodels of the uncoupled (left) and coupled (right) ensembles for a sine function (true function: dotted) with their corresponding training sets. The bottom figures illustrate the combined models of both ensembles (uncoupled (left); coupled (right)). This shows the effect and improvement obtained by coupling of the learning processes for the individual submodels.

Regression

Toy example: sine function In this example we give a comparison between models constructed via an uncoupled ensemble (as defined in Section 6.7.3) versus a coupled overparameterized ensemble (as defined in Section 6.7.4) on a sine function estimation problem. We did the training of the submodels on 4 disjoint subsets locally sampled in the input space. Although this local sampling of the input space is usually not possible in practice, we will use it here for demonstration purposes in order to visualize the effect of coupling. The coupling points are uniformly spread over the x-axis. Although subsets of different sizes can be taken, we assume here that the subsets are of equal size $g = N/q$. One can clearly see in Figure 6.8 that the individual models of the coupled overparameterized system have improved performances in the regions where they are not trained. Compared with the individual models of the uncoupled ensemble, the latter clearly shows a bad performance in regions where there are no training points.

Furthermore, we constructed a weighted combination based on these ensembles. We see that the performance of the coupled overparameterized models is better than for the uncoupled ensemble. This example illustrates that the coupling improves the generalization performance, which is caused by the information exchange between the different submodels during the learning process. The reason is that averaging of individual models that have better generalization performance will probably lead to a better performance of the ensemble model.

Boston housing data set The Boston housing data set is a multivariate regression data set of 506 cases in 14 attributes. It has two prototasks: NOX, in which the nitrous oxide level is to be predicted, and price MEDV, in which the median value of a home is to be predicted. In these tests we do a regression on the data sets with the models constructed via a simple ensemble and via a coupled overparameterized ensemble on randomly sampled subsets. We trained on 400 training points and use the remaining 106 points as test set and this for different randomizations.

As coupling set we use either 10 % of the total training data, randomly chosen, without the y-values or we use the test set as coupling set. In the latter case we have a transductive learning scheme. In both cases we used the Gaussian radial basis function kernel. The used hyperparameters are found by cross-validation for the individual models. For the NOX we used the hyperparameters $(\gamma, \sigma^2) = (20.67, 15.44)$ and for the MEDV $(\gamma, \sigma^2) = (81.19, 12.19)$. The coupling parameter is kept constant at $\nu = 1$.

To test the difference in performance between the methods, we computed the mean squared error (MSE) on a test set of both the MEDV and NOX estimations. We compared the distributions for 100 randomizations (see Figure 6.9) based on a Wilcoxon ranksum test. This indicates a statistically significantly difference between the averages of the two groups.

The results are summarized in Table 6.1. In this table we show the MSE performances on a test set. We use ensemble models based on collections or learning algorithms of size 8 or 16 (the number of models is indicated by the second number in Table 6.1, example NOX16 is an ensemble model based on 16 individual models on the NOX prediction). We give the mean (mea), median (mea) and variance (var) of these distributions after randomization. In the last two columns the result of the Wilcoxon ranksum test are given: h indicates the test that the two distributions are different and p is the corresponding p-value. If h equals 1 there is a statistically significant difference with 95% probability.

From these results we can conclude that the coupling of the models with transductive inference leads to statistically significantly better performance in all the shown cases. If the coupling set is randomly chosen, the performances are not significantly better for the coupled ensembles. But notice that only in one of the shown tests, the NOX prediction with 16 models, the performance of the coupled ensemble is worse.

In Figure 6.9 we show the distributions of the MSE performances of ensemble models created on 8 individual models for both the NOX prediction as the MEDV. One sees the performance of the uncoupled and coupled ensembles. Notice that besides the fact that the mean MSE of the coupled ensemble is always better, also the variance is smaller. We observe that this property holds for the majority of experiments in Table 6.1. An intuitive explanation of this behavior is the following.

By coupling of the submodels within the ensemble, we prevent the fact that some of the models give a very bad prediction. These *outlier models* can have a very disturbing influence on the overall performance of the averaging models. This is indicated by a big variance. In statistics it is also known that taking an average is very sensitive to outliers. By coupling of the individual models, we prevent that some of the models of the ensemble have a performance that deviates too much from the other models. As long as the majority of the elements of the ensemble give good predictions, they will correct the outlier models. As we can notice this results into a smaller variance.

Although it is not our purpose in this thesis to make an extensive benchmarking with other learning models, we have also tested the performance of

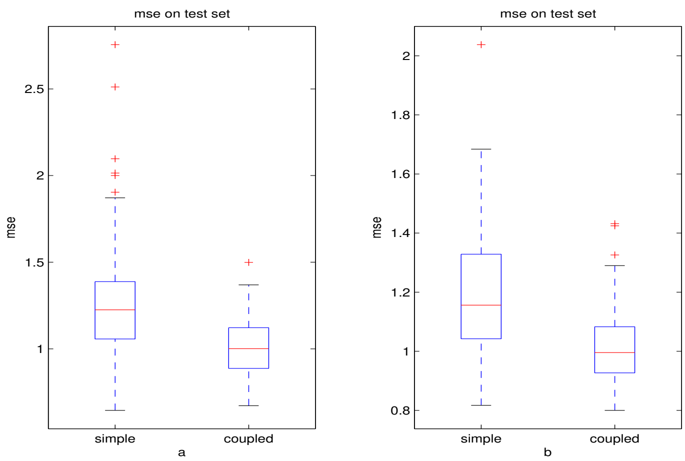

Figure 6.9: This figure shows the distribution of the mse performances on the Boston Housing data set of uncoupled (left boxplots) and coupled (right boxplots) ensemble models after 100 randomizations. Part *a* shows the performance on the NOX prediction and part *b* shows the performance on the MEDV prediction. In both cases we see the improvement of the coupled ensembles with a lower mean and a smaller variance. These experiments are done for a fixed value of $\nu = 1$.

	Simple Ensemble			Coupled Ensemble				
	mea	med	va	mea	med	va	h	p
NOX,8	1.20	1.16	4.76e-2	1.31	1.20	6.78e-2	0	6.74e-2
NOX,8,T	1.12	1.16	4.49e-2	1.01	1.00	1.33e-2	1	8.82e-11
NOX,16	1.92	1.59	2.85e-2	2.49	1.78	7.17e-2	1	3.20e-2
NOX,16,T	2.03	1.61	7.24e-2	1.05	1.02	1.74e-2	1	0
MEDV,8	1.30	1.21	7.20e-2	1.34	1.23	8.27e-2	0	7.26e-1
MEDV,8,T	1.28	1.23	6.00e-2	0	1.00	3.02e-2	1	4.37e-11
MEDV,16	2.11	1.58	5.61e-1	2.23	1.55	4.71e-1	0	6.04e-1
MEDV,16,T	2.28	1.74	9.17e-1	1.14	1.10	4.41e-2	1	0
NOX, LS-SVM	1.84	1.76	1.19e-1					
MEDV, LS-SVM	1.33	1.31	6.08e-2					

Table 6.1: This table shows the mse performances (Boston housing data problem) on a test set. We use ensemble models based on collections of learning algorithms of either size 8 or 16 (the number of models is indicated by the second number in Table 1, example NOX16 is an ensemble model based on 16 individual models on the NOX prediction). We show the mean (mea), median (mea) and variance (var) of these distributions after 100 randomization. In the last two columns the result of a Wilcoxon ranksum test are given: h indicates the test that the two distributions are different and p is the corresponding p-value. In the last two rows the mse performance of a standard LS-SVM is given. Again the mean (mea), median (mea) and variance (var) of these distributions are given after 100 randomization. From these results we can conclude that the coupling of the models with transductive inference leads to statistically significantly better performance in all the shown cases. If the coupling set is randomly chosen, the performances are not significantly better for the coupled ensembles. But notice that only in one of the shown tests, the NOX prediction with 16 models, the performance of the coupled ensemble is worse.

a standard LS-SVM with the same hyperparameters (γ, σ^2) trained on the whole data set of 400 points and tested on a test set of 106 points. After 100 randomizations we get the results gives in the last two rows of Table 6.1. We observe that the ensemble methods that we use here, give performances comparable to the standard LS-SVM. For the coupled ensemble models with transductive inference, we notice that the performances are better.

The role of the coupling parameter In order to show the effect of the coupling parameter ν, we have done the following test. We computed the MSE on a test set for different values of ν while keeping the other hyperparameters constant. To overcome worst-case effects we randomized the test and training set 20 times for each value of ν.

As we explained previously, by the coupling parameter ν we can adjust the coupling strength. If we put $\nu = 0$ we end up with a model similar to the one defined in Section 6.7.3; $\nu > 0$ indicates a synchronization coupling between the models; $\nu < 0$ indicates that the models will be *de-synchronized*. This last option may cause numerical problems because the convexity of the problem formulation may be violated.

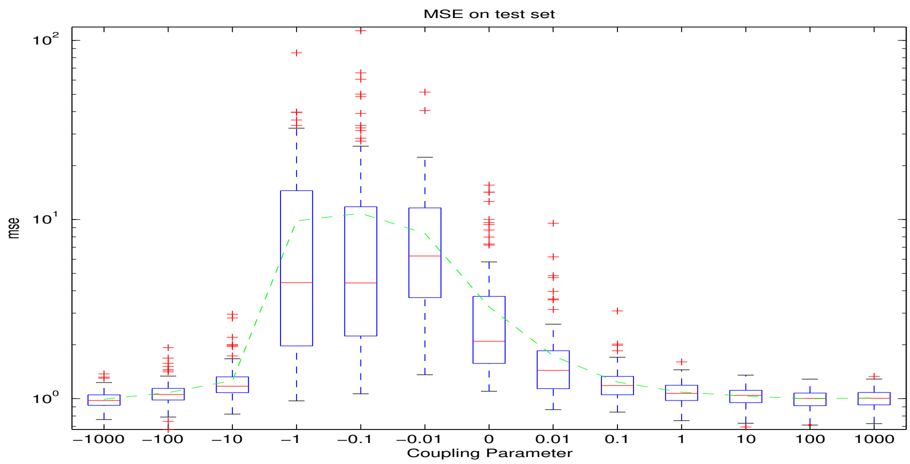

Figure 6.10: This figure shows the mse performance on a test set for the Boston Housing data set with different values of the coupling parameters ν, after randomization. The dotted line is a fitting between the means of each randomization experiment. The mse values are plotted on log-scale.

As we can observe in Figure 6.10, the coupling with $\nu > 0$ improves the performance of the ensemble models in comparison with the uncoupled situation. This indicates that the submodels are learning from each other. The inductive bias of the individual algorithms improves the overall performance. Notice also that there is a clear reduction in the variance of the models.

For small negative coupling $\nu < 0$, the experiment shows that the performance improves, not only in average performance but also in variance. Surprising is the behavior for larger negative values of ν. Apparently the performance can also improve when we the models are *de-synchronized.* We do not have a clear explanation for this behavior. But, we want to stress that high value of ν should be used carefully, as we will explain.

Remember that the main goal is to create averaging models based on individual models that are data driven. Each individual model is fitted to its corresponding data set. By synchronizing them, each of them is slightly modified by the information exchange, also mentioned as the inductive bias. Now, if we increase the size of ν, the coupling will gain on influence. In the limit, the coupled ensemble models will focus more on synchronizing the individual models than creating data driven submodels with a small correction achieved by coupling. Therefore, in the focus of this work, we advise the take the coupling not too large.

Classification

Although all the models discussed in the previous section are intended for regression tasks, we will also use them for classification. This approach works well as was shown for LS-SVMs (see [129]). Because the output of a classification task is a binary variable, we will have to adapt our models. Therefore we use for the evaluation of new inputs:

$$f\left(\mathbf{x}\right) = \operatorname{sign}\left(\sum_{j=1}^{\mathrm{SM}} \beta^{(j)} \operatorname{sign}\left(\sum_{p=1}^{g} \alpha_p^{(j)} k(\mathbf{x}_p^{(j)}, \mathbf{x})\right)\right). \tag{6.48}$$

Training of the first layer of the uncoupled as well as the coupled ensemble remains the same. But for the training of the second layer we will always use the sparse formulation (6.46), with number of support models $\mathrm{SM} \leq q$. This sparse solution also improves the performance as observed in the experiments.

Tic-tac-toe data set The Tic-Tac-Toe Endgame database is an UCI ([8]) data set contributed by Aha, encoding the complete set of possible board configurations at the end of the tic-tac-toe games. The target concept is "win for x" where "x" is assumed to have played first. The data set consists of 958 observations with 9 attributes. Note that the 16 records that contained missing values were removed from the data set. The first 638 observation were used as training set while the last 320 as the test set. This data set is known to be a separable by a highly nonlinear model. We employ the Gaussian RBF kernel with the hyperparameters $(\gamma, \sigma^2) = (9.46e1, 9.96)$. In all tests on this data set, we used an ensemble consisting of 11 submodels and the coupling set is chosen as test set (transductive setting).

As a first test we studied the effect of the coupling parameter ν in this classification setup. We randomized the test and training set 20 times for a whole range of values of ν. For each of those values we measure the misclassification rate on a test set. The result of this experiment is shown in Figure 6.11. In comparison with the regression case we now see a clear minimum in the region 0.01 and 1. Since this is for $\nu > 0$, this indicates that coupling as synchronization between the models has the best performance here.

If we now compare the uncoupled and the coupled ensemble models constructed based on 11 models, with $\nu = 0.1$ (corresponding to the minimum of the previous test) and with the same hyperparameters, we see the following results. There is a significant improvement of the coupled ensemble compared to the uncoupled ensemble. This again is shown via a Wilcoxon ranksum test after 100 randomizations. The mean, median and variance of these tests together with the p-value is given in Table 6.2. In a last test we show the performance of the ensemble model in relation to the individual submodels. Therefore we have taken one of the models as trained in the previous randomization experiment with the optimal hyperparameter and coupling parameter as indicated. We constructed the receiver operating characteristic (ROC) curves for the submodels of the ensemble together with the ROC curve of the ensemble model. In Figure 6.12 the results are shown where it is clear that the ensemble performance is better than the mean performance of the elements of the ensemble. If we compare the area-under-the-curve (AUC) of the ensemble model, we see that for the uncoupled ensemble the AUC is 0.77 with standard error $SE = 0.03$, for the coupled ensemble this is 0.86 with standard error $SE = 0.02$. This is an improvement in performance of nearly 10% for the coupled ensemble.

In a further study of this experiment we show the weights of the individual submodels of the ensemble as found from (6.46). In the previous chapter it was already explained that by using (6.46) a sparse solution for the second layer is obtained. In Figure 6.13 we show the values of the weights as a result of this optimization problem. In the uncoupled as well as the coupled

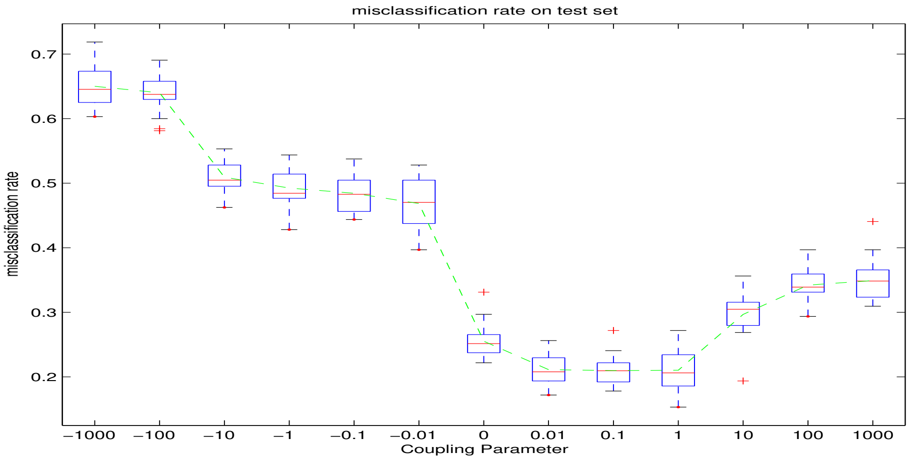

Figure 6.11: This figure shows the effect of the coupling parameter on the misclassification performance of the Tic-Tac-Toe classification example. For each of the values of ν we have done 20 randomizations. The dotted line is a fitting between the means of each randomization experiment.

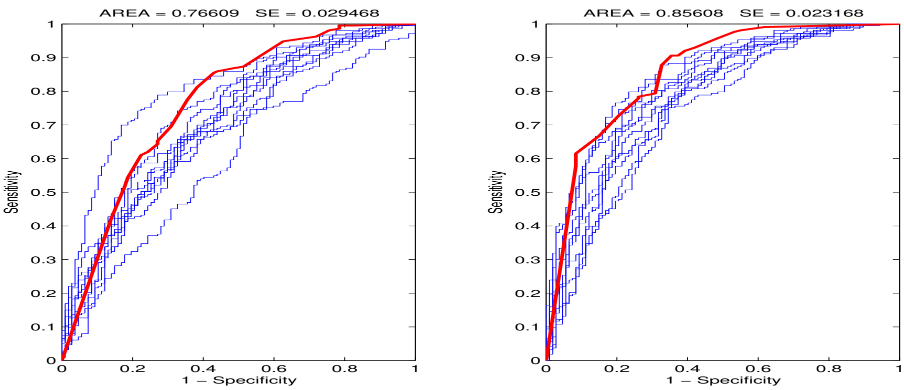

Figure 6.12: This figure shows the ROC performance of the Uncoupled (left) and the Coupled (right) ensemble on the Tic-Tac-Toe data set. Each figure represents the ROC curve of each of the submodels of the ensemble (thin lines) together with the ROC curve of the corresponding ensemble model (thick line). At the top of each figure the area-under-the-curve together with the standard error is indicated. One can see an improvement of nearly 10% by taking the Coupled ensemble.

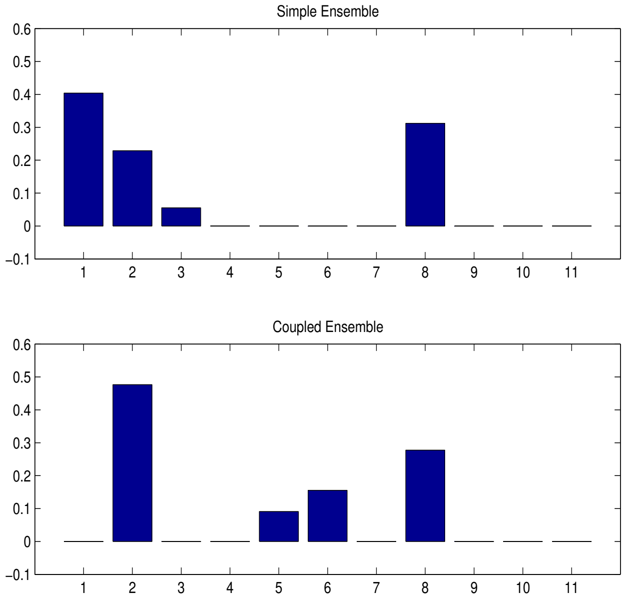

Figure 6.13: Sparseness of the solution for the parameter vector of the second layer (Tic-Tac-Toe data set). In the uncoupled (top) ensemble as well as the coupled ensemble (bottom) we see that only 4 out of 10 models have a weighting different from zero.

ensemble case only 4 out of 10 models have a weight different from zero.

Australian credit card data set The Australian Credit Card data set is an UCI data set donated by Quinlan and is one of the Credit Approval Databases that were used in the Statlog project. There 690 observations in this data set with six numerical and eight attributes. The optimal hyperparameters for the Gaussian RBF kernel are $\left(\gamma, \sigma^2\right) = (9.03e2, 12.15)$. All the ensembles were based on 10 submodels. Similar to all the previous experiments did we check the influence of the coupling parameters ν by a randomization test for different values of ν. The behavior was similar to the Tic-Tac-Toe experiment with a minimum in $\nu = 1$.

For this optimal value of ν we then did a randomization test to check the differences between a coupled and uncoupled ensemble. The results are given in Figure 6.14 and the numerical values of the distributions together with a result of the Wilcoxon ranksum test are given in Table 6.2. Again we may

	Simple Ensemble			Coupled Ensemble				
	mea	med	va	mea	med	va	h	p
ttt,11,T	0.25	0.25	7.09e-4	0.21	0.21	8.37e-4	1	0
acr,10,T	0.16	0.16	5.08e-4	0.15	0.14	5.08e-4	1	9.8e-5
adult,100,T	0.27	0.28	7.92e-4	0.24	0.24	8.34e-4	1	2.7e-3

Table 6.2: Misclassification rates on a test set (Tic-Tac-Toe (ttt), Australian Credit Card Data Set (ACR) and the Adult Data Set (ADULT)). The number of models is indicated by the second number in the table, example ttt11 is an ensemble model based on 11 individual models on, the ttt prediction. We give the mean (mea), median (mea) and variance (var) of these distributions after 100 randomization. In the last two columns the result of a Wilcoxon ranksum test are given: h indicates the test that the two distributions are different and p is the corresponding p-value.

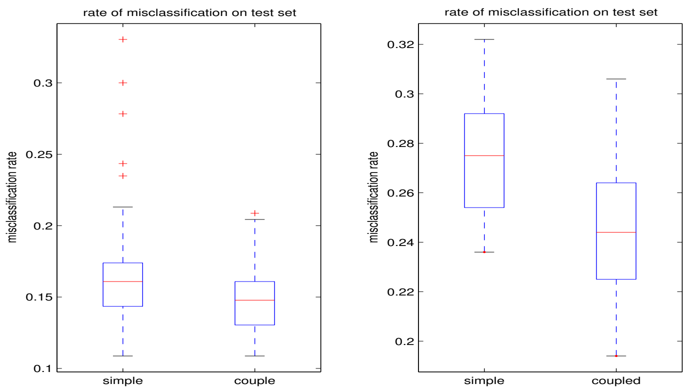

Figure 6.14: This figure shows the distribution of the misclassification rates on the Australian Credit Card data set (figure-left) of uncoupled (right) and the coupled (left) ensemble models after 100 randomizations. One sees the improvement of the coupled ensemble with a lower mean and smaller variance. Both experiments are done with a $\nu = 1$. (Figure-right) similar experiments and results on the Adult data set.

conclude that the performance of the coupled ensemble is significantly better compared with the uncoupled ensemble. If one compares the differences between the mean and median of both distributions, one sees that it is small. However, there is a difference in the distribution as indicated by the ranksum test. If we take a closer look at Figure 6.14 we see that the uncoupled ensemble counts some outliers with a bad performance. It are these results of the misclassification tests that cause the difference. This behavior was often noticed in our experiments (see also Figure 6.9). The coupling between the elements of the ensemble causes the models to be more robust against outlier performances.

Adult data set The Adult data set is an UCI data set donated by Kohavi. It involves the prediction whether income exceeds 50,000 dollars a year based on census data. The original data set consists out of 48,842 observations each described by six numerical and eight categorical attributes. All the observations with missing values were removed from consideration. To show the use of our method towards larger data set we took a subset of 8,000 points. We used another 500 points as test set. The optimal hyperparameters for the Gaussian RBF kernel are $(\gamma, \sigma^2) = (1.11, 3.00e1)$ and the coupling parameters $\nu = 1$. All the experiments used the transductive setting in which the test set is used as coupling set for the coupled ensemble model. To study the behavior of our models with large ensembles we constructed the ensemble on 100 submodels. In this way, the needed memory complexity is $\mathcal{O}(27N^2/100)$ which means a reduction by a factor 4 compared to the memory usage of a traditional kernel model (in a worst case scenario).

In a first experiment on this data set, we used (6.46) to compute the weights of the second layer. As explained in the previous section, this results into a sparse solution. Only about 5% of the submodels had a weight different from zero. Computationally, this means a big advantage towards evaluation at new points with this ensemble model. However, after a randomization experiment, we did not observe a statistically significant difference of the misclassification performance between the coupled and uncoupled ensemble models on this problem. The explanation behind this is the following. In the previous sections we mentioned that coupling results into a reduction of the 'outlier' models. Those submodels of the ensemble who intent to have a bad performance are corrected by the other models. In this way, the overall performance of the ensemble model increases. But, the computation of weights of the second layer based on (6.46) does a similar thing. Also here the submodels with a bad performance are pruned away.

If, like in this case, only 5% of the submodels have a weight different from zero, the effects of the coupling is less obvious. To check this we did another experiment.

In a second experiment, we computed the weights of the second layer based on (6.31). In this case the ensemble model is built on all 100 submodels, so there is no pruning. In this case we see a clear distinction between the performances of the uncoupled and the coupled ensemble as is shown in Table 6.2. This was again verified by a Wilcoxon ranksum test after a randomization experiment of which the results are also shown in Table 6.2. Figure 6.14 the distributions of the misclassification performance after the randomization experiment are shown for the uncoupled and coupled ensemble. In this case where the ensemble models is built on a larger set of submodels the effect of the coupling becomes more clear.

6.8 Conclusions

In this chapter we have given an overview of the most popular ensemble methods. Hereby we explained when one should use theses methods and what the advantages are both from theoretical learning as from computational point of view. We explained how the training process can be divided in two parts. In the first part one designs a set or ensemble of learning models. These modes will then be combined in a second step to an ensemble model. This gives a clear view of all the different ensemble model strategies like there are: bagging, boosting, mixture of experts and stacking models.

Since the main goal of our work is to use kernel models for large scale application we explained how these ensemble methodologies can be applied to kernel models. Hereby we summarized some of the theoretical advantages these methods offer. Again a clear focus on the computational necessities of these methods were given.

In a third part we have shown the effect of coupling between the submodels within ensemble models. We introduced the concept of a coupling set and showed that this can be used to achieve a new way of transductive learning for regression as well as classification problems. We explained the links between multitask learning and coupling and showed that coupling can be regarded as a group regularization imposed on all the elements of the ensemble. We also demonstrated different adaptations of the learning scheme towards large data sets. In the experiments we have illustrated how the coupling between the elements of the ensemble almost always leads to improved test set performance, in comparison with uncoupled ensembles.

Chapter 7

General Conclusions and Future Research

7.1 Summary

In the previous sections we have shown different derivations of kernel models using a quadratic loss function. In a first approach we started with an approximation in an RKHS. By optimizing the functional, which balances the empirical risk and the structural risk, one obtains a linearly parameterized model. By using the squared loss function for minimizing the empirical risk the optimal parameters of the model are found by solving a linear system. This leads to Regularization Network. The same formulation can be derived starting form an optimization approach by defining a linear model in feature space. Also here one defines a cost function, which balances between a structural risk or regularization and empirical risk. This viewpoint introduces the Least-Squares Support Vector Machines for an optimization approach with primal-dual formulations.

We showed that both methodologies apply for regression and time-series prediction as well as for classification problems. The main characteristic of these model formulations is the unique solution optimization problem that defines the model. The solutions of this optimization problem can be found by solving of a set of linear systems of which the computational complexity scales up with the number of input points N.

These RN and LS-SVM models have many relations to other theories. Models we described for regression are closely related to Gaussian processes [81], Kriging ([73],[27]), kernel ridge regression [116] and the presented Regularization Networks ([108]). But also the classification setup finds many similar or closely related derivations as there are: kernel Fisher discriminant analysis [87], proximal support vector machines [50] and Regularized Least-Square classification [113]. Therefore most of the solutions that were presented in this work apply to all of these models.

We also explained links between these models and Support Vector Machines. An overview of the state-of-the-art large scale training procedures are discussed and the reason for sparseness is studied from an optimization perspective.

In the second chapter we presented different numerical methods for training LS-SVM models. We studied direct and iterative methods and compared their characteristics towards large scale problems. The training of the LS-SVM consists of solving a set of linear systems, which scales with the number of data points. Because of the symmetric positive definite structure of these linear systems, many efficient numerical algorithms exist. We showed that towards large data sets, the iterative methods (SOR, CG,...) have many advantages.

The first iterative method we have tested was the successive overrelaxation method. Exploiting the structure in combination with a block-implementation of the SSOR gave a more memory efficient algorithm. We also showed that the GM-acceleration reduces the required amount of iterations before convergence. The extra advantage is that its dependency on the block size and the overrelaxation parameter is reduced. This GM-acceleration combined with a block-SSOR gives many advantages towards large data sets compared with the classical SOR method.

Comparisons between the block-SSOR (with GM acceleration) and the conjugate gradient (CG) method showed that the latter outperforms the block-SSOR in all cases. A second Krylov method, the block conjugate gradient, even showed an improved performance compared to the CG. On test on the UCI data sets, training methods using the block-CG method converges faster for the non-linear classification tasks. Therefore we conclude that the Krylov methods (CG and block-CG) are most suited for training on large data sets for $N < 5000$.

In the third chapter we have studied the role of the choice of the kernel function and its influence on the training process. To study the computational consequences of using the different types of kernels, we made a classification of the different types of kernels. We discussed the Stationary, Locally stationary and the Non-Stationary kernels. For each type we discussed the possible computational or memory advantages that can be achieved by exploiting its properties.

We paid much attention to the use of compactly supported kernels. We showed that the popular RBF kernels can be efficiently transformed into a kernel with compact support. The use of compactly supported kernels can decrease the computational cost and memory requirements. In our study we have seen that for all low dimensional problems the generalization performance as well as the conditioning of the matrices is comparable for compactly supported kernels. This last feature is important towards the convergence properties of the iterative methods like the conjugate gradient method. However, on a problem of chaotic time-series prediction the compactly supported RBF kernels fails to produce good results when having a sparse Gram matrix. As a result one may conclude that compactly supported RBF kernels may be useful for low dimensional problems $d < 5$. But also in a general context, one should be careful to use these kernels. In a last part of this chapter we have discussed some efficient procedures for training LS-SVM models with non-stationary separable and linear kernels. We showed that the intrinsic definition of these kernels results in rank deficient kernel matrix. This rank deficiency leads to good factorizations of the kernel matrix that can reduce the computational and memory complexity of the algorithms.

In Chapter 5 we have shown that the computational complexity of the kernel models can further be reduced by using low rank approximations for general types of kernels. We discussed two types of approximations.

The first class makes use of a factorization of the kernel matrix. By applying the Sherman-Morrison-Woodbury formula to the factorized kernel matrix solving the linear systems can be done efficiently. One of the methods to factorize is based on the the Nyström approximation. This leads to a factorization of the kernel matrix at a fixed but low computational cost. We showed different types of Nyström factorization and their corresponding computational necessities. A second approach for factorization was based on the incomplete Cholesky factorization. This method fully exploits the rank of the kernel matrix in an iterative way. Although that all the approximations resulting from the low rank factorization methods showed in almost no cases a decrease in performance, the computational advantages are not always guaranteed. Much depends on the type of learning problem one wants to solve. In the regression task the Cholesky factorization showed an impressive decrease in computational complexity. However, for classification tests the Nyström fractionization showed a clear computational advantage.

A second class of low rank methods are based on different methodology. To decrease the computational complexity the number of parameters in the models are reduced. This can be achieved by either reduce the number of parameters in the dual space by using an optimal basis representation in feature space, or by reducing the number of parameters in primal space. The latter uses again Nyström approximation which leads to the *fixed size approximation* with a sparse representation. The key concept is to construct a basis based on a subsample found by using different selection criteria.

We conclude that the computational advantages resulting form these different low rank models are almost similar. Depending on the task one method gives a better computational advantage compared to the other. Also from the learning performance perspective a significant reduction in generalization performance is rare. Therefore these low rank approximation are promising and advisable in most large scale tasks for $5000 < N < 50000$.

In Chapter 6 we have given an overview of the most popular ensemble methods. This learning methodology gives the opportunity to divide the large computational cost over a set of learning models, which are combined afterwards. This makes them applicable for data sets of size $N > 50000$. Hereby we explained when one should use theses methods and what the advantages are both from theoretical learning as from computational point of view. We explained how the training process can be divided in two parts. In the first part one designs a set or ensemble of learning models. These modes will then be combined in a second step to an ensemble model. This gives a clear view of all the different ensemble model strategies like there are: bagging, boosting, mixture of experts and stacking models.

Since the main goal of our work is to use kernel models for large scale application will we explain how these ensemble methodologies can be applied to kernel models. Hereby we summarized some of the theoretical advantages these methods offer. Again a clear focus on the computational necessities of these methods were given.

In a last part we have shown the effect of coupling between the submodels within ensemble models. We introduced the concept of a coupling set and showed that this can be used to achieve a new way of transductive learning for regression as well as classification problems. We explained the links between multitask learning and coupling and showed that coupling can be regarded as a group regularization imposed on all the elements of the ensemble. We also demonstrated different adaptations of the learning scheme towards large data sets. In the experiments we have illustrated how the coupling between the elements of the ensemble almost always leads to improved test set performance, in comparison with uncoupled ensembles.

7.2 Future Research

Although a lot of progress has been made, there are still many aspects that need further investigation. We will postulate different possibilities for improvements according to the structure as followed in the thesis.

In Chapter 1 we have seen that an appropriate choice of the cost functions introduces sparseness as is known form the SVM literature. Also for the LS-SVM models, which are characterized by the equality constraints, other cost functions might introduce sparseness. One of the possibilities is by using other error penalization criteria. Once this sparseness is achieved all the known chunking strategies can be applied. However, since the sparseness is the result of the complementary slackness conditions, inequalities are necessary. Using these inequalities most probably will lead to a constrained optimization problem which can not be transformed into a linear system.

The most optimal Krylov methods for the saddle point problems as introduced in Chapter 3 is still an active domain of research [114]. The use of optimal preconditioners specifically designed for the semi positive definite kernel matrices might still reduce the computational demands of the Krylov methods.

Also in the kernel choice there are many possibilities for further improvement. Little studies have been done on which kernel is most optimal for high dimensional problems in comparison to its computational cost. One of the possibilities to reduce the computational cost are kernels with a compact support, but we still need theoretical guidelines on when to use these kernels and how to adapt the cut-off distance automatically.

Further, we have seen that separable nonstationary kernels by nature are rank deficient. However there is not much known about separable nonstationary or other rank deficient kernels in literature. More research has to be done on this subject because it would reduce the computational bottleneck hopefully without the reducing the generalization performance of the corresponding kernel models.

For the known factorization methods like the Nyström approximation it is not known which is the best sample to construct the factorization on. Many sampling schemes have been proposed in literature based on information theory criteria, optimal basis vectors in feature space,... but also here not much is known about which is theoretically the best method.

In the last chapter we introduced ensemble methods on subsamples. Only recently the first proves of the theoretical benefits of this strategy are given. It is still an open questions if it is necessary to use disjoint or overlapping subset. In this thesis we assumed that all the data is necessary to build our model on. Other possible scenario are that one monitors the performance of the ensemble constructed on a subset of the whole data set and decides online to add extra models to the ensemble.

Also the coupled learning raises many unanswered questions. One of the open problems is to study the effect of construction an ensemble based on kernel models using different kernels or with different hyperparameters. The latter method can be the result of different solutions resulting form a multi-start coupled-local-minimizers optimization of the hyperparameter selection. In this way one achieves a synchronization between the models on two different levels: the hyperparameter selection and the ensemble model formulation.

Appendix A

Numerical Linear Algebra and Optimization

In this appendix we give a summary of several definitions and properties originating from numerical algebra, which are be often used during the text.

For proofs or further information the reader may consult the books of Golub and Van Loan [57] and Datta [30].

A.1 Sherman-Morrison-Woodbury formula

If $A \in \mathbb{R}^{n\times n}$ and $U, V \in \mathbb{R}^{n\times k}$ then

$$\left(A + UV^T\right)^{-1} = A^{-1} - A^{-1}U\left(I + V^TA^{-1}U\right)^{-1}V^TA^{-1}, \tag{A.1}$$

where we assume that A and $I + V^TA^{-1}U$ are non-singular [57].

A.2 Positive (semi) definite matrices

In this appendix we will give a summary of the definitions and properties of positive (semi) definite matrices. A symmetric matrix $A \in \mathbb{R}^{d\times d}$ is positive definite if for every non-zero vector $x \in \mathbb{R}^d$ holds that $x^TAx > 0$. If this inequality is only fulfilled upon $x^TAx \geq 0$ the matrix is called positive semi definite. The following properties hold for positive (semi) definite matrices:

- a symmetric matrix A is positive definite $\Leftrightarrow \forall\lambda_i > 0$, λ_i eigenvalues of A,

- a symmetric matrix A is positive semi definite $\Leftrightarrow \forall \lambda_i \geq 0$, λ_i eigenvalues of A,
- if A, B are positive definite then $A + B$ is pos. definite,
- if A is positive definite and B is pos. semi definite then $A + B$ is positive definite,
- if A, B are positive semi definite then $A + B$ is positive semi definite,
- a symmetric matrix A is positive definite $\Leftrightarrow \forall a_{i,j} > 0$ where $A = [a_{ij}]_{i,j=1\ldots n}$,
- if $A \in \mathbb{R}^{n \times n}$ is positive definite and $X \in \mathbb{R}^{n \times k}$ of rank k then $X^T A X$ is positive definite. From this follows that $X^T X$ is positive definite. If X is rank deficient, which means that $\text{rank}(X) < k$, then $X^T X$ is positive semi definite.
- if A is positive (semi) definite, then A^p, where $p \in \mathbb{N}_0$, is also positive (semi) definite.

A.3 Condition number

The condition number of a matrix with respect to the p-norm is defined as $\text{cond}_p(A) = \|A\|_p \left\|A^{-1}\right\|_p$ The following properties hold for the condition number of a matrix:

- $\forall p$, $\text{cond}_p(A) \geq 1$,
- if A is an orthogonal matrix, $\text{cond}_2(A) = 1$,
- $\text{cond}_p(A^T A) = \left(\text{cond}_p(A)\right)^2$,
- $\text{cond}_p(A) = \text{cond}_p(A^T)$,
- $\text{cond}_p(AB) \leq \text{cond}_p(A)\text{cond}_p(B)$,
- $\text{cond}_p(\alpha A) = \text{cond}_p(A)$ if α is a non zero scalar,
- $\text{cond}_2(A) = \frac{\sigma_1}{\sigma_n}$ where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$ are the singular value of A.

A.4 A dot product

A **Dot Product** on a vector space $\mathcal{V}$ is a symmetric bilinear form,

$$\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \to \mathbb{R} : (\mathbf{x}, \mathbf{z}) \mapsto \langle \mathbf{x}, \mathbf{z} \rangle \tag{A.2}$$

that is strictly positive definite. This means that $\forall \mathbf{x} \in \mathcal{V} : \langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ and if $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ only if $\mathbf{x} = \mathbf{0}$.

A.5 The Karush-Kuhn-Tucker theorem

Assume we want to solve the following optimization problem over the convex domain $\Omega \subseteq \mathbb{R}^N$:

$$\begin{cases} \min\limits_{\mathbf{x}} f(\mathbf{x}) \\ \text{s.t. } g_i(\mathbf{x}) = 0, \ \forall i \in \{1 \dots m\}, \\ \quad h_j(\mathbf{x}) \geq 0, \ \forall j \in \{1 \dots n\}, \end{cases} \tag{A.3}$$

with f is a convex and first order differentiable function and g_i, h_j are affine.

The *Lagrangian function* defined for this problem is given by

$$\mathcal{L}(\mathbf{x}, \alpha, \beta) = f(\mathbf{x}) - \sum_{i=1}^{m} \alpha_i g_i(\mathbf{x}) - \sum_{j=1}^{n} \beta_j h_j(\mathbf{x}), \tag{A.4}$$

where the $\{\alpha_i\}_{i=1}^m$ and $\{\beta_j\}_{j=1}^n$ are the *Lagrange multipliers.*

The necessary and sufficient conditions for a point $\mathbf{x}^*$ to be a minimizer of the above optimization problem, where certain regularity assumption holds at $\mathbf{x}^*$, is the guaranteed existence of Lagrange multipliers α_i^* and β_j^* that satisfy the following system:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = 0, \tag{A.5}$$

$$g_i(x) = 0, \ \forall i \in \{1 \dots m\}, \tag{A.6}$$

$$h_j(\mathbf{x}) \geq 0, \ \forall j \in \{1 \dots n\}, \tag{A.7}$$

$$\beta_j \geq 0, \ \forall j \in \{1 \dots n\}, \tag{A.8}$$

$$\beta_j h_j(\mathbf{x}) = 0, \ \forall j \in \{1 \dots n\}. \tag{A.9}$$

These are called the *Karush-Kuhn-Tucker conditions*. The final condition is sometime referred to as the complementary slackness condition. This states that β_j^* and $h_j(\mathbf{x}^*)$ can not be both non-zero.

For further information about optimization theory we advise the following work: [96],[97] and [10].

Appendix B

Jitter Factor

In Chapter 2 it was shown that for kernel models like LS-SVM or Regularization Networks the training consisted of solving linear systems of the form:

$$\left(K + \frac{1}{\gamma} I\right) \mathbf{x} = \mathbf{b}. \tag{B.1}$$

Hereby the term $\frac{1}{\gamma}I$ acts as a *jitter factor*. But what is its influence form a numerical point of view? We will show three important aspects.

A first effect of the jitter factor is that it makes the matrix $H = K + \frac{1}{\gamma}I$ positive definite. Following the definition and properties of positive definiteness (Appendix A.2) it can be proven that : $x^T(K+\frac{1}{\gamma}I)x = x^T Kx + \frac{1}{\gamma}x^T x > 0$ for every non-zero vector x, $\gamma > 0$ and semi-positive definite K.

A second aspect is the effect of the jitter factor on the solution of the linear system. For kernel models we know that the matrix K is in general positive semi-definite. The solution of the linear system $K\mathbf{x} = \mathbf{b}$ for the cases that $\text{rank}[K] = \text{rank}[K\,|b] = r \leq N$ is a solution space $\mathcal{S}$ of dimension r. The solution space $\mathcal{S}'$ of $\left(K + \frac{1}{\gamma}I\right)\mathbf{x} = \mathbf{b}$ has no points in common with $\mathcal{S}$. Remember that $\mathcal{S}'$ only has one element because $K + \frac{1}{\gamma}I$ is positive definite. But the solution of $\mathcal{S}$ that is closest to $\mathcal{S}'$ is the one with the smallest norm. This can be easily seen as follows. For the elements of $\mathcal{S}'$ the minimization of $\left\|\left(K + \frac{1}{\gamma}I\right)\mathbf{x} - \mathbf{b}\right\|_2$ is equal to zero. The minimization of $\left\|\left(K + \frac{1}{\gamma}I\right)\mathbf{x} - \mathbf{b}\right\|_2$ by elements of $\mathbf{x} \in \mathcal{S}$ is the element with the smallest 2-norm, since

$$\left\|\left(K + \frac{1}{\gamma}I\right)\mathbf{x} - \mathbf{b}\right\|_2 = \left|\frac{1}{\gamma}\right| \|\mathbf{x}\|_2 . \tag{B.2}$$

One can also regard the jitter factor as a disturbance of the matrix K

such that $\left(K+\frac{1}{\gamma}I\right)(\mathbf{x}+\Delta\mathbf{x})=\mathbf{b}$. If one assumes that K is non-singular and that $\left\|\frac{1}{\gamma}I\right\|_2 \leq \left\|K^{-1}\right\|_2^{-1}$ then the disturbance on the solution $\Delta\mathbf{x}$ is bounded by

$$\frac{\|\Delta\mathbf{x}\|_2}{\|\mathbf{x}\|_2} \leq \frac{\mathrm{cond}_2(K)\frac{\left\|\frac{1}{\gamma}I\right\|_2}{\|K\|_2}}{1-\mathrm{cond}_2(K)\frac{\left\|\frac{1}{\gamma}I\right\|_2}{\|K\|_2}}. \tag{B.3}$$

A third effect that can easy bee seen is that the jitter factor $\frac{1}{\gamma}I$ uniformally increases the eigenvalue spectrum of K with $\frac{1}{\gamma}$. this is proven as follows. Let λ_i and $\mathbf{v}_i$ be the eigenvalues and eigenvectors of K. Then $\forall i$

$$\begin{aligned}\left(K+\frac{1}{\gamma}I\right)\mathbf{v}_i &= K\mathbf{v}_i+\frac{1}{\gamma}\mathbf{v}_i \\ &= \left(\lambda_i+\frac{1}{\gamma}\right)\mathbf{v}_i.\end{aligned} \tag{B.4}$$

which proves that the eigenvectors are the same as those of K and that the corresponding eigenvalues increased by an amount $\frac{1}{\gamma}$. This has important implications for the convergence of some numerical procedures. This also effects the condition number of $K+\frac{1}{\gamma}I$ where

$$\mathrm{cond}_2\left(K+\frac{1}{\gamma}I\right)=\frac{\lambda_{\max}+\frac{1}{\gamma}}{\lambda_{\min}+\frac{1}{\gamma}} \tag{B.5}$$

and $\lambda_{\min}$ and $\lambda_{\max}$ are respectively the minimum and maximum eigenvalues of K.

Appendix C

UCI Data Sets

We give a small description of some of the UCI data [8] set that are often used in this work. Table C.1 shows the characteristics of the data sets together with the optimal hyperparameters for the linear kernel and the Gaussian RBF kernel. These optimal hyperparameters are found after 10-fold cross-validation as given by the LS-SVM toolbox [104] after standardizing the data.

1. The Statlog Australian Credit (**acr**) was donated by R. Quinlan and is one of the Credit Approval Databases which were used in the Statlog project. There are six numerical attributes, eight categorical attributes and 690 records.

2. The BUPA Liver Disorders (**bld**) data set was contributed by R.S. Forsyth and consists of 325 instances of male patients with 6 numerical attributes considered relevant for detecting liver disorders.

3. The Statlog German Credit data set (**gcr**) is another Statlog Credit Approval Database consisting of 1000 instances each described by seven numerical and thirteen categorical attributes. This data set was donated by H. Hofmann. Notice tbat the Statlog project used unequal misclassification costs, making a comparison with our results infeasible.

4. The Statlog Heart Disease (**hea**) data set is from the Cleveland Clinic Foundation, courtesy of R. Detrano. It contains 270 instances each described by 7 numerical and 6 categorical attributes. Again, the Statlog project used unequal misclassification costs.

5. The Johns Hopkins University Ionosphere (ion) data set was contributed by V. Sigillito and represents radar returns from the ionosphere. The data set consists of 351 samples with 34 numerical attributes. One attribute was removed because it was zero for all instances in the data set.

6. The Pima Indians Diabetes (**pid**) data set was donated by V. Sigillito and contains 768 records of female Pima Indians. Each record contains of 8 numerical attributes partially describing a patient's medical history. It is a considered quite a challenge on which even state of the art neural techniques still misclassify about one fourth of the population.

7. The Sonar (**snr**) is the data set used by Gorman and Sejnowski [58] to illustrate the classification of sonar signals by using back propagation neural networks. Each instance has 60 features ranging from 0 to 1.

8. The Tic-Tac-Toe (**ttt**) Endgame database is another UCI data set contributed by D. W. Aha, encoding the complete set of possible board configurations at the end of tic-tac-toe games. The target concept is "win for x" where "x" is assumed to have played first. The data set consists of 958 observations each described by 9 categorical attributes.

9. The Wisconsin Breast Cancer (**wbc**) data set was collected at the University of Wisconsin by W.H. Wolberg. It consists of 699 records with 9 features. Note that the 16 records that contained missing values were removed from the data set.

10. The Adult (**adu**) database was donated by R. Kohavi and involves the prediction whether income exceeds 50000 dollars a year based on census data. It consists of 48842 observations each described by six numerical and eight categorical attributes. Again, 3620 observations had missing values and were removed from consideration. Notice hereby for computational reasons the hyperparamter selection is computed based on a subsample of 2000 points.

data	N	N_{train}	N_{test}	d	d_{num}	d_{cat}	γ_{lin}	γ_{RBF}	σ^2_{RBF}
acr	690	460	230	14	6	8	6.74e-3	9.04e-2	1.22e1
bld	345	230	115	6	6	0	1.08e-1	1.44	2.12e1
grc	1000	666	334	20	7	13	4.89e1	4.29	1.66e1
hea	270	180	90	13	7	6	6.74e-3	1.12	5.06
ion	351	234	117	33	33	0	6.74e-3	2.24	1.53e1
pid	768	512	256	8	8	0	5.42e2	1.13	5.68
snr	208	138	70	60	60	0	6.74e-3	5.08e1	2.54e1
ttt	958	638	320	9	0	9	6.28e-3	9.46e1	9.96
wbc	683	455	228	9	9	0	5.88e1	1.88e-1	3.22e1
adu$_{sub}$	2000	1334	666	14	6	8	2.52	1.11	3.00e1
adu$_{Org}$	45222	33000	12222	14	6	8	-	-	-

Table C.1: Details of the UCI classification datasets like the input dimension d which can be divided in d_{num} numerical attributes and d_{cat} categorical ones according to the data sets. Further the size of the data set N is given together with the size of the parts used for training N_{train} and testing N_{test}. The optimal hyperparamters were found by 10-fold crossvalidation by making use of LS-SVMlab.

Bibliography

[1] B.J. Bakker and T.M. Heskes. Clustering ensembles of neural network models. *Neural Networks*, 16(2):261–269, March 2003.

[2] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.* SIAM, Philadelphia, PA, 1994. ftp.netlib.org/templates/templates.ps.

[3] G. Baudat and F. Anouar. Kernel-based methods and function approximation. In *Proc. of the International Joint Conference on Neural Networks (IJCNN)*, pages 1244–1249, Washington DC, 2001.

[4] R.K. Beatson, J.B. Cherrie, and C.T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics*, 11:253–270, 1999.

[5] R.K. Beatson, W.A. Ligth, and S. Billings. Fast solutions of radial basis function interpolation equations: Domain decomposition methods. *Siam Journal on Scientific Computing*, 22(5):1717–1740, 2000.

[6] R.K. Beatson and G.N. Newsam. Fast evaluation of radial basis functions: Moment-based methods. *Siam Journal on Scientific Computing*, 19(5):1428–1449, 1998.

[7] C.M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[8] C.L. Blake and C.J. Merz. UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Sciences.

[9] O. Bousquet and A. Elisseeff. Stability and generalization. *Journal of Machine Learning Research 2*, pages 499–526, 2002.

[10] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[11] J. De Brabanter, K. Pelckmans, J.A.K. Suykens, and J. Vandewalle. Robust cross-validation score function for non-linear function estimation. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN 2002)*, pages 713–719, Madrid, Spain, 2002.

[12] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[13] L. Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Department of Statistics, University of California, Berkeley, 1996.

[14] L. Breiman. Arcing the edge. Technical Report 486, Department of Statistics, University of California, Berkeley, 1997.

[15] C. Brezinski. Convergence acceleration during the 20th century. *Journal of Computational and Applied Mathematics*, 2000.

[16] M.P.S. Brown, W.N. Grundy, D. Lin., N. Cristianini, C. Sugnet, M. Ares, and D. Haussler. Support vector machine classification of microarray gene expression data. *Proceedings of the National Academy of Sciences*, 97(1):262–267, 2000.

[17] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.

[18] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[19] G.C. Cawley and N.L.C. Talbot. Efficient formation of a basis in a kernel induced feature space. In *Proc. European Symposium on Artificial Neural Networks (ESANN)*, pages 1–6, Brugge Belgium, 2002.

[20] G.C. Cawley and N.L.C. Talbot. Improved sparse least-squares support vector machines. *Neurocomputing*, 48:1025–1031, 2002.

[21] O. Chapelle, V. Vapnik, and J. Weston. Transductive inference for estimating values of functions. In *Advances in Neural Information Processing Systems*, pages 421–427, 1999.

[22] S. Chen, C. Cowan, and P. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[23] J.B. Cherrie, R.K. Beatson, and G.N. Newsam. Fast evaluation of radial basis functions: Methods for generalised multiquadrics in R^n. *Siam Journal on Scientific Computing*, 23(5):1549–1571, 2002.

[24] K.S. Chua. Efficient computations for large least squares support vector machine classifiers. *Pattern Recognition Letters*, 24:75–80, 2003.

[25] R.T. Clemen. Combining forecasts: A review and annoted bibliography. *International Journal of Forecasting*, 5:559–583, 1989.

[26] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002.

[27] N. Cressie. *Statistics for Spatial Data*. John Wiley & Sons, New York, 1993.

[28] N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, 2000.

[29] R.D. da Cunha and T. Hopkins. A comparison of acceleration techniques applied to the SOR method. Technical Report 4-94*, University of Kent, Computing Laboratory, March 1994.

[30] B.N. Datta. *Numerical Linear Algebra*. Brooks/Cole, 1995.

[31] D. DeCoste and K. Wagstaff. Alpha seeding for support vector machines. In *International Conference on Knowledge Discovery & Data Mining (KDD)*, 2000.

[32] T.G. Dietterich. Machine learning research: Four current directions. *AI Magazine*, 18(4):97–1367, 1997.

[33] T.G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857, 2000.

[34] T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–158, 2000.

[35] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley interscience, 2 edition, 2001.

[36] A. Elisseeff, T. Evgeniou, and M. Pontil. Hypothesis stability of randomized algorithms with an application to bootstrap methods. Technical Report 2002/74/TM, INSEAD, Fontainebleau, 2002.

[37] A. Elisseeff and M. Pontil. Leave-one-out error and stability of learning algorithms with applications. In J.A.K. Suykens, G. Horvath, S. Basu, C. Micchelli, and J. Vandewalle, editors, *Advances in Learning Theory: Methods, Models and Applications*, volume 190 of *NATO Science Series III: Computer and Systems Sciences*, pages 111–125. IOS Press Amsterdam, 2003.

[38] M. Espinoza, K. Pelckmans, L. Hoegaerts, J.A.K. Suykens, and B. De Moor. A comparative study of LS-SVM's applied to the silver box identification problem. Technical Report 04-17, ESAT-SISTA, Katholieke Universiteit Leuven, Belgium, 2004. (submitted for publication).

[39] M. Espinoza, J.A.K. Suykens, and B. De Moor. Least squares support vector machines and primal space estimation. In *Proc. of the IEEE 42nd Conference on Decision and Control*, pages 3451–3456, Maui, USA, Dec. 2003.

[40] T. Evgeniou. *Learning with kernel machine architectures*. PhD thesis, Massachusetts Institute of Technology, 2000.

[41] T. Evgeniou, L. Perez-Breva, M. Pontil, and T. Poggio. Bounds on the generalization performance of kernel machine ensembles. In *Proc. 17th International Conf. on Machine Learning*, pages 271–278, San Francisco, CA, 2000.

[42] T. Evgeniou, T. Poggio, M. Pontil, and A. Verri. Regularization and statistical learning theory for data analysis. *Computational Statistics & Data Analysis*, 38:421–432, 2002.

[43] T. Evgeniou, M. Pontil, and A. Elisseeff. Leave-one-out error, stability, and generalization of voting combinations of classifiers. Technical report, INSEAD -TM, 2001.

[44] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[45] J. Fan and Q. Yao. *Nonlinear Times Series. Nonparametric and Parametric Methods.* Springer Series in Statistics, 2003.

[46] M.R. Field. Optimizing a parallel conjugate gradient solver. *SIAM Journal on Scientific Computing*, 19(1):27–37, 1998.

[47] S. Fine and K. Scheinberg. Efficient training using low-rank kernel representations. *Journal of Machine Learning Research 2*, pages 243–263, 2001.

[48] B. Fischer. *Polynomial Based Iteration Methods for Symmetric Linear Systems*. Wiley Teubner, 1996.

[49] C. Fowlkes, S. Belongie, and J. Malik. Efficient spatiotemporal grouping using the Nyström method. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2001.

[50] G. Fung and O.L. Mangasarian. Proximal support vector classifiers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 77–86, San Francisco, USA, 2001.

[51] A. Gammerman, V. Vapnik, and V. Vovk. Learning by transduction. In *Uncertainty in Artificial Intelligence*, pages 148–155, 1998.

[52] M. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research 2*, pages 299–312, 2001.

[53] M. Gibbs and D. MacKay. Efficient implementation of gaussian processes. ftp://www.inference.phy.cam.ac.uk/pub/www/mng10/GP/gpros.ps.gz, 1997.

[54] J.R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in matlab:design and implementation. *SIAM Journal on Matrix Analysis.*, 13(1):333–356, 1992.

[55] M. Girolami. Orthogonal series density estimation and the kernel eigenvalue problem. *Neural Computation*, 14(3):669–688, 2002.

[56] T. Gneiting. Compactly supported correlation functions. *Journal of Multivariate Analysis*, 82:493–508, 2002.

[57] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore MD, 1989.

[58] R.P. Gorman and T.J. Sejnowski. Analysis of hidden units in layerd network trained to classifiy sonar targets. *Neural Networks*, 1:75–89, 1988.

[59] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* SIAM, Philadelphia, 1997.

[60] B. Hamers, J.A.K. Suykens, and B. De Moor. A comparison of iterative methods for least squares support vector machine classifiers. Technical Report 01-110, ESAT-SISTA, Katholieke Universiteit Leuven, Belgium, 2001.

[61] B. Hamers, J.A.K. Suykens, and B. De Moor. Coupled transductive ensemble learning of kernel models. Technical report, ESAT-SISTA, Katholieke Universiteit Leuven, Belgium, 2003. (submitted to Journal of Machine Learning Research 2).

[62] B. Hamers, J.A.K. Suykens, V. Leemans, and B. De Moor. Ensemble learning of coupled parmeterized kernel models. In *Supplementary Proc. of the International Conference on Artificial Neural Networks and International Conference on Neural Information Processing (ICANN/ICONIP)*, pages 130–133, Istanbul, Turkey, 2003.

[63] B. Hamers, J.A.K. Suykens, and B. De Moor. Compactly supported rbf kernels for sparsifying the gram matrix in LS-SVM regression models. In *International Conference on Artificial Neural Networks ICANN*, pages 130–133, Madrid, Spain, 2002. (Supplementary Proceedings).

[64] J.A. Hanley and B.J. McNeil. A method of comparing the areas under the reciever operating characteristic curves derived from the same cases. *Radiology*, 148:839–843, September 1983.

[65] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[66] G.H. Hardy and E.M. Wright. *An Introduction to the Theory of Numbers.* Oxford, England: Clarendon Press, 5 edition, 1979.

[67] L. Hoegaerts, J.A.K. Suykens, J. Vandewalle, and B. De Moor. Subset based least squares subspace regression in RKHS. *Neurocomputing*, 2004. in press.

[68] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of the European Conference on Machine Learning*, pages 137–142, Berlin, 1998. Springer.

[69] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 200–209. Morgan Kaufmann Publishers, San Francisco, US, 1999.

[70] S.S. Keerthi and S.K. Shevade. SMO algorithm for least squares SVM formulations. *Neural Computation*, 15:487–507, February 2003.

[71] 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*, 13:637–649, March 2001.

[72] C. Keller, N.I.M Gould, and A.J. Wathen. Constraint preconditioning for indefinite linear systems. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1300–1317, 2000.

[73] D.G. Krige. A statistical approach to some basic mine valuation problems on the witwatersrand. *J. Chem. Metall. Mining Soc. S. Africa*, 52(6):119–139, 1951.

[74] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In G.Tesauro, D.S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. MIT press, 1995.

[75] J.T. Kwok. Support vector mixture for classification and regression problems. In *Proceedings of the International Conference on Pattern Recognition (ICPR98)*, pages 255–258, Brisbane, Australia, 1998.

[76] Y.J. Lee and O.L. Mangasarian. RSVM: Reduced support vector machines. In *Proc. of the First SIAM International Conference on Data Mining*, 2001.

[77] C.J. Lin and R. Saigal. An incomplete cholesky factorization for dense symmetric positve definite matrices. *BIT*, 40:536–558, 2000.

[78] K.M. Lin and C.J. Lin. A study on reduced support vector machines. *IEEE transactions on Neural Networks*, 14(6):1449 – 1459, November 2003.

[79] L. Ljung. *System Identification: Theory for the User (2nd Ed)*. Prentice Hall, New Jersey, 1999.

[80] Lukas. *Least Squares Support Vector Machines Classification Applied to Brain Tumour Recognition using Magnetic Resonance Spectroscopy*. PhD thesis, Departement of Electrical Engineering ESAT-SCD, Katholieke Universiteit Leuven, 2003.

[81] D.J.C. MacKay. Introduction to gaussian processes. In C.M. Bishop, editor, *Neural networks and machine learning*, volume 168 of *NATO-ASI Series F*, pages 133–165. Springer, computer and systems sciences edition, 1998.

[82] O.L. Mangasarian and D.R. Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10:1032–1037, 1999.

[83] O.L. Mangasarian and D.R. Musicant. Lagrangian support vector machines. *Journal of Machine Learning Research*, 1:161–177, 2001.

[84] D. Matterra 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*, pages 211–241. MIT Press, 1999.

[85] R. Meir. Bias, variance and the combination of least-squares estimators. In *Advances in Neural Information Processing Systems 7*, pages 295–302, 1994.

[86] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, London*, A 209:415–446, 1909.

[87] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and R. Müller. Fisher discriminant analysis with kernels. In *Proceedings of IEEE Neural Networks for Signal Processing Workshop IX*, volume Neural Networks for Signal Processing, pages 41–48, 1999.

[88] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[89] E.H. Moore. On properly positive Hermitian matrices. *Bull. Amer. Math. Soc.*, 23(59), 1916.

[90] S. Mukherjee, P. Niyogi, T. Poggio, and R. Rifkin. Statistical learning: Well-posedness is necessary and sufficient for consistency of empirical risk minimization. Technical Report 223, CBCL, Massachusetts Institute of Technology, 2003.

[91] S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear prediction of chaotic time series using a support vector machine. In *IEEE Workshop on Neural Networks for Signal Processing VII*, page 511. IEEE Press, 1997.

[92] S. Mukherjee, P. Tamayo, P. Mesirov, J.P. Slonim, A. Verri, and T. Poggio. Support vector machine classification of microarray data. Technical Report CBCL paper 182/ AI memo 1676, MIT, Cambridge MA, 1999.

[93] S. Mukkamala, G. Janoski, and A. Sung. Intrusion detection using neural networks and support vector machines. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, volume 2, pages 1702 – 1707, 2002.

[94] K.R. Müller, A. Smola, G. Rätsch, B. Schölkopf, and J. Kohlmorgenand V. Vapnik. Predicting time series with support vector machines. In *Proceedings ICANN'97*, page 999. Springer Lecture Notes in Computer Science, 1997.

[95] D.E. Myers. Kriging, cokriging, radial basis functions and the role of positive definiteness. *Computers Math. Applic.*, 24(12):139–148, 1992.

[96] S.G. Nash and A. Sofer. *Linear and Nonlinear Programming.* McGraw-Hill Industrial Engineering Series, 1996.

[97] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer-Verlag, New York, 1999.

[98] E.J. Nyström. Uber die praktische auflosung von linearen integralgleichungen mit anwendungen auf randwertaufgaben der potentialtheorie. *Commentationes PhysicoMathematica*, 4(15):1–52, 1928.

[99] D.P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra and its Applications*, 29:293–322, 1980.

[100] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In Principe J., Giles L., Morgan N.,

and Wilson E., editors, *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, pages 276–285, New York, 1997.

[101] B. Parmanto, P. W. Munro, and H. R. Doyle. Improving committee diagnosis with resampling techniques. In MIT Press, editor, *Advances in Neural Information Processing Systems*, volume 8, 1996.

[102] P. Pavlidis, J. Weston, J. Cai, and W. Noble Grundy. Gene functional classification from heterogeneous data. In *Proceedings of the Fifth International Conference on Computational Molecular Biology*, pages 242–248, 2001.

[103] D. Pavlov, J. Mao, and B. Dom. Scaling-up support vector machines using boosting algorithm. In *Proceedings of the International Conference on Pattern Recognition, ICPR-2000*, 2000.

[104] K. Pelckmans, J.A.K. Suykens, T. Van Gestel, J. De Brabanter, L. Lukas L., B. Hamers, B. De Moor, and J. Vandewalle. LS-SVMlab : a Matlab/C toolbox for Least Squares Support Vector Machines. Technical Report 02-44, ESAT-SISTA, Katholieke Universiteit Leuven, Belgium, 2002. (presented at NIPS2002 Vancouver in the demo track).

[105] M.P. Perrone. *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization*. PhD thesis, Department of Physics at Brown University, May 1993.

[106] M.P. Perrone and L.N. Cooper. When networks disagree: Ensemble method for neural networks. In R.J. Mammone, editor, *Neural Networks for Speech and Image processing*, pages 126–142. Chapman-Hall., 1993.

[107] J. 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, 1999.

[108] T. Poggio and F. Girosi. Networks for approximation and learning. In *Proceedings of the IEEE*, volume 78 (9), pages 1481–1497, 1990.

[109] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

[110] T. Poggio, S. Mukherjee, R. Rifkin, A. Rakhlin, and A. Verri. b. In *Proceedings of the Conference on Uncertainty in Geometric Computations*, 2001.

[111] G. Rätsch. *Robust Boosting via Convex Optimization: Theory and Applications*. PhD thesis, University of Potsdam, 2001.

[112] R. Rifkin. *Everything Old Is New Again: A Fresh Look at Historical Approaches in Machine Learning.* PhD thesis, MIT, 2002.

[113] R. Rifkin, G. Yeo, and T. Poggio. Regularized least-squares classification. In J.A.K. Suykens, G. Horvath, S. Basu, C. Micchelli, and J. Vandewalle, editors, *Advances in Learning Theory: Methods, Models and Applications*, volume 190 of *NATO Science Series III: Computer and Systems Sciences*, pages 131–153. IOS Press Amsterdam, 2003.

[114] M. Rozloznik and V. Simoncini. Krylov subspace methods for saddle point problems with indefinite preconditioning. *SIAM Journal On Matrix Analysis and Applications*, 24(2):368–391, 2002.

[115] C. Rudin. A different type of convergence for statistical learning algorithms. Technical report, PACM, Princeton, 2003. www.math.princeton.edu/ crudin/StabilityPaperShortVersionMar72003.pdf.

[116] C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithms in dual variables. In *Proc. of the 15th Int. Conf. on Machine Learning (ICML-98)*, pages 515–521, Madison-Wisconsin, 1998.

[117] R.E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[118] R.E. Schapire. The boosting approach to machine learning: An overview. In MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA, March 2001.

[119] B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA., 2002.

[120] A. Schwaighofer and V. Tresp. The Bayesian committee support vector machine. In *Proceedings of ICANN 2001*, number 2130 in Lecture Notes in Computer Science, pages 411–417. Springer Verlag, 2001.

[121] A. Schwaighofer and V. Tresp. Transductive and inductive methods for approximate gaussian process regression. In *Advances in Neural Information Processing Systems*, volume 15, 2003.

[122] J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, School of Computer Science, Carnegie Mellon University, August 1994. http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.ps.

[123] Y. Singer. Multiclass learning with output codes. In J.A.K. Suykens, G. Horvath, S. Basu, C. Micchelli, and J. Vandewalle, editors, *Advances in Learning Theory: Methods, Models and Applications*, volume 190 of *NATO Science Series III: Computer and Systems Sciences*, pages 251–266. IOS Press Amsterdam, 2003.

[124] A.J. Smola. *Learning with Kernels*. PhD thesis, Technische Universität Berlin, 1998.

[125] A.J. Smola and B. Schölkopf. A tutorial on support vector regression. Technical report, ESPRIT Working group in Neural and Compuational Learning II, 1998. NeuroCOLT2 Series NC2-TR-1998-030.

[126] A.J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *International Conference on Machine Learning*, 2000.

[127] P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful. In *Advances in Neural Information Processing Systems*, volume 8, pages 190–196. The MIT Press, 1996.

[128] J.A.K. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle. Weighted least squares support vector machines: Robustness and sparse approximation. *Neurocomputing*, 48:85–105, 2002.

[129] J.A.K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific, Singapore, 2002.

[130] J.A.K. Suykens, T. Van Gestel, J. Vandewalle, and B. De Moor. A support vector machine formulation to pca analysis and its kernel version. *IEEE Transactions on Neural Networks*, 14(2):447–450, 2002.

[131] J.A.K Suykens, P. Van Dooren, B. De Moor, and J. Vandewalle. Least squares support vector machine classifiers: a large scale algorithm. *European Conference on Circuit Theory and Design, ECCTD'99*, pages 839–842, 1999.

[132] J.A.K Suykens and J. Vandewalle. Least squares support vector machine classifier. *Neural Processing Letters*, 9(3):293–300, Jun 1999.

[133] J.A.K Suykens and J. Vandewalle. Multiclass least squares support vector machines. In *IJCNN'99 International Joint Conference on Neural Networks*, Washington, DC, 1999.

[134] J.A.K. Suykens, J. Vandewalle, and B. De Moor. Intelligence and cooperative search by coupled local minimizers. *International Journal of Bifurcation and Chaos*, 11(8):2133–2144, 2001.

[135] J.A. Swets. ROC analysis applied to the evaluation of medical imaging techniques. *Investigative Radiology*, 14(2):109–121, 1979.

[136] J.A. Swets. Measuring the accuracy of diagnostic systems. *Science*, 240(4857):1285–1293, 1988.

[137] A.N. Tikhonov and V. Yarsenin. *Solution of Ill-Posed Problems*. Winston, Washington DC, 1977.

[138] H. Tong. *Non-linear Time Series. A dynamical Systems Approach*. Oxford Science Publications, 1993.

[139] L.N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[140] M. Unser. Splines: a perfect fit for signal and image processing. *IEEE Signal Processing Magazine*, 16(6):22–38, 1999.

[141] 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, January 2004.

[142] T. Van Gestel, J.A.K. Suykens, D.E. Baestaens, A. Lambrechts, G.R.G Lanckriet, B.Vandaele, B. De Moor, and J. Vandewalle. Financial time series prediction using least squares support vector machines within the evidence framework. *IEEE Transactions on Neural Networks*, 12(4):809–821, 2001. (special issue on Neural Networks in Financial Engineering).

[143] T. Van Gestel, J.A.K. Suykens, D.E. Baestaens, A. Lambrechts, G.R.G Lanckriet, B. Vandaele, B. De Moor, and J. Vandewalle. Bayesian interpretation of least squares support vector machines for financial time series prediction. In *Proc. of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI)*, pages 254–259, Orlando, Florida, 2001.

[144] T. Van Gestel, J.A.K. Suykens, G. Lanckriet, A. Lambrechts, B. De Moor, and J. Vandewalle. Multiclass LS-SVMs: Moderated outputs and coding-decoding schemes. *Neural Processing Letters*, 15(1):45–58, 2002.

[145] V. Vapnik. *Estimations of Dependences Based on Empirical Data.* Springer-Verlag, Berlin, 1982.

[146] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, 1995.

[147] V. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998.

[148] V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. *Advances in Neural Information Processing Systems*, 9:281–287, 1997.

[149] R.S. Varga. *Matrix Iterative Analysis.* Prentice-Hall, Englewood Cliffs, 1962.

[150] G. Wahba. *Spline Models for Observational Data*, volume 59 of *Series in Applied Mathematics.* SIAM, Philadelphia, 1990.

[151] C.K.I. Williams, C.E. Rasmussen, A. Schwaighofer, and V. Tresp. Observations on the Nyström method for gaussian process prediction. Technical report, Institute for Adaptive and Neural Computation, School of Informatics, University of Edinburgh, 2002.

[152] C.K.I. Williams and M. Seeger. The effect of the input density distribution on kernel-based clasifiers. In *Proceedings of the Seventeenth Internalional Conference on Machine Learning*, 2000.

[153] C.K.I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T.K. Leen and T.G. Diettrich ans V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2001.

[154] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[155] M. Yamana, H. Nakahara, M. Pontil, and S. Amari. On different ensembles of kernel machines. In *European Symposium on Artificial Neural Networks (ESANN)*, pages 197–201, Bruges, Belgium, 2003.

[156] D.M. Young. *Iterative solution of large linear systems.* Academic Press, New York, 1971.

Curriculum Vitae

Bart Hamers was born May 3, 1974 in Maaseik, Belgium.

He received his Bachelor degree in Physics form the Limburgs Universitair Centrum, Diepenbeek, Belgium at June 1995.

Afterwards he continued his studies at the Katholieke Universiteit Leuven. His main subject was Nuclear Physics and the Master's thesis was devoted to '*Study of tensor activity in nuclear β-decay by means of Nuclear Orientation methods*' finished at the Institute of Nuclear and Radiation Physics (IKS) under the supervision of prof. Dr. Natal Severijns. At June 1997 he received his Master (Licentiaat, MSc) degree in Physics.

In September 1997 he started his studies in Artificial Intelligence at the Katholieke Universiteit Leuven with as main subject Engineering. The Master's thesis was titled: '*A neural network model of grouping by proximity in dot lattices*' under the supervision of prof. dr. ir. Marc Van Hulle (Lab of Neuro and Psychofysiology, KUL) and prof. dr. Johan Wagemans (Department of Psychology, KUL). At June 1999 he received his Advanced Master of Artificial Intelligence degree.

In August 1999 he started working as a researcher at the Electrical Engineering Department ESAT in the Lab of Signals, Identification, System Theory and Automation SCD/SISTA at the Katholieke Universiteit Leuven. After first working on a KULeuven grant he became a Fellow of the Instituut voor de aanmoediging van Innovatie door Wetenschap en Technologie in Vlaanderen (IWT), which offered a PhD grant from January 2000 until December 2003.

List of Publications

Conference publications

- B. Hamers, J.A.K Suykens, B. De Moor, *Compactly supported RBF kernels for sparsifying the Gram matrix in LS-SVM regression models,* Proceedings International Conference of Artificial Neural Networks, Madrid Spain, 2002.
- B. Hamers, J.A.K. Suykens, V. Leemans, B. De Moor, *Ensemble Learning of Coupled Parameterized Kernel Models.,* Proceedings International Conference of Artificial Neural Networks, Istanbul Turkey, 2003.

Manuscripts submitted for publication

- B. Hamers, J.A.K Suykens, B. De Moor., *Coupled Transductive Ensemble Learning of Kernel Models*, Internal Report 03-172, ESAT-SISTA, K.U.Leuven (Leuven, Belgium), 2003.

Internal Reports

- B. Hamers, J.A.K Suykens, B. De Moor, *A comparison of iterative methods for least squares support vector machine classifiers*, Internal Report 01-110, ESAT-SISTA, K.U.Leuven (Leuven, Belgium), 2001.
- K. Pelckmans, J.A.K. Suykens, T. Van Gestel, J. De Brabanter, L. Lukas, B. Hamers, B. De Moor, J. Vandewalle, *LS-SVMlab : a Matlab/C toolbox for Least Squares Support Vector Machines*, Internal Report 02-44, ESAT-SISTA, K.U.Leuven (Leuven, Belgium), 2002. (presented at demo track NIPS 2002)

Abstracts

- B. Hamers, J.A.K Suykens, B. De Moor, *Ensemble Learning with Output Synchronization,* 22nd Benelux Meeting on Systems and Control, Lommel, Belgium, 2003.
- B. Hamers, J.A.K Suykens, B. De Moor, *Kernel Methods for Large Scale Data Mining Applications,* First Flanders Engineering PhD Symposium, Brussels, Belgium, 2003.

List of presentations

- B. Hamers, J.A.K Suykens, B. De Moor, *Large Scale Data Mining using Convex Optimization,* IUAP study day, Louvain-la-Neuve, 1999.
- K. Marchal, B. Hamers, J. Mathys, F. De Smet , G. Thys, Y. Moreau, J. Suykens, B. De Moor, *Extension of high quality clusters by combined use of SVM and motif finding*, Belgian Bioinformatics Conference, Gent, Belgium, 2001.
- B. Hamers, J.A.K Suykens, B. De Moor, *Compactly supported RBF kernels for sparsifying the Gram matrix in LS-SVM regression models,* IUAP study day, Leuven, Belgium, 2002.
- B. Hamers, J.A.K Suykens, B. De Moor, *LS-SVM for Large Scale Applications*, NATO Advanced Study Institute on "Learning Theory and Practice" (LTP 2002), Leuven, Belgium, 2002.
- B. Hamers, J.A.K Suykens, V. Leemans, B. De Moor, *Compactly supported RBF kernels for sparsifying the Gram matrix in LS-SVM regression models*, International Conference of Artificial Neural Networks, Madrid, Spain, 2002.
- B. Hamers, J.A.K Suykens, B. De Moor, *Ensemble Learning with Output Synchronization,* 22nd Benelux Meeting on Systems and Control, Lommel, Belgium, 2003.
- B. Hamers, J.A.K. Suykens, V. Leemans, B. De Moor, *Ensemble Learning of Coupled Parameterized Kernel Models,* International Conference of Artificial Neural Networks, Istanbul Turkey, 2003.
- B. Hamers, J.A.K Suykens, B. De Moor, *Ensemble Learning with Output Synchronization,* IUAP study day, Louvain-La-Neuve, Belgium, 2003.
- B. Hamers, J.A.K Suykens, B. De Moor, *Large Scale Data Mining Applications,* First Flanders Engineering Ph.D. Symposium, Brussels, Belgium, 2003.