Universiteit Antwerpen
Faculteit Wetenschappen
Departement Wiskunde-Informatica
Academiejaar 2005-2006

Studie van kernel-gebaseerde technieken voor single-class classificatie en feature selectie op basis van optimalisatie van de kernel parameters

— *Koen Smets* —

Eindwerk ingediend met het oog op het behalen
van de graad van Licentiaat in de Wetenschappen

Richting: Wiskunde (optie Informatica)

Promotor: Prof. Dr. Brigitte Verdonk
Copromotor: Dr. Piet van Remortel

Dankwoord

Graag wens ik enkele personen te bedanken die deze eindverhandeling mede mogelijk maakten.

Eerst en vooral uiteraard mijn promotoren, professor Brigitte Verdonk en Piet van Remortel, die me carte blanche gaven om zelfstandig de wondere wereld van de support vector machines uit te diepen en me daar waar nodig bijstuurden met opbouwende kritiek. De eerst genoemde wil ik graag extra bedanken voor het aanreiken van enkele basisbegrippen uit de optimalisatietheorie in de cursus "operationeel onderzoek" waaruit ik kon putten bij het bestuderen van het support vector algoritme.

In het kader van de uitwerking van de experimentele resultaten wil ik Koen Van Leemput bedanken voor de vrijgemaakte tijd om het gebruik van random grafen toe te lichten en de uitnodiging om de resultaten op een meer gestructureerde wijze in de verf te zetten. Daarnaast een woord van dank aan professor Chih-Jen voor het overhandigen van een door één van zijn studenten, met name Yi-Wei Chen, geschreven tool als uitbreiding van LIBSVM, die ik als basis kon gebruiken bij het testen van verschillende feature selectie algoritmen. Ook Elsa Jordaan dank ik voor het leerrijke bezoek aan Dow in Terneuzen, en tegelijkertijd wil ik me bij haar langs deze weg verontschuldigen voor het geen gevolg geven aan de analyse van de door haar aangereikte regressie data sets.

Verder ben ik de docenten van de boeiende cursus "taaltechnologie", professor Walter Daelemans en Antal van den Bosch, dankbaar om met voldoende oog voor detail verschillende machine learning technieken in het kader van natuurlijke taal verwerking uit de doeken te doen, wat zeker een positieve bijdrage leverde tijdens mijn verkenningstocht van het machine learning domein.

Dit alles zou echter niet mogelijk zijn, mocht ik me nu niet in het laatste jaar licentiaat wiskunde bevinden. Ik wens hiervoor mijn ouders te bedanken voor de kans die ze me hebben gegeven om mezelf te ontplooien, alsook mijn zus en vrienden voor alle steun en broodnodige ontspanning tijdens mijn studies.

Bedankt!

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van algoritmes

Woord vooraf

Op de zoektocht naar een geschikt onderwerp binnen de wereld van machine learning, werd ik al snel geconfronteerd met de *state-of-the-art* support vector machines (SVM) en ermee gepaard gaande methodologie van kernel methoden.

Vorig jaar werd reeds in het eindwerk van Pieter Wellens [71] een vlot leesbare introductie neergezet tot het support vector machine algoritme voor binaire/multi-class classificatie en de mogelijke uitbreiding besproken om regressietaken te verrichten.

Persoonlijke interesse – netwerk veiligheid en de daarbij nauw samenhangende intrusion detection – bracht al snel het onontgonnen gebied van de outlier/novelty detectie met behulp van kernel methoden aan het licht, waardoor de single-class classificatie beschreven in Hoofdstuk 2 zijn recht van bestaan kreeg. Spijtig genoeg bereikte ik bij uitvoering van experimenten in een prille beginfase slechts in uitzonderlijke gevallen de resultaten beschreven in de geraadpleegde literatuur en werd al snel duidelijk dat verwerving van inzicht in de keuze van de geschikte modelparameters, kortom modelselectie, cruciaal is én, spijtig genoeg, nog steeds een open probleem vormt. Modelselectie wordt bovendien in het specifieke geval van de single-class methode nog extra bemoeilijkt doordat we enkel gebruik maken van "goede" voorbeelden en slechts in uitzonderlijke gevallen beschikken over sporadische afwijkingen.

Dankzij het heersende verband tussen single-class technieken en binaire classificatie kunnen we echter de intuïtieve verklaring van de parameters overnemen, met enkele heuristieken om deze af te stellen tot gevolg. Bovendien laat dit verband ons eveneens toe, de theoretische resultaten van het intensiever bestudeerde classificatie probleem over te nemen.

Vandaar dat we in Hoofdstuk 3 terugvallen op binaire classificatie en we een aantal methoden beschouwen om de meest optimale keuze van de parameters, zowel van het algoritme als van de (gewogen) kernels, te bepalen. De grootste aandacht gaat uit naar methoden die afstappen van een brute force zoekstrategie en deze vervangen door computationeel aantrekkelijkere

algoritmen afkomstig uit de wereld van operationeel onderzoek en optimalisatietechnieken.

Aan de keuze van de geschikte modelparameters gaat in de praktijk vaak een belangrijke preprocessing stap vooraf: de keuze van de input variabelen. Eén van de vele slogans van SVMs, of kernel methoden in het algemeen, is dat zij uitstekend in staat zijn met data van ogenschijnlijk hoge dimensionaliteit om te gaan. In Hoofdstuk 4 geven we echter na hoe we de prestaties van de geconstrueerde modellen in de praktijk kunnen opdrijven door een juiste selectie van de input variabelen. We bestuderen hiervoor twee technieken, SVM-RFE (*Recursive Feature Elimination*) en *scaled feature selection*, die beide berusten op specifieke karakteristieken van de support vector methodologie en waarbij de laatst genoemde als het ware een terugkoppeling vormt met de algoritmen besproken in Hoofdstuk 3 om de gewichtsfactoren van een gewogen kernel te bepalen.

Tot slot beschrijft Hoofdstuk 5 hoe de in dit eindwerk besproken methoden voor enerzijds het optimaliseren van de parameters en anderzijds het selecteren van de input variabelen zich gedragen in een aantal concrete toepassingen.

Hoofdstuk 1

Kennismaking met SVMs

In dit hoofdstuk voeren we aan de hand van een *kernelisering* van het eenvoudige perceptron algoritme, een aantal begrippen kenmerkend voor kernel methoden in, zodat uitdrukkingen zoals "het impliciet werken in een feature ruimte gebruikmakende van een kernel functie" naderhand als betekenisvolle muziek in de oren gaat klinken en leggen we de noodzakelijke basis om het support vector algoritme in te leiden.

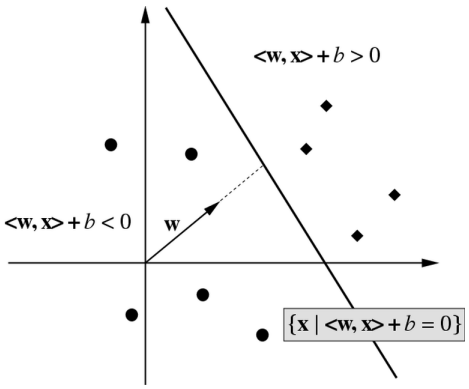
1.1 Lineaire classifiers

Vooraleer we van start gaan, leggen we een aantal notaties vast die ook in de rest van dit werk gehanteerd zullen worden. Aangezien we werken in een gesuperviseerde leeromgeving, beschikken we over een aantal trainingsvoorbeelden,

$$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\} \in \mathcal{X} \times \mathcal{Y},$$

uit een input ruimte $\mathcal{X}$, die we voor de eenvoud beschouwen als reële n-dimensionale vectoren $\boldsymbol{x}_i$, samen met hun output label y_i. De output ruimte $\mathcal{Y}$ reduceert zich in het geval van binaire classificatie tot $\{\pm 1\}$, terwijl we bij regressieproblemen een willekeurige reële waarde als etiquette voorzien.

We zullen ons in deze introductie bezig houden met de meest intuïtieve vorm van modellering, namelijk binaire classificatie. Met behulp van een *linear learning machine* (LLM), voeren we een classificatie uit door, zoals aangegeven op Figuur 1.1, een hypervlak – in twee dimensies, een rechte – te construeren die de input ruimte zodanig opdeelt dat het de klassen van elkaar scheidt. Wanneer we uiteindelijk een nieuw object ter classificatie aanbieden, bepalen we aan welke zijde van het hypervlak dit punt terecht komt en kennen dan het overeenstemmende label toe.



Figuur 1.1: [56] Lineaire scheiding in een twee dimensionale input ruimte.

Gebuikmakende van het canonieke inproduct,

$$\langle \boldsymbol{x}, \boldsymbol{z} \rangle = \sum_{i=1}^{n} x_i z_i,$$

kunnen we de vergelijking van het hypervlak,

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0,$$

en beslissingsfunctie,

$$f(\boldsymbol{x}) = \operatorname{sgn}\Big(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\Big), \tag{1.1}$$

op een bondige manier noteren. Binaire classificatie reduceert zich in dit geval tot het aanleren van de coëfficiënten van de normaal vector $\boldsymbol{w}$ en de threshold term b.

Alvorens we ons storten in de wereld van support vector machines, is het nuttig om de nodige terminologie in te voeren door een wellicht bekend en eenvoudig lineair algoritme, het perceptron algoritme, te analyseren.

Algoritme 1 toont het algoritme in zijn meest eenvoudige vorm, waarbij we enkel de gewichtsvector $\boldsymbol{w}$ berekenen en de bias term b achterwege laten[1]. Van dit algoritme weten we [32] dat het, wanneer onze training set lineair te scheiden is, steeds convergeert en in dat geval een oplossing voor het classificatie probleem biedt.

Belangrijk is op te merken dat de aanpassing aan de gewichtsvector zo gebeurt dat deze uiteindelijk te schrijven is als een lineaire combinatie van de

[1]De enige aanpassing die men eigenlijk moet maken, is het uitbreiden van elke vector met een component $+1$ waardoor we de threshold term b kunnen opnemen binnen het scalair product.

Algoritme 1 Perceptron algoritme – primaire voorstelling

$\boldsymbol{w} \leftarrow \mathbf{0}$
repeat
  errors $\leftarrow 0$
  for $i = 1 \ldots m$ **do**
    if $y_i\Big(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle\Big) \leq 0$ **then**
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y_i \boldsymbol{x}_i$
      errors $\leftarrow$ errors $+ 1$
    end if
  end for
until errors $= 0$

trainingsvoorbeelden $\boldsymbol{x}_i$:

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i. \tag{1.2}$$

Dit laat ons toe om zowel het algoritme als de beslissingsfunctie te herformuleren en zo de zogenaamde *duale voorstelling* te bekomen. Substitutie van (1.2) in (1.1) levert ons volgende beslissingsfunctie op

$$f(x) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = \sum_i \alpha_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + b. \tag{1.3}$$

We kunnen de update regel eveneens uitdrukken in functie van de α_i's,

$$\alpha_i \leftarrow \alpha_i + \eta, \tag{1.4}$$

wat aanleiding geeft tot Algoritme 2.

Algoritme 2 Perceptron algoritme – duale voorstelling

$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
repeat
  errors $\leftarrow 0$
  for $i = 1 \ldots m$ **do**
    if $y_i\Big(\sum_j \alpha_j y_j \langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle\Big) \leq 0$ **then**
      $\alpha_i \leftarrow \alpha_i + \eta$
      errors $\leftarrow$ errors $+ 1$
    end if
  end for
until errors $= 0$

Het belang van de duale representatie zal naarmate dit hoofdstuk vordert steeds duidelijker in de verf worden gezet. Voorlopig volstaat het op te merken dat we nu beschikken over een algoritme waar de training data $\boldsymbol{x}_i$

enkel binnen het scalair product voorkomen. Voorts kunnen we ook het concept *sparseness* van de oplossing uit de doeken doen. Immers, wanneer een object in de training fase nooit aan de verkeerde kant van het hypervlak belandt, is de overeenstemmende coëfficiënt α_i nog steeds gelijk aan nul en heeft dit object geen invloed op de beslissing en kan dus achterwege gelaten worden in de beschrijving van het uiteindelijke model.

Omdat de leerconstante η in (1.4) positief is, zijn de coëfficiënten α_i positief. Bovendien kunnen we een meer gevoelsmatige betekenis van "moeilijkheidsgraad" toekennen, aangezien de grootte van zulke α_i enkel afhangt van het aantal keer dat we $\boldsymbol{w}$ moeten aanpassen ten gevolge van een misclassificatie van het trainingsvoorbeeld $\boldsymbol{x}_i$.

1.2 Niet-lineaire classifiers

Het spreekt voor zich dat de aldus geconstrueerde lineaire classifier enkel bruikbaar is voor klassen die lineair van elkaar te scheiden zijn en dat de generalisatie eigenschappen sterk worden teruggedrongen wanneer outliers, eventueel te wijten aan ruis op de data, in de training set aanwezig zijn.

In de wereld van de neurale netwerken (NN) wordt het probleem van niet-lineair separabele data opgelost door meerdere eenvoudige perceptronen (*units*) op een gepaste wijze met elkaar te verbinden. Moeilijkheden die men dan moet overwinnen, is het ontwerpen van de juiste netwerk topologie: het netwerk mag noch te klein zijn, omdat we dan niet in staat zijn complexe beslissingen te nemen, noch te groot zijn, omdat we dan mogelijk over een netwerk beschikken dat overgevoelig is aan ruis en dat door het grote aantal parameters makkelijk overfit tijdens de training.

Een andere oplossing – en tevens ook diegene die we ook in het vervolg zullen aanmoedigen – gaat ervan uit dat elk probleem intrinsiek een lineair probleem is en dat het enkel een kwestie is van op de juiste manier naar de data te kijken. Het idee is dat door de dataset met (niet-)lineaire features uit te breiden, we wel in staat zullen zijn om de data met behulp van een lineaire classifier op te delen.

Formeel houdt dit in dat we de data preprocessen door deze met behulp van een *feature mapping* af te beelden op een (hogere) dimensionale ruimte $\mathcal{H}$,

$$\begin{aligned} \Phi : \mathcal{X} &\to \mathcal{H} \\ \boldsymbol{x} &\mapsto \Phi(\boldsymbol{x}), \end{aligned}$$

waarbij de feature ruimte $\mathcal{H}$ een Hilbertruimte is. Kortom, een ruimte uitgerust met een scalair product.

Algoritme 3 Perceptron algoritme in feature ruimte – duale voorstelling

1: $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
2: **repeat**
3: errors $\leftarrow 0$
4: **for** $i = 1 \dots m$ **do**
5: **if** $y_i\Big(\sum_j \alpha_j y_j \langle \Phi(\boldsymbol{x}_j), \Phi(\boldsymbol{x}_i)\rangle\Big) \leq 0$ **then**
6: $\alpha_i \leftarrow \alpha_i + \eta$
7: errors $\leftarrow$ errors $+ 1$
8: **end if**
9: **end for**
10: **until** errors $= 0$

Het enige wat we moeten veranderen aan het perceptron algoritme, is dat we niet langer de scheiding uitvoeren in de input ruimte, maar Algoritme 3 laten runnen in de feature ruimte $\mathcal{H}$.

Een verduidelijkend voorbeeld is hier wel op zijn plaats. Op Figuur 1.2 zien we dat we de bolletjes van de kruisjes kunnen scheiden door er een ellips rond te tekenen. Om dit te realiseren maken we gebruik van volgende transformatie,

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

waarbij de nieuwe features bekomen worden door het onderlinge product te nemen van de input variabelen. De data is in deze ruimte wel lineair te scheiden en door Algoritme 3 daar los te laten, vinden we volgende vergelijking als hypervlak,

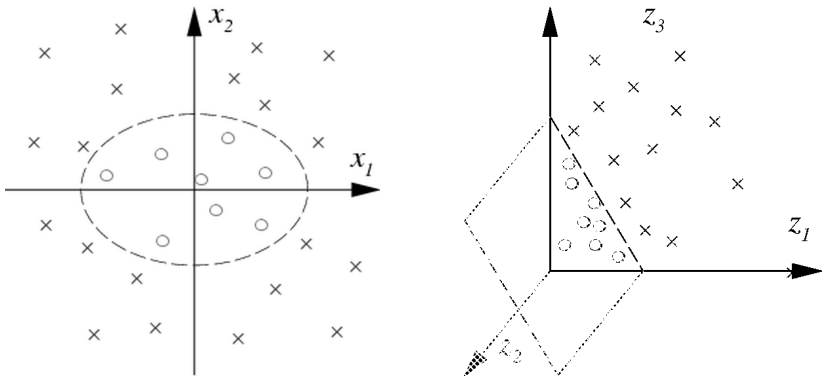
$$az_1 + 0z_2 + bz_3 = c,$$

aangezien het vlak evenwijdig is met de Z_2-as. Wanneer we deze vergelijking terugtrekken naar de input ruimte, zien we de vergelijking van een ellips verschijnen

$$ax_1^2 + bx_2^2 = c.$$

Dit fenomeen is te veralgemenen: wanneer de data in een voldoende hoge dimensionale ruimte wordt gemapped, dan zal deze steeds lineair separabel zijn. Wanneer we beschikken over m trainingsvoorbeelden zijn deze steeds in $m - 1$ of hoger dimensionale ruimte te scheiden door een hypervlak[2].

[2]Meer formeel: men kan aantonen dat VC (Vapnik Chervonenkis) dimensie van een hypervlak in een m-dimensionale ruimte gelijk is aan m+1, omdat we uit de beschouwde verzameling van hypothese functies (in dit geval alle hypervlakken) voor elke van de 2^m mogelijke toekenningen van labels een hypervlak kunnen vinden dat de data zonder fouten classificeert (Cover's theorem [56]).



Figuur 1.2: [56] Binaire classificatie in input en feature ruimte.

Willen we niet enkel kwadratische scheidingen, dan kan men proberen gebruik te maken van een hogere orde product mapping. Wanneer we echter kijken naar het aantal verschillende monomen van graad d dat we kunnen construeren, vertrekkende van een input ruimte van dimensie n, is dit gelijk aan

$$\binom{d+n-1}{d} = \frac{(d+n-1)!}{d!(n-1)!}$$

en dus van orde n^d.

Dit legt meteen een pijnpunt bloot: willen we bijvoorbeeld de resultaten uit [53] behalen waarbij het experiment er uit bestond om handgeschreven cijfers van de US Postal Service (USPS) data set te herkennen, dan zouden de feature vectoren, gevormd door telkens 5 input variabelen afkomstig uit $16 \times 16 = 256$ pixels uit de zwart/wit figuren te combineren, om en bij de 10^{10} componenten bevatten. Wat natuurlijk niet handelbaar is in de praktijk.

1.3 Kernel-induced feature space

Om het computationele probleem, dat gepaard gaat met het werken in zulke hoge (en mogelijk oneindig) dimensionale ruimten, te overwinnen, is men kernel functies gaan beschouwen.

Van zulke functies verwachten we dat ze gegeven twee objecten (hoeven geen vectoren meer te zijn!) de waarde van het scalair product in de feature ruimte uitdrukken. Kortom,

$$K(\boldsymbol{x}, \boldsymbol{z}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{z}) \rangle.$$

Wanneer we over zo'n – hopelijk efficiënt uit te rekenen – functie K beschikken, kunnen we het perceptron algoritme *kerneliseren*, door de *kernel*

trick toe te passen, i.e. door overal het scalair product te vervangen door de evaluatie van de kernel functie:

$$\langle \boldsymbol{x}, \boldsymbol{z} \rangle \leftarrow K(\boldsymbol{x}, \boldsymbol{z}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{z}) \rangle.$$

Algoritme 4 geeft de finale versie van het Kernel Perceptron algoritme aan.

Algoritme 4 Kernel-perceptron algoritme – duale voorstelling

```
1: α ← 0
2: repeat
3:    errors ← 0
4:    for i = 1 ... m do
5:       if y_i ( Σ_j α_j y_j K(x_j, x_i) ) ≤ 0 then
6:          α_i ← α_i + η
7:          errors ← errors + 1
8:       end if
9:    end for
10: until errors = 0
```

Deze eenvoudige kunstgreep vormt hét centrale, achterliggende concept en dé kracht bij kernel methoden: elk algoritme waarbij de data enkel binnen inproducten voorkomt, of anders gezegd, geschreven kan worden in duale representatie, kunnen we van een stel niet-lineaire varianten voorzien. Onder deze noemer vallen vele meetkundige algoritmen die enkel gebruik maken van scalaire producten of nauw verwante concepten afstand of loodrechte projecties, zoals bijvoorbeeld Principal Component Analysis (PCA).

Een andere centrale structuur die we met kernels kunnen associëren is de Gram matrix (of kernel matrix),

$$\boldsymbol{K} = \begin{bmatrix} K(\boldsymbol{x}_1, \boldsymbol{x}_1) & K(\boldsymbol{x}_1, \boldsymbol{x}_2) & \dots & K(\boldsymbol{x}_1, \boldsymbol{x}_m) \\ K(\boldsymbol{x}_2, \boldsymbol{x}_1) & K(\boldsymbol{x}_2, \boldsymbol{x}_2) & \dots & K(\boldsymbol{x}_2, \boldsymbol{x}_m) \\ \vdots & \vdots & \ddots & \vdots \\ K(\boldsymbol{x}_m, \boldsymbol{x}_1) & K(\boldsymbol{x}_m, \boldsymbol{x}_2) & \dots & K(\boldsymbol{x}_m, \boldsymbol{x}_m) \end{bmatrix}.$$

Met dit begrip kunnen we, zonder in detail te treden, de voor kernel methoden geschikte verzameling van kernel functies definiëren, met name de positief definiete kernels.

Definitie 1.1 (pd kernel)**.** *Beschouw een niet-lege verzameling $\mathcal{X}$. Een functie K op $\mathcal{X} \times \mathcal{X}$, die voor elke $m \in \mathbb{N}$ en voor $\boldsymbol{x}_1, \dots, \boldsymbol{x}_m$ aanleiding geeft tot een symmetrische (semi) positief definiete Gram matrix $\boldsymbol{K}$, i.e.*

1. $\boldsymbol{K} = \boldsymbol{K}^{\boldsymbol{T}}$
2. $\boldsymbol{c}^T \boldsymbol{K} \boldsymbol{c} \geq 0, \quad \boldsymbol{c} \in \mathbb{R}^m$

wordt een positief definiete (pd) kernel *genoemd.*

De noodzaak om enkel positief definiete kernels te beschouwen, is niet uit de lucht gegrepen, aangezien de definitie van een scalair product postuleert dat we steeds te maken hebben met een positief definiete, symmetrische bilineaire vorm. Per definitie volgt dan onmiddellijk dat wanneer we over een functie Φ beschikken die de inputruimte $\mathcal{X}$ afbeeldt op een Hilbertruimte $\mathcal{H}$, dat dan $K = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{z}) \rangle$ een pd kernel is op $\mathcal{X} \times \mathcal{X}$. Deze (theoretische) kernel is echter van weinig nut in de praktijk omdat we hiermee vanzelfsprekend het computationele probleem niet omzeilen.

Een belangrijke stelling uit de functionaalanalyse, het theorema van Mercer, waarvan een sterk vereenvoudigde versie stelt dat we elke symmetrische, positief definiete matrix kunnen beschouwen als een kernel matrix, en dus als een inproduct matrix uit een of andere Hilbertruimte.

Kernels hebben daarenboven een aantal nuttige (praktische) eigenschappen.

Eigenschap 1.2. *Als K, K' kernels zijn, dan zijn ook*

1. $K + K'$
2. cK, *als* $c > 0$
3. $aK + bK'$, *als* $a, b > 0$

kernels.

Eigenschap 1.2 toont dat de verzameling van kernels gesloten is onder sommige operaties en kernels als het ware modulaire bouwstenen vormen en we dus complexere kernels kunnen construeren door het samennemen van eenvoudigere, met goede praktische resultaten tot gevolg [58].

Ook vanuit praktisch standpunt kan het gebruik van pd kernels gestaafd worden. Zoals zal blijken, zullen de verschillende optimalisatieproblemen, zoals om het optimale hypervlak te construeren bij het support vector algoritme, een convexe objectieve functie hebben waardoor we weten dat elke lokale oplossing tevens ook een globale oplossing voor het probleem is.

Voor een diepgaande bespreking van de eigenschappen, theorema's en de verschillende mappings die we met kernels kunnen associëren, verwijzen we naar het desbetreffende hoofdstuk in [56].

1.4 Voorbeelden van kernels

Een vaak gebruikte familie van kernels zijn de polynomiale kernels van graad d

$$K(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x}, \boldsymbol{z} \rangle^d.$$

Mits een klein beetje rekenwerk kunnen we inzien dat de polynomiale kernel van graad d overeenkomt met de reeds eerder beschouwde mapping naar de ruimte opgespannen door monomen van graad d. De uitwerking voor het specifieke geval $n = d = 2$, gaat als volgt

$$\begin{aligned}\langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{z}) \rangle &= (x_1^2, \sqrt{2}x_1x_2, x_2^2)(z_1^2, \sqrt{2}z_1z_2, z_2^2)^T \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1z_1x_2z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= \langle \boldsymbol{x}, \boldsymbol{z} \rangle^2 \\ &=: K(\boldsymbol{x}, \boldsymbol{z}).\end{aligned}$$

Een andere veelvuldig gehanteerde kernel is de Gaussische Radial Basis Function (RBF) kernel

$$K(\boldsymbol{x}, \boldsymbol{z}) = \exp(\frac{-||\boldsymbol{x} - \boldsymbol{z}||^2}{2\sigma^2}),$$

met parameter σ die de spreiding vastlegt.

Naar mate dit eindwerk vordert, zullen een aantal specifieke verschillen tussen beide type kernels in de verf gezet worden. Belangrijk is op te merken dat beide kernels ons op een efficiënte manier toelaten om impliciet te werken in een hoog dimensionale ruimte.

Wanneer we nu teruggrijpen op het experiment om handgeschreven letters te herkennen, beschreven in [53], kunnen we door het inpluggen van een polynomiale kernel van graad 5 dezelfde resultaten bekomen zonder expliciet de feature vectoren met $\sim 10^{10}$ componenten te construeren.

1.5 Support vector machines

Voor een meer volledige beschrijving van support vector machines (SVMs) verwijs ik in eerste instantie naar het desbetreffende, eenvoudig te verteren hoofdstuk in [71] en beperk in mij hier tot een summiere inleiding om enkele notaties in te voeren waarnaar in het vervolg verwezen kan worden.

In tegenstelling tot het perceptron algoritme, waar we tevreden zijn met het eerste hypervlak dat onze data scheidt, tracht het support vector algoritme het "optimale" hypervlak te construeren. In het geval van separabele data komt dit overeen met het canonieke hypervlak met een maximale *margin* $\rho_{(\boldsymbol{w},b)}$. Dit is voor correct geclassificeerde punten de loodrechte afstand tot het hypervlak, en voor misclassificaties de negatieve afstand. Formeel,

$$\rho_{(\boldsymbol{w},b)}(\boldsymbol{x}, y) = \frac{y\Big(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\Big)}{||\boldsymbol{w}||}.$$

We kunnen het concept margin uitbreiden voor meerdere punten door het minimum te nemen. In het geval van onze training set herleidt dit zich tot

$$\rho_{(\boldsymbol{w},b)} = \min_{i=1,\ldots,m} \rho_{(\boldsymbol{w},b)}(\boldsymbol{x}_i, y_i).$$

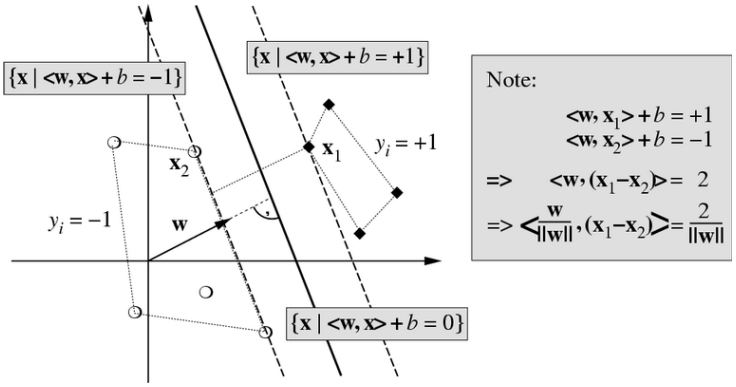
Door te vereisen dat $(\boldsymbol{w}, b)$ zo geschaald wordt dat de punt(en) die het dichtst bij het hypervlak liggen, voldoen aan de voorwaarde

$$|\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b| = 1,$$

bekomen we de *canonieke* vorm van het hypervlak. In dit geval is de margin, de loodrechte afstand van de punten tot het hypervlak, gelijk aan $1/\,||\boldsymbol{w}||$. Om dit in te zien volstaat het om twee tegengestelde punten te beschouwen waarvoor $|\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b| = 1$ geldt:

$$\begin{aligned} \rho_{(\boldsymbol{w},b)} &= \frac{\rho_{(\boldsymbol{w},b)}(\boldsymbol{x}_1, y_1) + \rho_{(\boldsymbol{w},b)}(\boldsymbol{x}_2, y_2)}{2} \\ &= \frac{1}{||\boldsymbol{w}||}, \end{aligned}$$

waarbij de tussenstappen worden geïllustreerd in Figuur 1.3.



Figuur 1.3: [56] Maximal margin classifier.

Dit maakt meteen het verband tussen de richtingsvector $\boldsymbol{w}$ en de margin duidelijk. Om de *maximal margin classifier* te vinden, moeten we zoeken naar het canonieke hypervlak dat de data scheidt en waarbij de norm minimaal is. Dit kunnen we samenvatten in volgend optimalisatieprobleem

$$\underset{\boldsymbol{w}\in\mathbb{R}^n, b\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^m}{\text{minimaliseer}} \quad F(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\,||\boldsymbol{w}||^2 \tag{1.5}$$

$$\text{met } \; y_i\big(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b\big) \geq 1 \quad i = 1, \ldots, m. \tag{1.6}$$

Vooraleer we de kernel trick kunnen toepassen, moeten we dit algoritme herschrijven in zijn duale voorstelling, waarbij we op zoek gaan naar een uitdrukking van de gewichtsvector $\boldsymbol{w}$ waar de input vectoren enkel binnen het scalair product voorkomen. In het geval van support vector machines (en vele andere optimalisatieproblemen), is het bestaan van zulke duale representatie gegarandeerd door het Representer theorem. In de praktijk moet men hiervoor een vertaling van primaire naar duale vorm maken. Om een idee te krijgen van hoe dit in zijn werk gaat, kan men de gelijklopende uitwerking in Sectie 2.1.1 bestuderen, waarna de lezer aangemoedigd wordt om op eigen houtje een analoge redenering voor het SVM probleem uit te werken. Ter controle kan men nadien bijvoorbeeld [56, 71] raadplegen.

Wanneer we de kernel trick loslaten op het SVM algoritme, wordt de beslissingsfunctie van de large margin classifier in het algemeen gegeven door volgende uitdrukking:

$$f(\boldsymbol{x}) = \operatorname{sgn}\Big(\sum_{i=1}^{m} \alpha_i^* y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b\Big) \tag{1.7}$$

waarbij de coëfficiënten α_i^* verkregen worden als oplossing van volgend duaal optimalisatieprobleem

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximaliseer}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{1.8}$$

$$\text{met } \sum_{i=1}^{m} \alpha_i y_i = 0 \text{ en } \alpha_i \geq 0 \qquad i = 1, \ldots, m. \tag{1.9}$$

Deze formulering van het SVM optimalisatieprobleem, noemt men het *hard margin* algoritme omdat hierbij geen trainingsfouten worden getolereerd.

Om de generalisatie eigenschappen te verbeteren of in het geval men te maken heeft met niet-separabele data, kan men deze techniek veralgemenen door slack variabelen toe te voegen en zo een aantal fouten toestaan. Om de controle te behouden, gaat men het gebruik van slack variabelen moeten bestraffen. Zoals beschreven in [18], zal men in de uiteindelijk te optimaliseren functionaal hiervoor een penalty term toevoegen. Met elke keuze van de term komt een bepaald type SVM overeen. In de literatuur wordt onderscheid gemaakt tussen de zogenaamde L1-SVM en L2-SVM, waarbij deze laatste ook wel beter bekend staat als LS-SVM (*least squares SVM*). Waarom deze opsplitsing nodig is, zal duidelijk worden in Hoofdstuk 3.

De versie die het meest gehanteerd wordt, is diegene waarbij de overtredingen van de nevenvoorwaarden lineair bestraft worden. Het primaire pro-

bleem ziet er dan als volgt uit

$$\underset{\boldsymbol{w}\in\mathbb{R}^n, b\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^m}{\text{minimaliseer}} \quad F(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\,||\boldsymbol{w}||^2 + C\sum_{i=1}^{m}\xi_i \tag{1.10}$$

$$\text{met } \; y_i\big(K(\boldsymbol{w}, \boldsymbol{x}_i) + b\big) \geq 1 - \xi_i \text{ en } \xi_i \geq 0 \quad i = 1, \ldots, m. \tag{1.11}$$

Het duale probleem verschilt er enkel in dat de α_i uit de nevenvoorwaarde (1.9) nu ook langs boven begrensd worden door de parameter C.

Het veelal vergeten broertje – L2-SVM – waarbij de fouten kwadratisch in rekening worden gebracht, komt veelal voor onder volgende gedaante

$$\underset{\boldsymbol{w}\in\mathbb{R}^n, b\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^m}{\text{minimaliseer}} \quad F(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2}\,||\boldsymbol{w}||^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2 \tag{1.12}$$

$$\text{met } \; y_i\big(K(\boldsymbol{w}, \boldsymbol{x}_i) + b\big) \geq 1 - \xi_i \text{ en } \xi_i \geq 0 \quad i = 1, \ldots, m, \tag{1.13}$$

en bezit de handige eigenschap dat we dit met behulp van een standaard truc equivalent kunnen praten met *hard margin* SVM [19, 57] en dat dus alle theoretische resultaten die ervoor gelden eveneens van toepassing zijn op deze *soft margin* machine.

Immers, stellen we via het standaard procédé het duale probleem op, gebruikmakend van het feit dat de voorwaarde dat de ξ_i positief horen te zijn eigenlijk overbodig[3] is, verkrijgen we volgende uitdrukking

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximaliseer}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \frac{1}{2C}\sum_{i=1}^{m}\alpha_i^2 \tag{1.14}$$

$$\text{met } \sum_{i=1}^{m}\alpha_i y_i = 0 \text{ en } \alpha_i \geq 0 \qquad i = 1, \ldots, m, \tag{1.15}$$

waarbij de laatste term van in (1.14), die we de Wolfe dual noemen, afkomstig is van de voorwaarde $2C\xi_i - \alpha_i = 0$, die we bekomen door de Lagrangian, i.e. de uitdrukking die zowel de objectieve van het primaire probleem als de nevenwoorden samenbalt, afleiden naar ξ_i.

Dit probleem is equivalent met de hard margin formulatie, waarbij de kernel functie vervangen wordt door $K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{1}{C}\delta_{ij}$, of in matrix notatie $K + \frac{1}{C}I_m$. Om dit na te gaan, stop je gewoon deze nieuwe kernel in de omschrijving van de hard-margin SVM en wanneer men bovendien opmerkt dat volgende

[3]Dit laatste is eenvoudig in te zien. Stel namelijk dat deze negatief zijn, dan kunnen we de waarde van de objectieve functie nog naar beneden halen door de negatieve ξ_i gelijk te stellen aan 0. Hieruit volgt onmiddellijk dat negatieve waarden dus niet kunnen voorkomen in de optimale oplossing.

gelijkheid geldt

$$\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j \frac{\delta_{ij}}{C} = \frac{1}{2C}\sum_{i=1}^{m}\alpha_i^2$$

komt formule (1.14) eruit.

We zien dat in de feature space, bepaald door deze nieuwe kernel, de data wel exact separabel is. Het idee om bij de kernel een kleine positieve constante op te tellen is wel vaker toegepast in het verleden. In de literatuur [47, 51] worden vaak termen zoals *shrinkage method* of *ridge regression* gebruikt voor het beschrijven van dit trucje.

Hoofdstuk 2

Support Vector Data Description

In het eindwerk van Pieter Wellens [71] werd een introductie gegeven tot classificatie van gegevens met behulp van Support Vector Machines (SVM) en tevens een tipje van Support Vector Regression (SVR) belicht, een nauw verwante techniek geschikt voor regressietaken. In navolging van David Tax [60, 62] gaan we dieper in op het minder bekende, maar daarom niet te verwaarlozen broertje: *single-class classificatie*. Hierbij is het niet de bedoeling om de data te ordenen in een welbepaalde klasse (dan wel een willekeurige reële waarde te bekomen), maar gaat men trachten de gegeven data zo goed mogelijk te beschrijven. Dit verklaart meteen de naam van de methode: *data description*. De beschrijving moet in staat zijn om de objecten van de *target data* (i.e. klasse van data voorgesteld door de training set) te onderscheiden van *alle* andere mogelijke objecten in de object ruimte en speelt dus een belangrijke rol bij outlier detectie of novelty detectie, het detecteren van objecten die significant verschillen van de gegeven data set.

Daarnaast kan een data description methode ook gebruikt worden in classificatie problemen waar slechts één van de klassen zeer goed is vertegenwoordigd. Een mogelijke toepassing is een monitoring systeem [31, 66] dat de huidige staat van een machine in de gaten houdt en waarbij we wensen dat een alarm signaal ons waarschuwt wanneer de machine problemen vertoont. Metingen van de normale toestand van de machine zijn eenvoudig en goedkoop te verkrijgen. Om informatie over de outliers daarentegen te verkrijgen is een vernieling van de machine op *alle* mogelijke manieren vereist. Het spreekt voor zich dat dit een zeer kostelijke zaak is en dat het vaak onmogelijk is om alle abnormale situaties te simuleren. Een methode die zich enkel focust op de target data en die geen representatieve outliers nodig hebben, kan dit monitor probleem oplossen.

Date description is geen nieuw begrip en wordt meestal met behulp van sta-

tistische methoden behandeld, al dan niet door eerst een schatting te maken van de onderliggende dichtheid (*density*). Support Vector Data Description (SVDD) is daarentegen een niet-parametrische methode die ontstaan is in navolging van Vapnik's principe om nooit problemen op te lossen die algemener zijn dan datgene waarin men eigenlijk geïnteresseerd is. In dit geval zoeken we enkel een gesloten rand rondom de representatieve data en zou het vooraf maken van een schatting van de dichtheid misschien van het goede te veel zijn.

Een extra argument om bij one-class classificatie enkel de grens van de target class te modelleren in plaats van de volledige dichtheidsverdeling, wordt gegeven in [40, 41]. Vaak is het op voorhand niet duidelijk wat de specifieke verdeling in de praktijk zal zijn. Ondanks we de machine laten runnen in verschillende, legale modi operandi, die het volledige operationele gebied van het toestel definiëren, leert dit ons niets over de specifieke distributie. In de praktijk kan het toestel zich veel langer in een bepaalde toestand bevinden dan andere en deze modus zou dus undersampled kunnen zijn in de training fase. Een data omschrijving voor dit type data zal dus beter af zijn met een model dat de grens van de normale class beschrijft, in plaats van een beschrijving van de dichtheid.

We zullen eerst de standaard methode, waarbij we enkel beschikken over positieve trainingsvoorbeelden, bespreken, gevolgd door een beschrijving waarbij we negatieve samples in de data set toelaten (en waardoor de kwaliteit van de omschrijving beter wordt). Daarenboven werken we tevens de equivalentie uit met een andere techniek beschreven in [55] en lichten we het theoretische verband met binaire classificatie (dus de standaard Support Vector Machines) toe.

2.1 Standaard SVDD

We spreken volgende notatie af om de trainingsdata te omschrijven

$$\boldsymbol{x}_i, \quad i = 1, \ldots, m, \quad \boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^n,$$

waarbij in vergelijking tot Support Vector Machines de klasse labels $y_i \in \{\pm 1\}$ ontbreken.

We hebben immers te maken met een vorm van *unsupervised learning*, waarbij de data een onderliggende kansverdeling P heeft, en we opzoek gaan naar een "eenvoudige" omschrijving van een deelruimte S van de input ruimte, zodat de kans dat een testpunt, gegenereerd uit deze verdeling P, buiten dit gebied valt gelijk is aan een vooraf opgegeven waarde gelegen tussen 0 en 1. Om dit probleem op te lossen zoeken we naar een beslissingsfunctie f die een positief is in S en negatief op het complement.

Het idee achter SVDD is een gesloten rand te construeren die de gegeven trainingsdata (ook wel target data genoemd) omsluit. Om dit te bereiken gaan we de meest geschikte $n-$dimensionale sfeer zoeken die de data zo goed mogelijk beschrijft. Dit is echter een vrij summiere omschrijving die om concrete uitwerking vraagt.

2.1.1 Uitwerking

Een sfeer wordt gekarakteriseerd door een centrum $\boldsymbol{a}$ en een strikt positieve straal R. De voorwaarde dat een element van de training set $\boldsymbol{x}$ binnen de sfeer ligt, vertaalt zich tot

$$||\boldsymbol{x} - \boldsymbol{a}||^2 \leq R^2.$$

Aangezien we de data omschrijving later gaan gebruiken om mogelijke nieuwigheden op te sporen, zoeken we naar een sfeer die zo nauw mogelijk aansluit aan de gegeven trainingsdata. Kortom, we zoeken een sfeer met zo klein mogelijk volume. Om dit te bereiken minimaliseren we de straal R.

Dit alles kunnen we samenballen tot de *rigid sphere description*:

$$\underset{R\in\mathbb{R},\boldsymbol{a}\in\mathbb{R}^n}{\text{minimaliseer}} \quad F(R,\boldsymbol{a}) = R^2 \tag{2.1}$$

$$\text{met } ||\boldsymbol{x}_i - \boldsymbol{a}||^2 \leq R^2 \qquad i = 1,\ldots,m. \tag{2.2}$$

Aangezien deze omschrijving zeer gevoelig kan worden voor de meest verafgelegen objecten in de target set (de *outliers*), laten we – in navolging van de *soft margin classifier* – enkele data punten buiten de sfeer toe. Dit wordt gerealiseerd door introductie van positieve *slack variabelen* ξ_i, waardoor de voorwaarde (2.2) aangepast wordt tot

$$||\boldsymbol{x}_i - \boldsymbol{a}||^2 \leq R^2 + \xi_i \qquad i = 1,\ldots,m \tag{2.3}$$

$$\xi_i \geq 0 \qquad i = 1,\ldots,m. \tag{2.4}$$

Ook is de objectieve functie (2.1) aan een kleine verandering toe. Het aanspreken van de slack variabelen kan immers niet ongestraft gebeuren en dient in rekening gebracht te worden door toevoeging van bijvoorbeeld[1] een L1 error term

$$R^2 + C\sum_{i=1}^{m}\xi_i. \tag{2.5}$$

[1]In het algemeen kan men de bestraffing formuleren in een willekeurige norm. Een andere veel voorkomende is de 2-norm waarbij de fouten kwadratisch worden bestraft, i.e. $\sum_{i=1}^{m}\xi_i^2$.

Zo niet zouden we telkens een triviale omschrijving vinden: een sfeer met straal R gelijk aan 0 en waarbij dus elk trainingspunt als outlier zal worden beschouwd. De betekenis van de parameter C, die een gulden middenweg tracht te vinden tussen het volume van de sfeer enerzijds en het aantal objecten die buiten de sfeer geclassificeerd worden anderzijds, zal naarmate het verhaal vordert duidelijk worden.

Samenvattend krijgen we volgende omschrijving van de *soft sphere description*:

$$\underset{R\in\mathbb{R},\boldsymbol{a}\in\mathbb{R}^n,\boldsymbol{\xi}\in\mathbb{R}^m}{\text{minimaliseer}} \quad F(R,\boldsymbol{a},\boldsymbol{\xi}) = R^2 + C\sum_{i=1}^{m}\xi_i \tag{2.6}$$

$$\begin{aligned} \text{met } ||\boldsymbol{x}_i - \boldsymbol{a}||^2 &\leq R^2 + \xi_i \qquad & i = 1,\ldots,m \\ \xi_i &\geq 0 & i = 1,\ldots,m. \end{aligned} \tag{2.7}$$

Alvorens we de Lagrangian construeren door de voorwaarden (2.7) op te nemen in de objectieve functie (2.6), merken we op dat we conditie (2.3) als volgt kunnen herschrijven in termen van scalaire producten

$$\begin{aligned} R^2 + \xi_i &\geq ||\boldsymbol{x}_i - \boldsymbol{a}||^2 \\ &= \langle \boldsymbol{x}_i - \boldsymbol{a}, \boldsymbol{x}_i - \boldsymbol{a}\rangle \\ &= \langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle - 2\langle \boldsymbol{x}_i, \boldsymbol{a}\rangle + \langle \boldsymbol{a}, \boldsymbol{a}\rangle. \end{aligned}$$

We verkrijgen volgende Lagrangian van het primaire probleem,

$$\begin{aligned} L_p(R,\boldsymbol{a},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\gamma}) = R^2 &+ C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\gamma_i\xi_i \\ &- \sum_{i=1}^{m}\alpha_i\Big(R^2 + \xi_i - (\langle \boldsymbol{x}_i, \boldsymbol{x}_i\rangle - 2\langle \boldsymbol{x}_i, \boldsymbol{a}\rangle + \langle \boldsymbol{a}, \boldsymbol{a}\rangle)\Big), \end{aligned} \tag{2.8}$$

met positieve Lagrange multipliers $\alpha_i \geq 0$ en $\gamma_i \geq 0$.

We toveren dit primaire probleem om tot z'n duale broertje door L_p af te leiden naar respectievelijk R, $\boldsymbol{a}$ en $\boldsymbol{\xi}$ om zo de zadelpuntsvoorwaarde te

definiëren,

$$\frac{\partial L_p(R, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\gamma})}{\partial R} = 0 \Leftrightarrow 2R - 2R\sum_{i=1}^{m} \alpha_i = 0$$

$$\Leftrightarrow \sum_{i=1}^{m} \alpha_i = 1 \tag{2.9}$$

$$\frac{\partial L_p(R, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\gamma})}{\partial \boldsymbol{a}} = 0 \Leftrightarrow -\sum_{i=1}^{m} \alpha_i(-2\boldsymbol{x}_i + 2\boldsymbol{a}) = 0$$

$$\Leftrightarrow \boldsymbol{a} = \sum_{i=1}^{m} \alpha_i \boldsymbol{x}_i \tag{2.10}$$

$$\frac{\partial L_p(R, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\gamma})}{\partial \xi_i} = 0 \Leftrightarrow C - \alpha_i - \gamma_i = 0 \quad (i = 1, \ldots, m)$$

$$\Leftrightarrow \gamma_i = C - \alpha_i, \tag{2.11}$$

en vervolgens deze terug te substitueren in L_p, zodat we een vergelijking enkel in functie van de Lagrange multipliers $\boldsymbol{\alpha}$ over houden. Dit zal dan de *Wolfe's dual* vormen. Voor meer theoretische beschouwing, verwijs ik naar Sectie 6.3 uit [56].

Wanneer we nu de *positivity constraints* van de Lagrange multipliers combineren met (2.11), dan kunnen we dit samenvatten tot volgende beperking op het bereik van α_i:

$$0 \leq \alpha_i \leq C \quad i = 1, \ldots, m. \tag{2.12}$$

Wanneer we nu (2.9) en (2.10) substitueren in (2.8), zien we dat we een aantal termen paarsgewijs kunnen schrappen en blijft er ten slotte volgende uitdrukking over

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x}_i \rangle - \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle. \tag{2.13}$$

Formuleren we ter verwijzing nog eventjes het duale optimalisatieprobleem:

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximaliseer}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x}_i \rangle - \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \tag{2.14}$$

$$\text{met} \sum_{i=1}^{m} \alpha_i = 1 \qquad i = 1, \ldots, m \tag{2.15}$$

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, m. \tag{2.16}$$

Hierin herkennen we een convex kwadratisch probleem, want de Gram matrix, gevormd door het nemen van onderlinge scalaire producten, is positief

semi-definiet. Dit probleem kunnen we dus oplossen met een convex Quadratic Problem (QP) solver naar keuze.

Rest er ons enkel nog te beschrijven hoe de uiteindelijke testfunctie, die aangeeft of een nieuw object $\boldsymbol{z}$ behoort tot de target set of eerder beschouwd kan worden als outlier, er uitziet. Hiervoor dienen we enkel te controleren of het zich binnen of buiten de gesloten sfeer bevindt. Dit kunnen we gemakkelijk doen aan de hand van volgende formule

$$\begin{aligned} ||\boldsymbol{z}-\boldsymbol{a}||^2 &= \langle \boldsymbol{z},\boldsymbol{z}\rangle - 2\langle \boldsymbol{z},\boldsymbol{a}\rangle + \langle \boldsymbol{a},\boldsymbol{a}\rangle \\ &= \langle \boldsymbol{z},\boldsymbol{z}\rangle - 2\sum_{i=1}^{m}\alpha_i\langle \boldsymbol{z},\boldsymbol{x}_i\rangle + \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\langle \boldsymbol{x}_i,\boldsymbol{x}_j\rangle \\ &\leq R^2, \end{aligned} \tag{2.17}$$

afgeleid uit (2.10) waar de center $\boldsymbol{a}$ geschreven is als lineaire combinatie van de $\boldsymbol{x}_i$'s en de coëfficiënten van de optimale multiplier $\boldsymbol{\alpha}$.

Dit alles resulteert in volgende beslissingsfunctie:

$$f(\boldsymbol{z}) = \operatorname{sgn}\Big(R^2 - \sum_{i,j}\alpha_i\alpha_j\langle \boldsymbol{x}_i,\boldsymbol{x}_j\rangle + 2\sum_{i}\alpha_i\langle \boldsymbol{z},\boldsymbol{x}_i\rangle - \langle \boldsymbol{z},\boldsymbol{z}\rangle\Big). \tag{2.18}$$

Alvorens we de uitbreiding voor het niet-lineaire geval uitwerken, is het nuttig om enkele eenvoudige eigenschappen en verbanden uit de doeken te doen.

Eerst en vooral een beetje terminologie. Net als in het geval bij SVMs voor classificatie doeleinden, zal uiteindelijk niet alle trainingsdata gebruikt worden bij het leveren van de data omschrijving. Deze eigenschap is wat men in de volksmond *sparseness* van de oplossing noemt. Dankzij de KKT (Karush-Kuhn-Tucker) condities weten we dat ofwel de nevenvoorwaarden actief zijn ofwel dat de Lagrange multipliers 0 zijn. Hierbij kunnen we opmerken dat enkel die vectoren waarvoor de overeenkomstige $\alpha_i > 0$ is, de zogenaamde *support vectors*, een bijdrage leveren bij de beschrijving van de $n-$dimensionale sfeer. De $\boldsymbol{x}_i$'s waarvoor de α_i gelijk is aan 0, liggen binnen de sfeer.

De aandachtige lezer vraagt zich nu af, hoe de fout geclassificeerde target data, de zogenaamde *false negatives*, te herkennen?

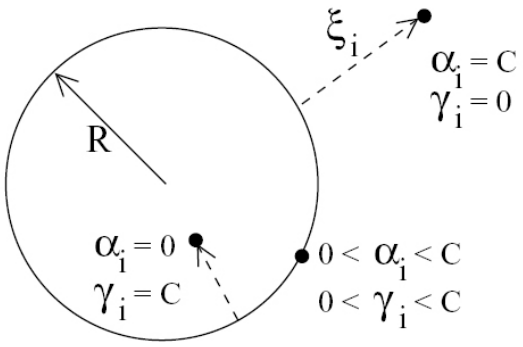
Tot hiertoe hebben we enkel nog maar rekening gehouden met de multiplier $\boldsymbol{\alpha}$, die overeenstemt met voorwaarde dat het merendeel van de data binnen de sfeer moet liggen. We hebben echter ook nog de multiplier $\boldsymbol{\gamma}$ die door middel van (2.11) sterk aan de α_i's gekoppeld wordt. Punten die buiten de sfeer liggen worden gekarakteriseerd door een positieve ξ_i en liggen dus op de sfeer met straal $R^2 + \xi_i$. Weer volgt uit de KKT condities, dat hierdoor

de overeenstemmende γ_i dient te vervagen tot 0. Wat op zijn beurt betekent dat α_i de bovengrens C bereikt.

Het centrum van de sfeer $\boldsymbol{a}$ kunnen we uitrekenen met behulp van (2.10). Om de parameter R uit te rekenen volstaat het de afstand te berekenen van $\boldsymbol{a}$ tot een support vector (ook wel afgekort tot SV) die op de rand van de sfeer ligt

$$R = ||\boldsymbol{x}_i - \boldsymbol{a}|| \text{ met } 0 < \alpha_i < C. \tag{2.19}$$

Een overzicht wordt verschaft in bijhorende Figuur 2.1.



Figuur 2.1: [60] De waarde van de twee Lagrange multipliers α_i en ξ_i voor objecten $\boldsymbol{x}_i$ in, op de rand van en buiten de hypersfeer.

2.1.2 Inpluggen van kernels

Wanneer we het duale optimalisatieprobleem (2.14) en (2.16) bekijken, zien we enkel een uitdrukking met scalaire producten. Net zoals bij niet-lineaire SVMs en andere kernel technieken, kunnen we ook hier de *kernel-trick* toepassen, waarbij we een impliciete mapping maken naar een hogere – mogelijk oneindige – dimensionale Hilbertruimte, die we de *feature space* noemen. De *curse of dimensionality* wordt echter omzeild wegens Mercer's theorem die ons door middel van een kernel functie toelaat impliciet te werken in deze ruimte zonder eerst onze data op te blazen. Met een klein beetje extra rekenwerk, namelijk het uitrekenen van de Gram matrix voor de evaluatie van de kernel functie van twee-aan-twee genomen input vectoren, laat dit flexibelere omschrijvingen toe.

Het idee achter deze methode is dat de meest ideale kernel een mapping vormt van de input data (de target set) naar een gesloten sferische ruimte en alle outliers er buiten afbeeldt. Bij toepassing op de SVDD methode vinden we dan deze beschrijving van de data door eenvoudigweg elk voorkomen van

een scalair product te vervangen door het overeenstemmende element uit de Gram matrix.

Beschouw een willekeurige kernel functie $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Door de scalaire producten in (2.14) en (2.18) te vervangen leidt, dit tot volgend veralgemeend optimalisatieprobleem

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximaliseer}} \quad W(\boldsymbol{\alpha}) = \sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.20}$$

$$\text{met} \sum_{i=1}^{m} \alpha_1 = 1 \qquad i = 1, \ldots, m \tag{2.21}$$

$$0 \leq \alpha_i \leq C \qquad i = 1, \ldots, m \tag{2.22}$$

en aangepaste beslissingsfunctie

$$f(\boldsymbol{z}) = \text{sgn}\Big(R^2 - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + 2\sum_i \alpha_i K(\boldsymbol{z}, \boldsymbol{x}_i) - K(\boldsymbol{z}, \boldsymbol{z})\Big). \tag{2.23}$$

Merk op dat dankzij het feit dat we werken met positief definiete kernels, we nog steeds te maken hebben met een convex kwadratisch probleem en de routine om dit op te lossen gewoon de kernel functie moeten berekenen in plaats van het canonieke inproduct.

In dit model is de keuze van de kernel functie en tevens ook de waarde van de hyperparameters, dit zijn de parameters die de gebruiker dient in te stellen in de kernel zelf, van primordiaal belang. Het blijkt echter dat, in tegenstelling tot bij classificatie met behulp van Support Vector Machines, niet elke kernel geschikt is voor one-class classificatie. We zullen in volgend deeltje aangeven waarom het gebruik van de polynomiale kernel te vermijden valt bij niet gestandaardiseerde data, kortom wanneer centrering rond de oorsprong en herschaling van de variantie niet eerder plaatsvindt.

Wat de Gaussische kernel betreft, laten we zien dat bij toename van de hyperparameter σ de oplossing varieert van een beschrijving van de data met alle trainingsobjecten als support vectors (i.e. het probleem wordt overfit/gememoriseerd) tot één met een minimum aan support vectors, wat de sferische benadering oplevert.

Polynomiale kernel

De *homogene polynomiale kernel* wordt gegeven door

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle^d,$$

waarbij de vrije parameter d de graad van de individuele termen aangeeft.

De cosinus van de hoek θ_{ij} tussen $\boldsymbol{x}_i$ en $\boldsymbol{x}_j$ wordt aan de hand van het scalair product als volgt gedefinieerd

$$\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle = \cos(\theta_{ij}) \, ||\boldsymbol{x}_i|| \, ||\boldsymbol{x}_j|| \, .$$

Wanneer de hoek θ_{ij} vrij klein is (en dus de $\cos(\theta_{ij}) \sim 1$), zal de waarde van het scalair product weinig veranderen en, bij toenemende graad d, zullen de objecten met de grootste norm (en dus het verst van de oorsprong) de bovenhand nemen:

$$\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle^d = \cos^d(\theta_{ij}) \, ||\boldsymbol{x}_i||^d \, ||\boldsymbol{x}_j||^d \simeq ||\boldsymbol{x}_i||^d \, ||\boldsymbol{x}_j||^d \, .$$

Dit effect treedt op wanneer de data niet gecentreerd is rond de oorsprong en kan dus mogelijk teruggedrongen worden door de data eerst rond te oorsprong te centreren en de data te herschalen zodat eenheidsvariantie bekomen wordt:

$$\begin{aligned} \overline{x}_j &= \frac{1}{m} \sum_{i=1}^{m} x_{ij} && j = 1, \ldots, n \\ x'_{ij} &= \frac{x_{ij} - \overline{x}_j}{\sqrt{\frac{1}{m} \sum_{i=1}^{m} (x_{ij} - \overline{x}_j)^2}} && i = 1, \ldots, m \quad j = 1, \ldots, n. \end{aligned}$$

Het is echter algemeen geweten dat het standaardiseren van data bij unsupervised learning technieken wordt afgeraden omdat dan de ruis vergroot wordt in richtingen die oorspronkelijk een kleine variantie hadden. Om dit tegen te gaan wordt aangeraden de data te preprocessen door eerst een Principal Component Analyse te doen om de dimensie te reduceren en enkel de meest variërende componenten over te houden en pas daarna deze de overgebleven principale componenten (PCs) te standaardiseren.

Tenslotte wordt in [65] aangehaald dat ook het centreren van data in feature space niet helpt, want

Stelling 2.1. *Centered SVDD is equivalent met de oorspronkelijke SVDD.*

Bewijs. Het centreren van data in feature space komt overeen met

$$\boldsymbol{x}'_i = \Phi(\boldsymbol{x_i}) - \frac{1}{m} \sum_{i=1}^{m} \Phi(\boldsymbol{x}_i) = \Phi(\boldsymbol{x_i}) - \overline{\Phi(x)},$$

waarbij Φ de impliciete mapping is van input space naar feature space en overeenkomt met de kernel functie $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle$.

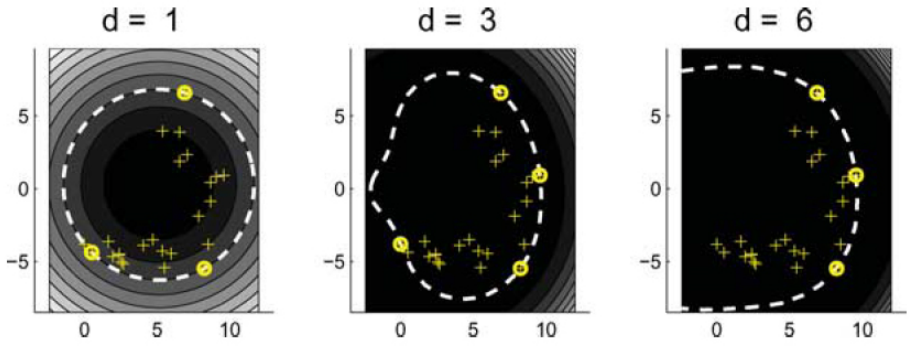
Als we dit invullen in (2.14) verkrijgen we

$$\begin{aligned}
W(\boldsymbol{\alpha}) &= \sum_i \alpha_i \langle \boldsymbol{x}'_i, \boldsymbol{x}'_i \rangle - \sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}'_i, \boldsymbol{x}'_j \rangle \\
&= \sum_i \alpha_i \langle \Phi(\boldsymbol{x}_i) - \overline{\Phi(x)}, \Phi(\boldsymbol{x}_i) - \overline{\Phi(x)} \rangle \\
&\quad - \sum_{i,j} \alpha_i \alpha_j \langle \Phi(\boldsymbol{x}_i) - \overline{\Phi(x)}, \Phi(\boldsymbol{x}_j) - \overline{\Phi(x)} \rangle \\
&= \sum_i \alpha_i \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_i) \rangle - 2\overline{\Phi(x)} \sum_i \alpha_i \Phi(\boldsymbol{x}_i) + \overline{\Phi(x)}^2 \\
&\quad - [\sum_{i,j} \alpha_i \alpha_j \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle - 2\overline{\Phi(x)} \sum_i \alpha_i \Phi(\boldsymbol{x}_i) + \overline{\Phi(x)}^2] \\
&= \sum_i \alpha_i \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_i) \rangle - \sum_{i,j} \alpha_i \alpha_j \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle \\
&= \sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j),
\end{aligned}$$

wat is gelijk aan (2.20).

Op analoge wijze kan men aantonen dat ook de testfunctie (2.23) ongewijzigd blijft. □

Op Figuur 2.2 wordt zichtbaar dat naarmate de graad d toeneemt de meest rechtse objecten, i.e. objecten verder verwijderd van de oorsprong, support vectors worden en dat de data beschrijving enkel onderscheidt maakt op basis van de norm. Grote gebieden in de input space zonder target objecten worden aanvaard door deze omschrijving.



Figuur 2.2: [65] Polynomiale kernel met variërende graad. De support vectors zijn aangegeven met bolletjes, en de streepjeslijn geeft de rand van de beschrijving aan.

Gaussische RBF kernel

Een andere vaak gehanteerde kernel is de *Gaussische radial basis function (RBF) kernel*

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2})$$

met parameter $\sigma^2 > 0$, die de breedte (i.e. de spreiding) bepaalt.

Aangezien dit een RBF kernel is en dus enkel de onderlinge afstand tussen objecten gebruikt, is deze kernel onafhankelijk van de ligging van de data ten opzichte de oorsprong. Bovendien geldt onmiddellijk dat $K(\boldsymbol{x}_i, \boldsymbol{x}_i) = 1$, onafhankelijk van de keuze van hyperparameter σ^2, en zal normeren van de input vectoren geen invloed hebben. Bovendien kunnen we hierdoor de objectieve functie van het duale probleem (2.20) en beslissingsfunctie (2.23) vereenvoudigen tot

$$W(\boldsymbol{\alpha}) = 1 - \sum_i \alpha_i^2 - \sum_{i \neq j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.24}$$

en

$$f(\boldsymbol{z}) = \operatorname{sgn}\Big(R^2 - \sum_i \alpha_i^2 - \sum_{i \neq j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + 2\sum_i \alpha_i K(\boldsymbol{z}, \boldsymbol{x}_i) - 1\Big). \tag{2.25}$$

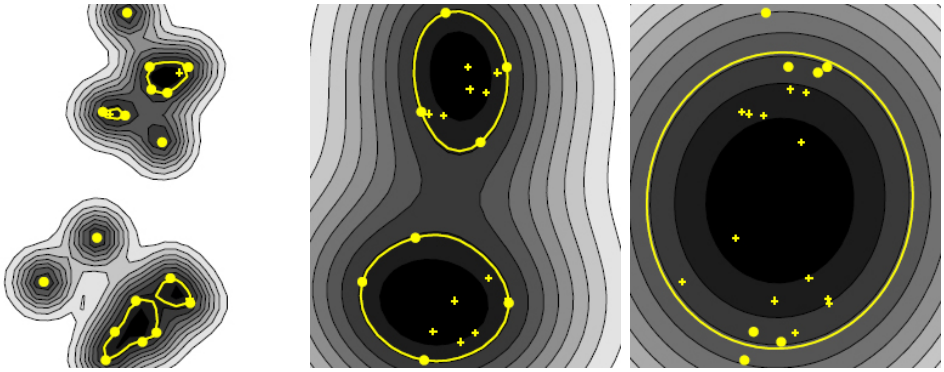
Om het maximum te vinden van (2.24) dat voldoet aan nevenvoorwaarden (2.22), kan men opmerken dat er twee tegenstrijdigheden opduiken:

1. hoe groter het aantal α_i's die verschillend zijn van 0, des te kleiner elke α_i en de term $\sum_i \alpha_i^2$ kunnen worden. Dit betekent echter een stijging in het aantal support vectors, wat de *sparseness* van de oplossing niet ten goede komt.

2. hoe groter het aantal α_i's gelijk aan 0, des te minder het product $\alpha_i \alpha_j$ een bijdrage levert in $\sum_{i \neq j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ en een daling betekent in het aantal support vectors.

De grootte van de α_i's en het product is afhankelijk van de *weighting factors* $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ en hangt dus af van de keuze van de parameter σ^2. We kunnen dus besluiten dat σ^2 het aantal support vectors bepaalt.

Wanneer we σ^2 zeer klein kiezen, dan is $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \simeq 0$. Dit betekent dat de term $\sum_{i \neq j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ wegvalt in (2.24) en we voor $\sum_i \alpha_i^2$ een optimale waarde vinden wanneer we $\alpha_i = 1/m$ stellen. $W(\alpha)$ is dan gelijk aan $1 - 1/m$. Hieruit kunnen we rechtstreeks afleiden dat het aantal support vectors gelijk is aan m. Kortom, elk trainingsobject wordt een support vector.



Figuur 2.3: [61] Afstand tot het centrum van de hypersfeer, teruggetrokken naar de input ruimte voor 3 verschillende, toenemende waarden van de parameter σ in de Gaussische RBF kernel.

Wanneer we anderzijds σ^2 zeer groot laten worden, is $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \simeq 1$. Waaruit volgt dat $W(\boldsymbol{\alpha}) = 1 - \sum_i \alpha_i^2 - 2\sum_{i<j} \alpha_i \alpha_j$ en een niet-negatieve waarde, namelijk 0, aanneemt wanneer één van de α_i's gelijk is aan 1 en de 0 zijn. Dit benadert de *rigid sphere description*.

Om dit in te zien bekijken we de Taylor ontwikkeling[2] van de kernel functie

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = 1 - \frac{||\boldsymbol{x}_i||^2}{2\sigma^2} - \frac{||\boldsymbol{x}_j||^2}{2\sigma^2} + \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{\sigma^2} + \ldots. \tag{2.26}$$

Wanneer we nu (2.26) substitueren in (2.20), verkrijgen we:

$$\begin{aligned}
W(\boldsymbol{\alpha}) &= \sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
&= \sum_i \alpha_i (1 - \frac{||\boldsymbol{x}_i||^2}{2\sigma^2} - \frac{||\boldsymbol{x}_i||^2}{2\sigma^2} + \frac{||\boldsymbol{x}_i||^2}{\sigma^2} + \ldots) \\
&\quad - \sum_{i,j} \alpha_i \alpha_j (1 - \frac{||\boldsymbol{x}_i||^2}{2\sigma^2} - \frac{||\boldsymbol{x}_j||^2}{2\sigma^2} + \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{\sigma^2} + \ldots) \\
&= \sum_i \alpha_i + \ldots - \sum_i \alpha_i \sum_i \alpha_i + 2\sum_i \alpha_i \frac{||\boldsymbol{x}_i||^2}{2\sigma^2} - \sum_{i,j} \alpha_i \alpha_j \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{\sigma^2} + \ldots \\
&= 1 + \ldots - 1 + \sum_i \alpha_i \frac{||\boldsymbol{x}_i||^2}{\sigma^2} - \sum_{i,j} \alpha_i \alpha_j \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{\sigma^2} + \ldots
\end{aligned}$$

[2]De Taylorontwikkeling van een functie met n variabelen

$$f(x_1 + \Delta_1, \ldots, x_n + \Delta_n) = \sum_{i=0}^{\infty} \frac{1}{i!} (\sum_{j=1}^{n} \Delta_j \frac{\partial}{\partial x_j})^i f(x_1, \ldots, x_n)$$

Als we de hogere orde termen negeren, zien we dat het optimalisatieprobleem equivalent is aan dat van de *rigid sphere description* op vermenigvuldiging met constante factor $1/\sigma^2$ na. Wanneer we dus in (2.14) de modelparameter C gelijkstellen aan C/σ^2, volgt de equivalentie.

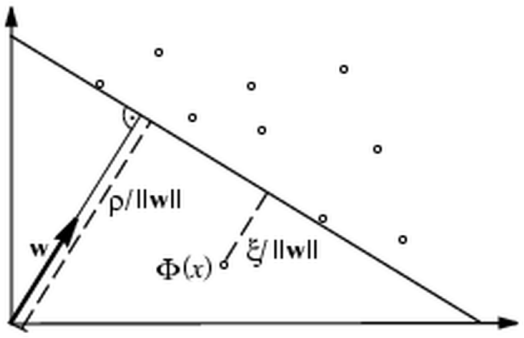
Kortom, als we σ^2 laten toe nemen zal ons model steeds dichter aanleunen bij het sferische model, en zal het aantal support vectors dalen.

2.2 ν single-class classifier

In deze sectie bekijken we het single-class probleem vanuit een andere hoek [55, 56], die nog nauwer aansluit bij het oorspronkelijk idee achter SVM voor binaire classificatie en waarvan we tevens de equivalentie met SVDD kunnen uitwerken.

Opnieuw zoeken we een hypothese functie f die de waarde +1 aanneemt in een "klein" gebied dat de meeste de datapunten bevat en -1 elders. Geheel in de trend van de support vector methodologie, werken we eerst het lineaire probleem uit, waarna we afstappen van het lineaire karakter van het algoritme door de data afbeelden in een feature ruimte, die overeenkomt met de gekozen kernel functie.

Daar waar we met SVDD op zoek gaan naar een sfeer rond de data met minimaal volume, scheiden we nu de data van de oorsprong door een hypervlak te construeren met maximale *margin*. Het idee om een hypervlak met de grootste mogelijke loodrechte afstand tot de oorsprong te beschouwen, wordt geïllustreerd in Figuur 2.4.



Figuur 2.4: [56] Het hypervlak $\langle \boldsymbol{w}, \Phi(\boldsymbol{x}) \rangle = \rho$, dat op één punt na de data scheidt van de oorsprong. De afstand van de outlier tot het hypervlak is gelijk aan $\xi / ||\boldsymbol{w}||$; de afstand tussen hypervlak en de oorsprong is $\rho / ||\boldsymbol{w}||$. Dit laatste impliceert dat een kleine $||\boldsymbol{w}||$ correspondeert met een grote margin tot de oorsprong.

Voor een testpunt $\boldsymbol{z}$, bepaalt de waarde $f(\boldsymbol{z})$ aan welke zijde van het hypervlak dit punt terecht komt in de feature ruimte. De vrijheid om verschillende types van kernel functies te gebruiken, zorgt ervoor dat dit eenvoudig meetkundig idee overeenkomt met een verscheidenheid aan niet-lineaire modellen in de input ruimte.

Om de data van de oorsprong te scheiden, lossen we volgend kwadratisch probleem op:

$$\underset{\boldsymbol{w}\in\mathcal{H},\boldsymbol{\xi}\in\mathbb{R}^m,\rho\in\mathbb{R}}{\text{minimaliseer}} \quad F(\boldsymbol{w},\rho,\boldsymbol{\xi}) = \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + \frac{1}{\nu m}\sum_i \xi_i - \rho \tag{2.27}$$

$$\text{met } \langle \boldsymbol{w}, \Phi(\boldsymbol{x}_i)\rangle \geq \rho - \xi_i \qquad i = 1,\ldots,m \tag{2.28}$$

$$\xi_i \geq 0 \qquad i = 1,\ldots,m.$$

Hier is $\nu \in (0,1]$ een parameter waarvan de betekenis snel duidelijk zal worden en $||\boldsymbol{w}||^2$ de *regulizer* die de lengte van de gewichtsvector in feature space controleert en het aantal mogelijke functies uit de hypothese ruimte beperkt.

Aangezien de slack variabelen ξ_i worden bestraft in de objectieve functie, kunnen we verwachten dat de gevonden $\boldsymbol{w}$, met een kleine regulatie term $||\boldsymbol{w}||$, en offset ρ het probleem zullen oplossen en dat de beslissingsfunctie

$$f(\boldsymbol{z}) = \text{sgn}\Big(\langle \boldsymbol{w}, \Phi(\boldsymbol{z}\rangle - \rho\Big) \tag{2.29}$$

voor de meeste voorbeelden $\boldsymbol{x}_i$ uit de training set de waarde 1 zal opleveren.

Gebruikmakende van de multipliers $\alpha_i, \gamma_i \geq 0$, kunnen we de Lagrangian

$$L(\boldsymbol{w},\boldsymbol{\xi},\rho,\boldsymbol{\alpha},\boldsymbol{\gamma}) = \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + \frac{1}{\nu m}\sum_i \xi_i - \rho - \sum_i \alpha_i(\langle \boldsymbol{w}, \Phi(\boldsymbol{x}_i)\rangle - \rho + \xi_i) - \sum_i \gamma_i\xi_i \tag{2.30}$$

introduceren en wederom de afgeleiden naar de primaire variabelen $\boldsymbol{w}, \boldsymbol{\xi}, \rho$ gelijkstellen aan 0, levert ons volgende verbanden

$$\boldsymbol{w} = \sum_i \alpha_i \Phi(\boldsymbol{x}_i) \tag{2.31}$$

$$\alpha_i = \frac{1}{\nu m} - \gamma_i \leq \frac{1}{\nu m} \tag{2.32}$$

$$\sum_i \alpha_i = 1 \tag{2.33}$$

waarbij (2.31) de zogenaamde *Support Vector expansion* is.

Met de impliciete uitdrukking van het scalair product in de feature ruimte, door gebruik te maken van een kernel functie, kunnen we de beslissingsfunctie (2.29) transformeren tot

$$f(\boldsymbol{z}) = \operatorname{sgn}\Big(\sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{z}) - \rho\Big) \tag{2.34}$$

en samen met de substitutie van voorwaarden (2.31) - (2.33) in (2.27), verkrijgen we uiteindelijk het kwadratische probleem voor ν single-class classificatie:

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximaliseer}} \quad W(\boldsymbol{\alpha}) = -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.35}$$

$$\text{met} \quad 0 \leq \alpha_i \leq \frac{1}{\nu m} \quad i = 1, \ldots, m \tag{2.36}$$

$$\sum_i \alpha_i = 1.$$

2.2.1 Verband met Support Vector Data Description

Wanneer we $1/(\nu m)$ gelijkstellen aan C, zien we dat de nevenvoorwaarden voor beide voorgestelde benaderingen samensmelten. Er is echter meer aan de hand. Beperken we ons tot translatie invariante kernels, zoals bijvoorbeeld de Gaussische RBF kernel, dan kunnen we het volgende aantonen.

Eigenschap 2.2. *Onderstel $K(.,.)$ translatie invariant, dan zijn SVDD en ν single-class classificatie equivalent.*

Bewijs. De translatie invariantie eigenschap van een kernel houdt in dat,

$$K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x} + \boldsymbol{x}_0, \boldsymbol{x}' + \boldsymbol{x}_0) \quad \forall \boldsymbol{x}_0 \in \mathcal{X}.$$

Hieruit kunnen we onmiddellijk afleiden dat, $K(\boldsymbol{x}, \boldsymbol{x})$ gelijk is aan een constante k onafhankelijk van de vector $\boldsymbol{x}$. Immers, stel $\boldsymbol{x}_0$ gelijk aan $-\boldsymbol{x}$ dan volgt dat

$$K(\boldsymbol{x}, \boldsymbol{x}) = K(\boldsymbol{0}, \boldsymbol{0}) = k.$$

We zijn nu in staat om de equivalentie van zowel het op te lossen optimalisatieprobleem als de beslissingsfuncties van beide benaderingen aan te

tonen.

$$\begin{aligned}
&\max_{\boldsymbol{\alpha}} \sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
\Leftrightarrow &\max_{\boldsymbol{\alpha}} \sum_i \alpha_i k - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
\Leftrightarrow &\max_{\boldsymbol{\alpha}} k - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
\Leftrightarrow &\max_{\boldsymbol{\alpha}} -\sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
\Leftrightarrow &\max_{\boldsymbol{\alpha}} -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)
\end{aligned}$$

Aangezien de constanten uit de beslissingsfunctie kunnen uitgerekend worden door een support vector $\boldsymbol{x}_i$ die voldoet aan de voorwaarde $0 < \alpha_i < C$ in te vullen, verkrijgen we enerzijds

$$R^2 = ||\Phi(\boldsymbol{x}_i) - \boldsymbol{a}||^2 = k - 2\sum_j \alpha_j K(\boldsymbol{x}_j, \boldsymbol{x}_i) + \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

en anderzijds

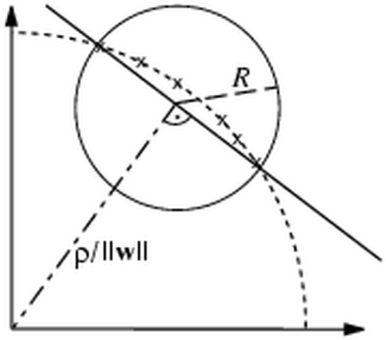
$$\rho = \langle \boldsymbol{w}, \Phi(\boldsymbol{x_i}) \rangle = \sum_j \alpha_j K(\boldsymbol{x}_j, \boldsymbol{x}_i).$$

Hieruit volgt dat ook dezelfde beslissing genomen wordt, want

$$\begin{aligned}
&f_{SVDD}(\boldsymbol{z}) \\
&= \operatorname{sgn}\Big(R^2 - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + 2\sum_i \alpha_i K(\boldsymbol{z}, \boldsymbol{x}_i) - K(\boldsymbol{z}, \boldsymbol{z})\Big) \\
&= \operatorname{sgn}\Big(k - 2\sum_j \alpha_j K(\boldsymbol{x}_j, \boldsymbol{x}_i) + \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
&\qquad - \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + 2\sum_i \alpha_i K(\boldsymbol{z}, \boldsymbol{x}_i) - k\Big) \\
&= \operatorname{sgn}\Big(2\sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{z}) - 2\sum_j \alpha_j K(\boldsymbol{x}_j, \boldsymbol{x}_i)\Big) \\
&= \operatorname{sgn}\Big(\sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{z}) - \sum_j \alpha_j K(\boldsymbol{x}_j, \boldsymbol{x}_i)\Big) \\
&= \operatorname{sgn}\Big(\sum_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{z}) - \rho\Big) \\
&= f_{\nu-SCC}(\boldsymbol{z}).
\end{aligned}$$

□

Ook meetkundig gezien is dit resultaat aanneembaar: voor constante $K(\boldsymbol{x}, \boldsymbol{x})$ liggen alle voorbeelden op een sfeer in de feature ruimte. Het vinden van de kleinste bol die alle punten bevat, komt overeen met het vinden van kleinste bolsegment waarop de data ligt. Dit segment kan echter eenvoudigweg gevonden worden door de sfeer te snijden met een hypervlak – het hypervlak met de grootste speling tot de oorsprong snijdt klaarblijkelijk het kleinste segment af (Figuur 2.5).



Figuur 2.5: [56] Voor RBF kernels, die enkel afhangen op $\boldsymbol{x} - \boldsymbol{x}'$, is $K(\boldsymbol{x}, \boldsymbol{x})$ constant, en liggen de getransformeerde data punten dus op een hypersfeer in feature space. In dit geval is het vinden van de kleinste sfeer rondom de data equivalent met het maximaliseren van de loodrechte afstand tot de oorsprong.

Opdat de equivalentie blijft gelden voor een willekeurige kernel, dienen we ervoor te zorgen dat de uitdrukking $K(\boldsymbol{x}, \boldsymbol{x})$ constant is. Dit kunnen we bijvoorbeeld realiseren door te werken in genormaliseerde feature ruimtes [27].

Definieer

$$\tilde{K}(\boldsymbol{x}, \boldsymbol{y}) = \frac{K(\boldsymbol{x}, \boldsymbol{y})}{\sqrt{K(\boldsymbol{x}, \boldsymbol{x}) K(\boldsymbol{y}, \boldsymbol{y})}}$$

dan volgt

$$\tilde{K}(\boldsymbol{x}, \boldsymbol{x}) = \frac{K(\boldsymbol{x}, \boldsymbol{x})}{\sqrt{K(\boldsymbol{x}, \boldsymbol{x})^2}} = \frac{K(\boldsymbol{x}, \boldsymbol{x})}{|K(\boldsymbol{x}, \boldsymbol{x})|} = 1,$$

waarbij we in de laatste stap gebruik hebben gemaakt van de eigenschap dat kernels positief zijn op de diagonaal[3]. Het is eenvoudig in te zien dat deze functie nog steeds een kernel is, de positief semi-definietheid blijft immers ongeschonden. Eigenlijk hebben we niets anders gedaan dan het normalise-

[3]Dit volgt triviaal uit de definitie van een kernel. (Hint: stel $m = 1$ in Definitie 1.1.)

ren van de vectoren in de feature ruimte,

$$\tilde{K}(\boldsymbol{x},\boldsymbol{y}) = \langle \tilde{\Phi}(\boldsymbol{x}), \tilde{\Phi}(\boldsymbol{y}) \rangle \text{ met } \tilde{\Phi}(\boldsymbol{x}) = \frac{\Phi(\boldsymbol{x})}{||\Phi(\boldsymbol{x})||}.$$

Men spreekt daarom van een *genormaliseerde mapping*.

Bij een Gaussische RBF kernel, liggen de *patterns* reeds op de eenheidssfeer in de geassocieerde Hilbertruimte en is normalisatie niet nodig. Voor monische polynomiale kernels, is het voldoende om, net zoals in het lineaire geval, de normalisatie uit te voeren in de input ruimte[4]. Hiervoor is een eenvoudige verklaring: een scalair product is een bilineaire vorm en dus constanten (in ons geval de norm van de vectoren) kunnen zowel in eerste als tweede veranderlijke voorop geschoven worden, waaruit onmiddellijk volgt dat

$$\tilde{K}(\boldsymbol{x},\boldsymbol{y}) = K(\frac{\boldsymbol{x}}{||\boldsymbol{x}||}, \frac{\boldsymbol{y}}{||\boldsymbol{y}||})$$

in het geval van een polynomiale kernel.

2.2.2 Verband met binaire classificatie

Alvorens we het verband met patroonherkenning uit de doeken doen, vermelden we dat het geconstrueerde hypervlak uniek is onder bepaalde voorwaarden. Hiervoor hebben we volgende definitie nodig.

Definitie 2.3 (Separabele data set)**.** *We noemen een data set* $\boldsymbol{X} := \boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ separabel *als er een richtingsvector* $\boldsymbol{w} \in \mathcal{H}$ *bestaat zodat* $\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle > 0$ *voor* $i = 1, \ldots, m$.

Stelling 2.4 (Supporting hypervlak [55])**.** *Als de data set* $\boldsymbol{X}$ *separabel is, dan bestaat er een uniek* supporting hypervlak *met volgende eigenschappen:*

1. *het scheidt de data van de oorsprong, en*
2. *de afstand tot de oorsprong is maximaal.*

Voor elke $\rho > 0$ *wordt het "ondersteunende" hypervlak gevonden door*

$$\min_{\boldsymbol{w} \in \mathcal{H}} \quad \frac{1}{2} ||\boldsymbol{w}||^2 \quad \textit{met} \quad \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq \rho, \quad i = 1, \ldots, m.$$

[4]Om normalisatie zonder verlies aan informatie door te voeren, kan men volgend standaard trucje [50] toepassen: introduceer een extra component gelijk aan 1 en normaliseer deze uitgebreide vector, formeel

$$\boldsymbol{x}' = \frac{(\boldsymbol{x}, 1)}{||(\boldsymbol{x}, 1)||}.$$

Wanneer we de Gaussische kernel gebruiken, dan is elke data set na mapping naar de feature ruimte separabel, aangezien alle voorbeelden in het zelfde orthant liggen[5], en bijgevolg bovenstaande stelling steeds van toepassing is. In woorden, we zullen steeds een oplossing vinden, inclusief het *hard margin* geval wanneer we dus geen fouten tolereren.

Om dit zelfde effect te bereiken met de polynomiale kernel, dienen we de data te normaliseren alvorens de routine op te roepen. Om ervoor te zorgen dat de evaluatie van de kernel voor willekeurige vectoren steeds positief is, moet bovendien in het geval van oneven graad (en dus ook in het lineaire geval) de data eerst herschaald worden naar het interval $[0, +\infty)$, alvorens normalisering toe te passen. Wat de uiteenzetting in Sectie 2.1.2 bijtreedt.

Volgend resultaat verheldert de relatie tussen single-class classificatie en binaire classificatie.

Stelling 2.5 (Verband met pattern recognition)**.**

1. *Stel dat* $(\boldsymbol{w}, \rho)$ *het supporting hypervlak parametriseert voor de data* $\boldsymbol{X}$*, dan parametriseert* $(\boldsymbol{w}, 0)$ *het optimale hypervlak dat de gelabelde data set*

$$\{(\boldsymbol{x}_1, 1), \ldots, (\boldsymbol{x}_m, 1), (-\boldsymbol{x}_1, -1), \ldots, (-\boldsymbol{x}_m, -1)\}$$

   scheidt.

2. *Stel* $(\boldsymbol{w}, 0)$ *het optimale scheidende hypervlak is dat door de oorsprong gaat voor de gelabelde data set*

$$\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}, \quad y_i \in \{\pm 1\} \text{ voor } i = 1, \ldots, m,$$

   zo georiënteerd dat $\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle$ *positief is als* $y_i = 1$*. Onderstel bovendien dat* $\rho / ||\boldsymbol{w}||$ *de margin is van het optimale hypervlak, dan geldt er dat* $(\boldsymbol{w}, \rho)$ *het ondersteunende hypervlak voor de data set zonder labels* $\{y_1 \boldsymbol{x}_1, \ldots, y_m \boldsymbol{x}_m\}$*.*

Een intuïtieve verklaring voor dit resultaat gaat schuil in het feit dat we telkens de data transformeren door een puntspiegeling ten opzichte van de oorsprong uit te voeren. Voor een meer gedetailleerd bewijs kan men bij [55] te rade gaan.

[5]Om dit in te zien, volstaat het op te merken dat het scalair product gelijk is aan de cosinus van de ingesloten hoek, want in feature space worden de vectoren automatisch genormeerd door de RBF kernel. Aangezien

$$\forall x, y \in \mathcal{X} \quad \cos(\angle(\Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}))) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle = K(\boldsymbol{x}, \boldsymbol{y}) > 0$$

wil dit niets anders zeggen dat de hoek tussen 2 willekeurige vectoren kleiner is als $\pi/2$ en ze dus in hetzelfde orthant liggen.

Het bewijs steunt op de uniciteit van het ondersteunende hypervlak, maar kan veralgemeend worden naar niet-separabele problemen. In dat geval laten de *margin errors* (punten die ofwel in het verkeerde halfvlak liggen of in de marge terecht komen) bij binaire classificatie zich vertalen tot *outliers* bij single-class classificatie. Afhankelijk van eigen voorkeur zijn dit ofwel de punten die aan de verkeerde kant van het hypervlak liggen, dan wel punten die zich buiten de hoog dimensionale sfeer bevinden. Desalniettemin blijft Stelling 2.5 gelden wanneer men deze "probleempunten" uitsluit.

De bruikbaarheid van Stelling 2.5 schuilt in het feit dat dankzij het verband een aantal resultaten afkomstig uit de wereld van binaire classificatie zonder meer kunnen worden overgedragen naar het single-class scenario. We zullen hiervan handig gebruik maken bij het verklaren van de betekenis van parameter C (of ν voor de puristen). De lezer zal geduld moeten uitoefenen tot Sectie 3.4 alvorens wee ons hieraan wagen. We hebben immers eerst broodnood aan intuïtie en stellingen over bovengrenzen van de generalisatie fout.

2.3 Negative SVDD – toch binaire classificatie?

Wanneer negatieve voorbeelden (objecten die moeten verworpen worden) beschikbaar zijn, kan men deze opnemen in de training set om de beschrijving te verbeteren. In de literatuur [54, 65] treffen we een aantal mogelijke oplossingen om deze voorkennis mee in het model op te nemen.

Een eerste mogelijke oplossing, vormt een onmiddellijke veralgemening van SVDD. In tegenstelling tot de target samples, waarvan we willen dat deze binnen de sfeer vallen, verwachten we dat negatieve data zich er buiten bevinden. Dit model verschilt met dat van een klassieke Support Vector Classifier (SVC) dat we steeds een gesloten gebied verkrijgen rondom de positieve klasse. Bovendien maakt een SVC enkel onderscheidt tussen twee (of meerdere) klassen en is het niet in staat outliers te detecteren die tot geen van de klassen behoren.

De uitwerking gaat als volgt. Veronderstel dat de target set respectievelijk de outliers, de labels +1 en -1 dragen en gemakkelijkheidshalve lopen we in het eerste geval over de indices i, j en anderzijds over k, l. We laten zowel fouten toe op de verzameling van doelobjecten als outliers en introduceren hiervoor slack variabelen ξ_i en ξ_k, de objectieve functie (2.6) passen we als volgt aan:

$$F(R, \boldsymbol{a}, \boldsymbol{\xi}) = R^2 + C_1 \sum_i \xi_i + C_2 \sum_k \xi_k.$$

Wat de nevenvoorwaarden betreft, dienen we uit te drukken dat objecten uit de target set $\boldsymbol{x}_i$ voornamelijk binnen de sfeer moeten vallen en de outliers

$\boldsymbol{x}_k$ er buiten en voeren dus volgende veranderingen door aan (2.7)

$$\begin{aligned} ||\boldsymbol{x}_i - \boldsymbol{a}||^2 &\leq R^2 + \xi_i & \xi_i &\geq 0 & \forall i \\ ||\boldsymbol{x}_k - \boldsymbol{a}||^2 &\geq R^2 + \xi_k & \xi_k &\geq 0 & \forall k. \end{aligned}$$

Om de voorwaarden van het duale probleem te vinden leiden we de, deze keer achterwege gelaten Lagrangian, af naar $R, \boldsymbol{a}, \boldsymbol{\xi}$, en vinden we

$$\begin{aligned} \sum_i \alpha_i - \sum_k \alpha_k &= 1 \\ \boldsymbol{a} = \sum_i \alpha_i \boldsymbol{x}_i - \sum_k \alpha_k \boldsymbol{x}_k & \\ 0 \leq \alpha_i \leq C_1 \ , \ 0 \leq \alpha_k \leq C_2. & \end{aligned}$$

Dit alles resulteert in volgend duale probleem:

$$\begin{aligned} W(\boldsymbol{\alpha}) = &\sum_i \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x}_i \rangle - \sum_k \alpha_k \langle \boldsymbol{x}_k, \boldsymbol{x}_k \rangle - \sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\ &+ 2 \sum_{i,k} \alpha_i \alpha_k \langle \boldsymbol{x}_i, \boldsymbol{x}_k \rangle - \sum_{k,l} \alpha_k \alpha_l \langle \boldsymbol{x}_k, \boldsymbol{x}_l \rangle \end{aligned}$$

Als we tot slot nog nieuwe variabelen $\alpha_i' = \alpha_i y_i$, waarbij nu i loopt over de ganse training set, dan is SVDD met negatieve voorbeelden identiek aan standaard SVDD.

Om outliers toe te laten in de training set bij de ν-SCC, maken we gebruik van een trucje uit verband met binaire classificatie en spiegelen we de data ten opzichte van de oorsprong. Men zou ook het speciale geval uit [54] kunnen gebruiken, waarbij we nu niet de data scheiden van de oorsprong maar van het gemiddelde van een bepaalde verzameling.

Dit alles kan de vraag doen rijzen: waarom dan niet gewoon een binaire classificatie uitvoeren in plaats van een nieuwe methode te hanteren? Zoals in de inleiding reeds aangehaald kan het zijn dat we enkel over data beschikken die het happy-day scenario uittekenen en dat het onmogelijk – dan wel vrij kostelijk – is om abnormale gevallen op te meten. Als we echter wel over zulke data beschikken, maar in mindere mate kan het misschien beter zijn om deze data te gebruiken ter verscherping van onze beschrijving en niet zo zeer als tweede klasse te beschouwen. Immers, niets garandeert ons dat deze kleine verzameling aan outliers echt representatief zullen zijn voor de problemen die zich in de praktijk kunnen voordoen. Een extra argument zou er uit kunnen bestaan dat indien we niet in staat zijn outliers te meten om welke reden dan ook, we misschien wel met onze domein kennis intuïtieve/artificiële outliers zou kunnen verzinnen om de generalisatie eigenschappen van ons model aan te scherpen, zonder dat deze even zwaar door wegen als bij binaire of multi-class classificatie.

Ter afsluiting van dit hoofdstuk, wil ik er op wijzen dat de ontwikkeling naar de ideale one-class classifier nog steeds aan de gang is. Recentelijk werd in [36] bijvoorbeeld lokale dichtheidsgraad gebruikt, onder het motto "hoe meer datapunten zich in een bepaalde omgeving bevinden des te belangrijker deze punten zijn". Om dit te realiseren werd een *density-induced* metriek gebruikt, waarbij SVDD en nearest neighbour classificatie elkaar te gemoed komen. Kleine voorspelling van mijnentwege, weldra volgt de verweving van zowel negative SVDD en density SVDD.

Hoofdstuk 3

Selectie van (hyper)parameters

Het doel van dit hoofdstuk is dieper in te gaan op enkele praktische methoden die toelaten een zinvolle keuze van de modelparameters te maken. Chapelle et al. [14] ontketenden een stroom aan publicaties [2, 3, 17, 20, 25, 35, 52] door als eersten het ingenieuze idee te hebben om deze parameters te bepalen door toepassing van een principe beter gekend onder de naam *minimax* benadering: waarbij men enerzijds de margin van het uiteindelijk te construeren hypervlak tracht te maximaliseren en de *generalisation error* te minimaliseren. De verschillende methoden die worden besproken hangen enerzijds af van de geopteerde methode voor het schatten van de performantie van de support vectorm machine en anderzijds van de gebruikte optimalisatietechniek: optimalisatie zonder nevenvoorwaarden (bijvoorbeeld een of andere vorm van gradient descent, (Quasi-)Newton, ...) of met nevenvoorwaarden (quadratic programming (QP) [52]).

Aangezien het huidige onderzoek zich tot hiertoe hoofdzakelijk geconcentreerd heeft op het uitwerken van problemen voor classificatie, pas ik dit in navolging eveneens toe op Support Vector Machines (SVM). Desalniettemin ben ik er sterk van overtuigd dat deze methoden in de toekomst aanleiding zullen geven tot methoden voor het geval van regressie en in mindere mate de one-class classificatie. Voor regressie werd onlangs aanvang genomen met [11, 15].

Wat Support Vector Data Description (SVDD) betreft, ligt het inschatten van generalisatie eigenschappen iets moeilijker: we beschikken immers enkel over gegevens van de target set. Om ook hier een vorm van minimax procedure te hanteren kan men de heuristieken uit [63, 64] gebruiken, waarbij men veronderstelt dat de outliers zich hetzij in een box, hetzij in een sfeer rondom de target set bevinden. We zullen ons in Sectie 3.4 dan ook beperken

tot de beloofde verklaring van de modelparameter C en enkele heuristieken beschrijven om de spreidingsparameter σ van de RBF kernel te kiezen.

3.1 Belang keuze van parameters

De voornaamste reden waarom we de parameters willen optimaliseren, spreekt voor zich: de kwaliteit van de uiteindelijk bekomen SVM hangt immers rechtstreeks af van de keuze van de parameters. Dit is enerzijds de regulatie parameter C en anderzijds de parameters die we dienen in te stellen voor een specifieke kernel. Echter het bepalen van deze parameters is geen triviale aangelegenheid en vereist veel ervaring. Ervaring, die wanneer we de techniek van support vector machines uiteindelijk willen consumeren als blackbox methode in gebieden waar achtergrond kennis minimaal is, uiteraard ontbreekt.

De tot nog toe meest gehanteerde techniek bestaat eruit om de optimale parameters te selecteren met behulp van een *grid search*, een *exhaustive* zoekstrategie waarbij de parameter ruimte opgesplitst wordt in een aantal vakjes (vandaar de benaming) en men telkens een nieuwe SVM traint met een nieuwe combinatie van parameters. Desondanks dat deze methode veelal leidt tot goede resultaten, is het grootste nadeel natuurlijk het verslinden van CPU cycles daar telkens opnieuw een kwadratisch optimalisatieprobleem dient opgelost te worden voor elk stel parameters. Wanneer het aantal parameters groter wordt dan twee, is deze methode computationeel gezien onhandelbaar en moeten we dus opzoek naar efficiëntere methoden.

Volgens [14] zou dit tevens één van de hoofdredenen zijn, dat we ons behelpen met een beperkt aantal eenvoudige kernels. Wanneer het selecteren van de parameters aantrekkelijker wordt, kunnen we in staat zijn om meer complexe, probleem specifieke kernels in te pluggen in de hoop nog betere resultaten te vergaren.

Een andere mogelijkheid die hier nauw bij aansluit, is het gebruik van *scaling factors*. In tegenstelling tot het geval van de sferische RBF kernel die gecontroleerd wordt door één enkele parameter die de variantie/breedte van de kernel aangeeft, zouden we elk attribuut in de input ruimte kunnen wegen met een schaalfactor die de variantie in een welbepaalde richting beschrijft

$$K(\boldsymbol{x}, \boldsymbol{z}) = \exp\Big(- \sum_i \frac{(x_i - z_i)^2}{2\sigma_i^2} \Big). \tag{3.1}$$

Aangezien support vector machines vaak gebruikt worden bij data met hoge dimensionaliteit, is het fijntunen van zulke kernel met de grid techniek uitgesloten.

Een laatste voordeel en tevens ook een handige toepassing van de scaling factors is *feature selectie*. Hierbij zou men immers de resultaten van de optimalisatie van de *scaled* RBF kernel kunnen gebruiken om de data te preprocessen en de attributen met te grote waarden voor σ_i^2 uitsluiten, het geheel opnieuw trainen en dit blijven herhalen tot er geen onbelangrijke features in de data set aanwezig zijn. Een meer volledige omschrijving bevindt zich in Sectie 4.3.

3.2 Schatting generalisatie fout

Als we spreken over optimale parameters, moeten we beschikken over criteria, in het vervolg kortweg genoteerd als een functie T, die een bepaalde SVM kan verkiezen boven een andere. Om hierover te oordelen, gaan we kijken naar de generalisatie eigenschappen van de gevonden oplossing. Hiervoor werd het begrip *generalisatie fout* ingevoerd, wat een maat is voor de prestatie die onze getrainde SVM zal bereiken wanneer toekomstige data wordt aangeboden en zal in dit werk genoteerd worden als $E_{p_{err}^m}$, de kans dat bij testen een fout wordt gemaakt wanneer we onze machine getrained hebben met m voorbeelden. Het uiteindelijke doel is ervoor zorgen dat we de margin maximaliseren, terwijl we de generalisatie fout minimaliseren.

We kunnen dit "meten" vanuit twee invalshoeken.

3.2.1 Validatie fout

Ten eerste, wanneer men over voldoende training samples beschikt, kan men ervoor kiezen om de training set in twee delen op te splitsen en het aantal fouten op deze validatie set als maat voor de generalisatie fout beschouwen. Deze wordt wiskundig op volgende manier bondig gevat als

$$T = \frac{1}{N} \sum_{i=1}^{N} \Psi(-y_i f(\boldsymbol{x}_i)), \tag{3.2}$$

waarbij Ψ de stapfunctie is, $\Psi(x) = 1$ als $x > 1$ en 0 elders, f de gekende beslissingsfunctie, y_i het label van de geëvalueerde vector $\boldsymbol{x}_i$ en N de grootte van de validatie set.

Deze vorm van schatting is *unbiased* (onder de veronderstelling dat de aangeboden testdata representatief is voor het toekomstige werk) en de variantie neemt af naar mate de validatie set groter wordt.

Om reeds een vooruitblik te werpen op wat volgt, is het handig nu reeds op te merken dat het gebruik van de stapfunctie ervoor zorgt dat we te maken hebben met een discontinue schatting. Immers, bij elke fout vindt een

sprongetje van $1/N$ plaats. Willen we nadien het minimum bepalen van deze error functie, zonder gebruik te maken van brute force algoritmen, zullen we een gladde variant moeten introduceren. Kortom, een benadering die we nadien naar hartelust kunnen differentiëren. Meer hierover in Sectie 3.3.3.

3.2.2 Leave-one-out bovengrenzen

Wanneer men niet over de luxe bezit, aanspraak te kunnen maken op een extra validatie set, zal men in het algemeen zijn toevlucht zoeken tot wat men samenbalt in de categorie *leave-one-out error estimates*, waarbij men bovengrenzen van de generalisatie fout analytisch uitdrukt in functie van de parameters.

De *leave-one-out* procedure bestaat eruit herhaaldelijk één element uit de training set te verwijderen, de beslissingsfunctie op te stellen en te testen of het weggelaten element correct geclassificeerd wordt. Op deze manier test men alle m objecten, van de beschouwde training set, met behulp van m verschillende beslissingsregels. We noteren het aantal fouten in de leave-one-out procedure door $\mathcal{L}(\boldsymbol{x}_1, y_1, \ldots, \boldsymbol{x}_m, y_m)$. Het is algemeen geweten dat de verwachtingswaarde van deze schatter, E_{LOO}, een *bijna* unbiased schatting oplevert van de verwachte *generalisation error*. De "bijna" slaat hier op het feit dat we een unbiased schatting krijgen voor een training set van grootte $m-1$ in plaats van m.

Stelling 3.1 (Luntz en Brailovsky)**.**

$$E_{p_{err}^{m-1}} = \frac{1}{m} E[\mathcal{L}(\boldsymbol{x}_1, y_1, \ldots, \boldsymbol{x}_m, y_m)] =: E_{LOO},$$

met $E_{p_{err}^{m-1}}$ de kans op een test error van een machine getraind op een sample van grootte $m-1$ en de verwachtingen genomen over een random keuze van zowel training- als test-samples.

De lezer kan het bewijs nalezen in [68].

Hoewel dit resultaat de leave-one-out estimator promoveert tot goede keuze voor de schatting van de generalisatie fout, is het spijtig genoeg een zeer kostelijke bedoening om deze effectief te berekenen aangezien het evenveel trainingen vereist als het aantal trainingssamples. Onder andere in [9, 49, 59, 67] wordt aandacht besteed aan *online* (ook wel *incremental/decremental*) Support Vector Machines, waarbij gezocht wordt naar de analytische aanpassing van de verkregen oplossing bij toevoeging of verwijdering van een sample, wat de exacte leave-one-out procedure efficiënter zal maken.

Wat deze sectie betreft, zullen we ons tevreden stellen met bovengrenzen of benaderingen van de schatter die eenvoudig te berekenen zijn en indien

mogelijk uit te drukken vallen in een analytische expressie in functie van de keuzeparameters.

Voor een volledig overzicht verwijzen we naar [13, 14] en beperken ons hier tot het vermelden van enerzijds *Support Vector Count*, nodig voor de gevoelsmatige verklaring van de modelparameter bij single-class clasificatie in Sectie 3.4, en anderzijds *Radius/Margin Bound*, een computationeel aantrekkelijke bovengrens die een flashback naar het vorige hoofdstuk zou moeten opwekken.

Support Vector Count

Stelling 3.2 (Support Vector Count [56])**.** *Voor hypervlakken die voldoen aan de maximum margin voorwaarde geldt:*

$$E_{p_{err}^{m-1}} \leq \frac{N_{SV}}{m}, \tag{3.3}$$

waarbij N_{SV} het aantal support vectors is.

Radius/Margin Bound

Stelling 3.3 (Radius/Margin Bound [68])**.** *Voor optimale hypervlakken zonder* threshold *(i.e. die door de oorsprong gaan) en zonder trainingsfouten (i.e.* hard-margin SVMs*) geldt:*

$$E_{p_{err}^{m-1}} \leq \frac{E\left(\frac{\mathcal{D}_m}{\rho_m}\right)^2}{m} \tag{3.4}$$

waarbij $\mathcal{D}_m = \max_i ||\boldsymbol{x}_i||$ en ρ de margin van het hypervlak.

Deze stelling kan veralgemeend [14] worden naar hypervlakken met een threshold parameter b verschillend van 0, waarbij nu niet de diameter van de sfeer rond de input data van belang is, maar de straal van de sfeer in de feature space in rekening wordt gebracht

$$T = \frac{1}{4m}\frac{R^2}{\rho^2} = \frac{1}{4m}R^2\,||\boldsymbol{w}||^2\,. \tag{3.5}$$

Na het doorgronden van Hoofdstuk 2 is het bepalen van

$$R^2 = \min_{\boldsymbol{a},\boldsymbol{x}_i} ||\Phi(\boldsymbol{x}_i) - \boldsymbol{a}||^2$$

kinderspel en komt overeen met oplossen van de *rigid sphere SVDD* waarbij we dus geen fouten toestaan:

$$\underset{\boldsymbol{\beta}\in\mathbb{R}^m}{\text{maximaliseer}} \quad W(\boldsymbol{\alpha}) = \sum_i \beta_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{i,j} \beta_i \beta_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{3.6}$$

$$\text{met} \sum_{i=1}^{m} \beta_1 = 1 \qquad i = 1, \ldots, m \tag{3.7}$$

$$\beta_i \geq 0 \qquad i = 1, \ldots, m \tag{3.8}$$

De Radius/Margin bound, ook wel R^2W^2 genoteerd, geldt ook voor problemen opgelost met behulp van L2-SVM, waarbij we weer enkel de kernel hoeven te vervangen door $K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{1}{C}\delta_{ij}$. Wat betreft SVMs waarbij de fouten lineair worden verrekend, treedt er bezwaar op om deze grens te gebruiken [20].

Immers, we kunnen het volgende aantonen.

Lemma 3.4. *$R^2 ||\boldsymbol{w}||^2$ gaat naar nul wanneer C en σ^2 naar nul respectievelijk oneindig gaan bij een Gaussische kernel.*

Bewijs. Aangezien $R^2 \leq 1$ (want, R^2 is gelijk aan (3.6) en deze uitdrukking kunnen we bij de Gaussische kernel begrenzen door 1), de α_i's begrensd worden door C en $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ naar 1 gaat wanneer σ^2 naar oneindig gaat, verkrijgen we het resultaat:

$$\begin{aligned}
R^2 ||\boldsymbol{w}||^2 &\leq ||\boldsymbol{w}||^2 \\
&\leq \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
&\leq \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
&\leq \sum_{i,j=1}^{m} \alpha_i \alpha_j \\
&\leq \sum_{i,j=1}^{m} C^2 \\
&\leq m^2 C^2.
\end{aligned}$$

Per veronderstelling gaat C naar 0, waaruit het gestelde volgt. □

Om dit te corrigeren kan men vervangen T door $\frac{R^2||\boldsymbol{w}||^2 + N_{SV}}{m}$, waarbij het probleem wordt omzeild bij de limietgevallen bij RBF kernels. Deze uitdrukking reduceert zicht dan gewoonweg tot de support vector count.

Ter volledigheid vermelden we dat deze voor de uiteindelijke optimalisatie niet gebruiken en bij de implementatie geopteerd werd voor de aangepaste bovengrens uit [17].

Aangepaste (L1-)Radius/Margin Bound

Wat betreft deze "lichtjes" gewijzigde bound voor L1-SVM, beperken we ons hier enkel tot het formuleren van de bovengrens

$$T = \frac{1}{m}(R^2 + \frac{\Delta}{C})(||\boldsymbol{w}||^2 + 2C\sum_i \xi_i) \tag{3.9}$$

met Δ een positieve constante dichtbij 1 en de ξ_i de multipliers voor het incapsuleren van de lineaire foutenterm in de Lagrangian. Voor meer details is het handig de paper [17] erop na te lezen. Daar wordt bewezen dat deze uitdrukking differentieerbaar is, worden ook de partiële afgeleiden volledig uitgewerkt en is bovendien een volledige uiteenzetting gewijd aan het uitspelen van de verschillende R^2W^2 bovengrenzen tussen L1-SVM en L2-SVM.

3.3 Optimaliseren van de kernel parameters

We onderstellen dat de kernel K afhangt van één of meerdere parameters voorgesteld door de vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$. Het doel is nu om de waarden voor $\boldsymbol{\alpha}$ en $\boldsymbol{\theta}$ te vinden zodat W wordt gemaximaliseerd (maximum margin algoritme) en T, het selectie criteria, geminimaliseerd (beste kernel parameters). Of nog formeel, voor een vaste $\boldsymbol{\theta}$, willen we

$$\boldsymbol{\alpha}^* = \arg\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}, \boldsymbol{\theta})$$

en we kiezen $\boldsymbol{\theta}^*$ zodat

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} T(\boldsymbol{\alpha}^*, \boldsymbol{\theta}).$$

De meest gebruikte techniek, maar tegelijk ook de computationeel minst aantrekkelijke methode voor het kiezen van de parameters, is een combinatie van een zoekalgoritme en *k-fold cross validation*. De procedure gaat als volgt: splits de training set random in k delen van ongeveer dezelfde grootte, train k keer een SVM door telkens een deel weg te laten en gebruik deze om de accuracy te evalueren. In het extreme geval waarbij we k gelijk aan m kiezen, komen we terug bij de leave-one-out procedure. Met behulp van bijvoorbeeld een gridsearch, doorlopen we dit "lusje" voor elke combinatie

van de parameters en kiezen uiteindelijk voor diegenen waarbij de error rate minimaal is.

De belangrijkste reden voor het gebruik van deze techniek is het feit dat deze methode veelal leidt tot de beste resultaten. Daarenboven wordt hierbij enkel informatie gebruikt over de beslissingsfunctie zelf. Het grootste nadeel is echter dat wanneer het aantal parameters twee overschrijdt, de methode onhandelbaar wordt. Dit mag ons niet verbazen want het is een rechtstreekse toepassing van de *curse of dimensionality*. Stel immers dat we voor elke parameter 10 keuzemogelijkheden hebben. In 2 dimensies moeten we dan 100 koppeltjes nagaan, en we moeten dit voor elk van de k folds herhalen. Dit brengt ons totaal van SVM trainingen op $100k$. Stel dat we nu gebruik willen maken van een scaled RBF kernel, waarbij het aantal parameters gelijk is aan de dimensie van de attributen. Kleine data sets hebben dan al gemakkelijk minstens 10 features en moeten we dus minstens $10^{10}k$ trainingen uitvoeren alvorens we de meest optimale waarde kunnen selecteren.

Om de factor k, afkomstig van het cross-validatie proces, kwijt te spelen, kan men in plaats van de test error één van de reeds vermelde bovengrenzen gebruiken. De *exhaustive* gridsearch zou vervangen kunnen worden door andere zoekmethoden, zoals bijvoorbeeld recentelijk onderzocht in [16]. Zij kwamen tot volgende algemene conclusie:

> No method performs best in every case but pattern search seems to be a good compromise between accuracy and computational effort. On the other hand grid- and random search are easily parallelized so if computational time is important and many processors are available grid search may be the method of choice.

Desondanks de opkomst van multiple processor machines en de tendens van alsmaar sneller wordende computers, is het toch nuttig om gebruik te maken van andere standaard technieken uit de optimalisatiewereld. In 2001 kwamen Chapelle et al. met het idee op de proppen om gebruik te maken van niet-lineaire optimalisatiemethoden die buiten de objectieve functie zelf, ook nog gebruik maken van de gradiënt van de bovengrenzen.

Algoritme 5 schetst het algemene framework.

3.3.1 Helpende hand bij berekening van de gradiënten

Volgend lemma zal ons heel wat zorgen besparen bij het berekenen van de gradiënten (in functie van de kernel parameters) voor verschillende schattingen van de generalisatie fout.

Algoritme 5 Selecteren van kernel parameters met behulp van gradiënt informatie

1: Initialiseer $\boldsymbol{\theta}$.
2: Gebruik een standaard SVM algoritme, om het maximum te vinden van de kwadratische vorm W:

$$\boldsymbol{\alpha}^* = \arg\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}, \boldsymbol{\theta})$$

3: Update de parameters $\boldsymbol{\theta}$ zodanig dat T geminimaliseerd wordt (gebruik makend van gradiënt informatie).
4: Hervat stap 2 of stop wanneer minimum van T bereikt wordt.

Lemma 3.5. *Gegeven een $(n \times 1)$ vector $\boldsymbol{\nu}_\theta$ en een $(n \times n)$ matrix $\boldsymbol{P}_\theta$ die* smooth *afhangen van een parameter θ, i.e. differentieerbaar zijn naar θ. Beschouw de functie:*

$$L(\theta) = \max_{\boldsymbol{x} \in F} \boldsymbol{x}^T \boldsymbol{\nu}_\theta - \frac{1}{2} \boldsymbol{x}^T \boldsymbol{P}_\theta \boldsymbol{x}$$

waarbij

$$F = \{\boldsymbol{x} \,:\, \boldsymbol{b}^T \boldsymbol{x} = c,\, \boldsymbol{x} \geq 0\}$$

Laat $\boldsymbol{x}^*$ de vector zijn waar het maximum van $L(\theta)$ wordt bereikt. Als dit optimum uniek is dat geldt

$$\frac{\partial L(\theta)}{\partial \theta} = \boldsymbol{x}^* \frac{\partial \boldsymbol{\nu}}{\partial \theta} - \frac{1}{2} \boldsymbol{x}^{*T} \frac{\partial \boldsymbol{P}_\theta}{\partial \theta} \boldsymbol{x}^*$$

Of anders gezegd, het is mogelijk L te differentiëren in functie van θ alsof $\boldsymbol{x}^*$ onafhankelijk is van θ.

Bemerk dat dit ook waar is als men één (of beide) voorwaarde(n) in de definitie van de feasible set F achterwege laat.

De eenvoudigste manier om dit resultaat te leren appreciëren is door een functie $f(u) = \min_\alpha F(u, \alpha)$ te beschouwen. Stel dat $\boldsymbol{\alpha}^*(u)$ de oplossing is van het optimalisatieprobleem, i.e. $\frac{\partial F}{\partial \alpha^*(u)} = 0$. Dan is $f(u) = F(u, \alpha^*(u))$. En dus, [1]

$$\frac{\partial f}{\partial u} = \frac{\partial F(u, \alpha^*(u))}{\partial u} = \frac{\partial F}{\partial u} + \frac{\partial F}{\partial \alpha}|_{\alpha = \alpha^*(u)} \frac{\partial \alpha^*(u)}{\partial u} = \frac{\partial F}{\partial u}.$$

De gradiënt van f in functie van u verkrijgt men eenvoudig door te differentiëren van F naar u alsof α geen invloed heeft op u. In geval van

[1] Verklaring van de tweede gelijkheid. Neem $F(x, y) = x^2 + y^2 - 1$, dan weten we door toepassing van de Impliciete Functie Stelling dat er een functie $f(x)$ bestaat zodat $F(x, f(x)) = 0$. Wanneer we nu $F(x, y)$ moeten afleiden naar x, moeten we dit eveneens

optimalisatieproblemen met nevenvoorwaarden is de argumentatie iets gecompliceerder en dient men gebruik te maken van resultaten afkomstig uit de perturbatie analyse van optimalisatieproblemen. Neem een kijkje in [14, 52] voor meer details en volledig uitgeschreven versies van het bewijs.

3.3.2 Radius/Margin bound

In het geval van de Radius/Margin bound (3.5) berekenen we de gradiënt van de te minimaliseren schatter als volgt

$$p = 1, \ldots, n: \quad \frac{\partial T}{\partial \theta_p} = \frac{1}{m}\Big(\frac{\partial \left|\left|\boldsymbol{w}\right|\right|^2}{\partial \theta_p} R^2 + \left|\left|\boldsymbol{w}\right|\right|^2 \frac{\partial R^2}{\partial \theta_p}\Big). \tag{3.10}$$

Het is duidelijk dan $\left|\left|\boldsymbol{w}\right|\right|^2$ afhangt van $\boldsymbol{\alpha}$ en dat $\boldsymbol{\alpha}$ op zijn beurt afhangt van de parameter θ_p. Echter omdat $\left|\left|\boldsymbol{w}\right|\right|^2$ berekend werd als oplossing van een optimalisatieprobleem, blijkt door toepassing van Lemma 3.5 dat de gradiënt van $\boldsymbol{\alpha}$ in functie van de hyperparameters niet voorkomt in de gradiënt van $\left|\left|\boldsymbol{w}\right|\right|^2$. Analoge redenering geldt voor de straal (in het kwadraat) van de kleinste sfeer die de trainingsdata (in feature space wel te verstaan) omhult. Hiermee kunnen we $\frac{\partial T}{\partial \theta_p}$ verder uitrekenen door gebruik te maken van volgende formules

$$\frac{\partial \left|\left|\boldsymbol{w}\right|\right|^2}{\partial \theta_p} = -\sum_{i,j} \alpha_i^* \alpha_j^* y_i y_j \frac{\partial K(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial \theta_p} \tag{3.11}$$

$$\frac{\partial R^2}{\partial \theta_p} = \sum_i \beta_i^* \frac{\partial K(\boldsymbol{x}_i, \boldsymbol{x}_i)}{\partial \theta_p} - \sum_{i,j} \beta_i^* \beta_j^* \frac{\partial K(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial \theta_p}. \tag{3.12}$$

---

impliciet doen en doen we eigenlijk het volgende

$$\begin{aligned} 0 = \frac{\partial F(x,y)}{\partial x} = \frac{\partial F(x, f(x))}{\partial x} &= \frac{\partial F}{\partial x} + \frac{\partial F}{\partial y}\Big|_{y=f(x)} \frac{\partial f(x)}{\partial x} \\ &= 2x - 2y|_{y=f(x)} \frac{\partial f(x)}{\partial x} \\ &= 2x - 2f(x)\frac{\partial f(x)}{\partial x} \\ \Rightarrow \frac{\partial f(x)}{\partial x} &= \frac{x}{f(x)}. \end{aligned}$$

Dit stemt overeen met wanneer we F, zoals in dit geval, in expliciete vorm kunnen herschrijven en dan afleiden. Ter controle $f(x) = \sqrt{1-x^2}$ en

$$\frac{\partial f(x)}{\partial x} = \frac{x}{\sqrt{1-x^2}} = \frac{x}{f(x)}.$$

Gewogen Gaussische RBF kernel

Ter illustratie nemen we de Gaussische radial basis functie (RBF) kernel met gewichtsparameter per attribuut

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\Big(-\frac{1}{2}\sum_{k=1}^{n} \frac{(x_{ik} - x_{jk})^2}{\sigma_k^2}\Big).$$

Wanneer we onze trainingsdata nu trainen met een *least squares* SVM, dienen we enkel de kernel aan te passen tot

$$\tilde{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{1}{C}\delta_{ij}.$$

Als parameters hebben we dus $\boldsymbol{\theta} = (C, \boldsymbol{\sigma}^2)$. Wanneer we nu de gradiënt naar C respectievelijk $\boldsymbol{\sigma}^2$ uitrekenen, verkrijgen we

$$\frac{\partial \, ||\boldsymbol{w}||^2}{\partial C} = \frac{\sum_i \alpha_i^{*2}}{C^2} \tag{3.13}$$

$$\frac{\partial \, ||\boldsymbol{w}||^2}{\partial \sigma_k^2} = -\sum_{i,j} \alpha_i^* \alpha_j^* y_i y_j \frac{\partial K(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial \sigma_k^2} \quad k = 1, \ldots, n \tag{3.14}$$

$$\frac{\partial R^2}{\partial C} = \frac{\sum_i \beta_i^* (1 - \beta_i^*)}{C^2} \tag{3.15}$$

$$\frac{\partial R^2}{\partial \sigma_k^2} = -\sum_{i,j} \beta_i^* \beta_j^* \frac{\partial K(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial \sigma_k^2} \quad k = 1, \ldots, n. \tag{3.16}$$

De verdwijning van de eerste term uit (3.12) in (3.16) heeft een eenvoudige verklaring, met name omdat $K(\boldsymbol{x}_i, \boldsymbol{x}_i) = 1$ en de afgeleide van een constante functie de nulfunctie oplevert.

$$\frac{\partial K(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial \sigma_k^2} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) \frac{(x_{ik} - x_{jk})^2}{2\sigma_k^2}$$

drukt ten slotte de partiële afgeleide van de kernel functie uit.

Bij deze bespreking hebben we de beperkingen op de parameters onder de mat geschoven, immers zowel C als $\boldsymbol{\sigma}^2$ dienen strikt positief te zijn. Dit kan opgelost worden met een eenvoudige logaritmische transformatie van de variabelen. Op die manier vermijden we de invoering van extra voorwaarden en kunnen we nog steeds gebruik maken van gekende *unconstrained* optimalisatietechnieken.

In plaats van de parameter ruimte $(C, \boldsymbol{\sigma}^2)$, beschouwen we de ruimte $(\ln C, \ln \boldsymbol{\sigma}^2)$, waardoor de voorwaarden automatisch vervuld worden. Wegens de kettingregel weten we dat

$$\begin{aligned}\frac{\partial f}{\partial \ln x} &= \frac{\partial f}{\partial x}\frac{\partial x}{\partial \ln x} \\ &= \frac{\partial f}{\partial x}\frac{\partial \exp \ln x}{\partial \ln x} \\ &= \frac{\partial f}{\partial x} \exp \ln x \\ &= \frac{\partial f}{\partial x} x.\end{aligned}$$

Uit deze kleine berekening leren we dat we de gradiënt uit (3.10) telkens moeten vermenigvuldigen met de waarde van parameter θ_p in kwestie.

Gaussische RBF kernel

Wanneer we geen onderscheid maken per attribuut en we de keuze van slechts één globale gewichtsparameter σ optimaliseren, krijgen we volgende uitdrukking voor de partieel afgeleide

$$\begin{aligned}\frac{\partial K(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial \sigma^2} &= K(\boldsymbol{x}_i, \boldsymbol{x}_j)\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^4} \\ &= -\frac{1}{\sigma^2} K(\boldsymbol{x}_i, \boldsymbol{x}_j) \ln(K(\boldsymbol{x}_i, \boldsymbol{x}_j)),\end{aligned}$$

waarbij de laatste stap enkel nuttig is voor efficiëntere evaluatie bij een uiteindelijke implementatie.

3.3.3 Validatie set

Wanneer we een voldoende grote training set ter onze beschikking hebben, kunnen we gebruikmaken van de test error die we verkrijgen door een validatie set te evalueren. De basis werd weeral eens gelegd in [14] en een volledig uitgewerkte versie werd aangereikt in [2, 3]. Hierbij wordt opnieuw gebruik gemaakt van niet-lineaire optimalisatiemethoden die gebruik maken van informatie afkomstig van de gradiënt. Daar echter de stapfunctie niet continu is, en bijgevolg zeker niet afleidbaar, moet voor dit ongemak een oplossing gevonden worden. Een standaard techniek is deze functie te benaderen door een continue probabilistische distributie functie.

Daartegenover staat een in meer recentelijk werk [52] beschreven methode, waarbij de niet-afleidbaarheid wordt overwonnen door introductie van slack variabelen en de optimale parameters gevonden worden na een iteratief procédé van telkens opnieuw een kwadratisch optimalisatieprobleem met nevenvoorwaarden op te lossen.

Optimaliseren door smoothen van stapfunctie

We gaan in deze subsectie nader in op de eerste methode.

Wanneer een *gradient descent* methode gehanteerd wordt, dan veronderstelt men dat foutenschatter afleidbaar is. Echter in (3.2) wordt gebruik gemaakt van de stapfunctie, waarvan we weten dat deze niet continue is, laat staan differentieerbaar. Om dit ongemak te overwinnen, is het mogelijk om de stapfunctie te benaderen met een logistieke functie van de vorm

$$\Psi(x) = \frac{1}{1 + \exp(-Ax + B)}, \tag{3.17}$$

waarbij de constanten[2] A en B bepaald worden door een schatting van de *posterior probabilities* [14] en waardoor we een gladde benadering krijgen van de test error functie.

We vervangen $\Psi(-y_i f(\boldsymbol{x}_i))$ uit (3.2) door

$$\frac{1}{1 + \exp(Ay_i f(\boldsymbol{x}_i) + B)} \tag{3.18}$$

en verkrijgen volgende te minimaliseren schatter voor de generalisatie fout

$$T = \frac{1}{m} \sum_{i=1}^{m} \Big(1 + \exp(Ay_i f(\boldsymbol{x}_i) + B)\Big)^{-1}. \tag{3.19}$$

Aangezien de benaderde stapfunctie wel afleidbaar is, kunnen we hiervan de afgeleide naar richting $\boldsymbol{\theta}$ als volgt berekenen.

Met behulp van de kettingregel[3] kunnen we dit als volgt opsplitsen

$$\frac{\partial T}{\partial \boldsymbol{\theta}} = \frac{\partial T}{\partial \boldsymbol{\alpha}} \frac{\partial \boldsymbol{\alpha}}{\partial \boldsymbol{\theta}}, \tag{3.20}$$

hierbij is $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$ de vector van de multipliers en k het aantal support vectors. De reden waarom enkel de support vectors in rekening worden gebracht is vrij eenvoudig in te zien, immers bij de andere vectoren zal α_i gelijk zijn aan 0 en zal de afgeleide sowieso 0 opleveren.

De componenten van de $(1 \times k)$ vector $\frac{\partial T}{\partial \boldsymbol{\alpha}}$, kan men berekenen met volgende

[2]Hierbij onderstellen we dat de parameter A positief is, om er voor te zorgen dat we met een stijgende functie werken.

[3]Kettingregel geldt immers ook voor vectoriële functies, waarbij we in dit geval $(1 \times n)$ vector opsplitsen in product van $(1 \times k)$ vector en een $(k \times n)$ matrix. n en k interpreteren we als het aantal te optimaliseren parameters en het aantal support vectors.

expressie

$$\begin{aligned}\frac{\partial T}{\partial \alpha_j} &= \frac{1}{m}\sum_{i=1}^{m}\frac{\partial\Big(1+\exp(Ay_i f(\boldsymbol{x}_i)+B)\Big)^{-1}}{\partial\alpha_j}\\ &= \frac{1}{m}\sum_{i=1}^{m}\frac{Ay_i\frac{\partial f(\boldsymbol{x}_i)}{\partial\alpha_j}}{\Big(1+\exp(Ay_i f(\boldsymbol{x}_i)+B)\Big)^2}\\ &= \frac{1}{m}\sum_{i=1}^{m}\frac{Ay_i y_j K_{\boldsymbol{\theta}}(\boldsymbol{x}_j,\boldsymbol{x}_i)}{\Big(1+\exp(Ay_i f(\boldsymbol{x}_i)+B)\Big)^2}.\end{aligned}$$

Rest er ons enkel nog de afgeleide van de multipliers vector $\boldsymbol{\alpha}$ naar parameter vector $\boldsymbol{\theta}$ uit te rekenen. In [14] werd volgende formule bewezen

$$\frac{\partial(\boldsymbol{\alpha}^*,b)}{\partial\theta_p} = -\boldsymbol{H}^{-1}\frac{\partial\boldsymbol{H}}{\partial\theta_p}(\boldsymbol{\alpha}^*,b)^T, \tag{3.21}$$

met $\boldsymbol{H}$ een vierkante $(k+1\times k+1)$ matrix die uitdrukt wat het betekent om op de margin te liggen:

$$\underbrace{\begin{pmatrix}\boldsymbol{K}^Y & \boldsymbol{Y}\\ \boldsymbol{Y}^T & \boldsymbol{0}\end{pmatrix}}_{\boldsymbol{H}}\begin{pmatrix}\boldsymbol{\alpha}^*\\ b\end{pmatrix} = \begin{pmatrix}\boldsymbol{1}\\ 0\end{pmatrix}$$

waarbij $\boldsymbol{K}^Y_{ij} = y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)$. In woorden uitgedrukt, zegt dit niets anders dan dat de punten correct geclassificeerd moeten zijn (weer enkel voor hard-margin SVM en L2-SVM!) en dat ze aan de voorwaarde moeten voldoen dat $\sum_{i=1}^m \alpha_i y_i = 0$. Om nu aan (3.21) te geraken, vermenigvuldigen we beide leden met de inverse van $\boldsymbol{H}$ en gebruiken de eenvoudig te bewijzen eigenschap

$$\frac{\partial\boldsymbol{M}^{-1}}{\partial x} = -\boldsymbol{M}^{-1}\frac{\partial\boldsymbol{M}}{\partial x}\boldsymbol{M}^{-1},$$

met $\boldsymbol{M}$ een willekeurige inverteerbare matrix die afhangt van een parameter x.

De reden waarom we steeds enkel rekening dienen te houden met de support vectors, is wederom een gevolg van feit dat wanneer we een niet-support vector uit de training set achterwege houden er niets aan de oplossing verandert.

De pientere lezer merkt bovendien op dat dit niet echt is wat we nodig hebben (de term b zou immers roet in het eten kunnen gooien). Toch is het eenvoudig in te zien dat dit eigenlijk niets uitmaakt, breiden we $\boldsymbol{\alpha}$ uit met extra component b en herhalen we het verhaal, dan moeten we enkel rekening houden met feit dat $\frac{\partial f_i}{\partial b} = 1$ bij het berekenen van $\frac{\partial T}{\partial\boldsymbol{\alpha}}$.

Samenvattend:

Algoritme 6 Selecteren van kernel parameters met differentieerbare benadering van de stapfunctie

1: Initialiseer $\boldsymbol{\theta}$.
2: Train SVM met deze $\boldsymbol{\theta}$.
3: Bereken de sigmoid parameters A en B (Newton optimalisatiemethode).
4: Evalueer functie en gradiënt van de differentieerbare variant van de validatie error.
5: Update $\boldsymbol{\theta}$ door gebruik te van een optimalisatie routine die informatie uit vorige stap gebruikt.
6: Hervat stap 2 of stop wanneer minimum bereikt wordt.

Optimaliseren van empirische test error

In [2] gebruikt men Algoritme 7 als leidraad, waarbij men afstapt van het criterium dat de effectieve fout tijdens validatie uitdrukt en deze vervangt door de kans op fouten, weliswaar eveneens berekent aan de hand van een aparte validatie set. Dit vereist natuurlijk een meer diepgaande bespreking.

Algoritme 7 Selecteren van kernel parameters met behulp van posterior probabilities

1: Initialiseer $\boldsymbol{\theta}$.
2: Train SVM met deze $\boldsymbol{\theta}$.
3: Bereken de sigmoid parameters A en B (Newton optimalisatiemethode).
4: Schat de kans op fouten met behulp van de validatie set.
5: Reken de gradiënt uit van die fout $\frac{\partial E(\boldsymbol{\alpha},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$.
6: Update $\boldsymbol{\theta}$ door gebruik te maken van Quasi-Newton update.
7: Hervat stap 2 of stop wanneer minimum bereikt wordt.

Net zoals in de afleiding van de *posteriore kansen* volgens [44], hercoderen we de labels tot $t_i = 1$ als de input vector $\boldsymbol{x}_i$ behoort tot klasse C_1 en $t_i = 0$ als het behoort tot klasse C_2. De kans dat er een fout wordt gemaakt op sample $\boldsymbol{x}_i$, kan men als volgt uitdrukken

$$E_i = P(y_i \neq z_i) = p_i^{1-t_i}(1-p_i)^{t_i}, \tag{3.22}$$

waarbij $z_i = \text{sgn}(f_i)$ en $f_i = f(\boldsymbol{x}_i)$ de output van het testen van het i-de sample uit de validatie set. Kortom, z_i geeft ons niets anders dan het resultaat van de eerder vermelde beslissingsfunctie. p_i is de posterior probabiliteit en drukt niets anders uit $P(class|input) = P(y = 1|\boldsymbol{x}_i)$ en deze is, zoals in Sectie 3.3.3 afgeleid wordt, gelijk aan

$$p_i = \frac{1}{1 + \exp(A^* f(\boldsymbol{x}_i) + B^*)}. \tag{3.23}$$

E_i drukt inderdaad uit wat we willen: voor een element uit klasse C_1 wordt $1 - p_i$ in rekening gebracht, wat niet anders is dan de kans dat element verkeerd geclassificeerd wordt. Voor element uit C_2 kijken we naar de kans p_i, ook dit is wat we willen vermits de kansen werden opgesteld vanuit standpunt dat element tot klasse C_1 werd beschouwd. In dit geval komt dit overeen met het gegeven dat sample i ten onrechte tot klasse C_1 gerekend wordt.

Wanneer we nu de volledige validatie set overlopen, drukt men de gemiddelde fout uit met behulp van de gekende expressie voor empirisch gemiddelde:

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i = \frac{1}{N}\sum_{i=1}^{N} p_i^{1-t_i}(1-p_i)^{t_i}, \tag{3.24}$$

waarbij N de grootte van de validatie set is.

Wanneer we de parameters $\boldsymbol{\theta}$ zo optimaal mogelijk willen kiezen, dienen we ervoor te zorgen dat de gradiënt van E naar $\boldsymbol{\theta}$ nul wordt. Met behulp van de kettingregel kunnen we dit als volgt opsplitsen

$$\frac{\partial E}{\partial \boldsymbol{\theta}} = \frac{\partial E}{\partial \boldsymbol{\alpha}} \frac{\partial \boldsymbol{\alpha}}{\partial \boldsymbol{\theta}}, \tag{3.25}$$

hierbij is $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$ de vector van de multipliers en k het aantal support vectors. De reden waarom enkel de support vectors in rekening worden gebracht is vrij eenvoudig in te zien, immers bij de andere vectoren zal α_i gelijk zijn aan 0 en zal de afgeleide sowieso 0 worden. De componenten van de $(1 \times k)$ vector $\frac{\partial E}{\partial \boldsymbol{\alpha}}$, kan men wederom met behulp van de kettingregel als volgt uitsmeren

$$\frac{\partial E}{\partial \alpha_j} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial E_i}{\partial p_i}\frac{\partial p_i}{\partial f_i}\frac{\partial f_i}{\partial \alpha_j}, \tag{3.26}$$

waarbij na eenvoudige uitwerking geldt dat

$$\begin{aligned}
\frac{\partial E_i}{\partial p_i} &= (1-t_i)p_i^{-t_i}(1-p_i)^{t_i} - p_i^{1-t_i}t_i(1-p_i)^{t_i-1} \\
\frac{\partial p_i}{\partial f_i} &= -p_i^2 \exp(Af_i + B)A \\
\frac{\partial f_i}{\partial \alpha_j} &= y_i K_{\boldsymbol{\theta}}(\boldsymbol{x}_j, \boldsymbol{x}_i).
\end{aligned}$$

Rest er ons enkel nog de afgeleide van de multipliers vector $\boldsymbol{\alpha}$ naar parameter vector $\boldsymbol{\theta}$ uit te rekenen. Hiervoor maken we opnieuw gebruik van (3.21) en breiden we wederom $\boldsymbol{\alpha}$ uit met extra component b.

Posteriore kansen in SVMs

Vorige twee methoden maken gebruik van parameters A en B die afkomstig zijn uit een model dat de posteriore kans uitdrukt van een bepaalde SVM. De vorm van het parametrisch model werd geïnspireerd door te kijken naar empirische data [44], waaruit men besloot dat de voorwaardelijke kans op een bepaalde output f gegeven een bepaalde klasse ($y = \pm 1$) exponentieel verdeeld lijkt.

Toepassing van de regel van Bayes suggereert vervolgens een vaak gebruikte verdeling binnen de Artificiële Intelligentie (AI), namelijk een sigmoide (of logistieke) verdeling.

$$\begin{aligned}
P(y=1|f) &= \frac{P(f|y=1)P(y=1)}{\sum_{i=-1,1} P(f|y=i)P(y=i)} \\
&= \frac{\lambda_1 e^{-\lambda_1 f} P(y=1)}{\sum_{i=-1,1} \lambda_i e^{-\lambda_i f} P(y=i)} \\
&= \Big(1 + \frac{\lambda_{-1} e^{-\lambda_{-1} f} P(y=-1)}{\lambda_1 e^{-\lambda_1 f} P(y=1)}\Big)^{-1} \\
&= \Big(1 + \frac{\lambda_{-1} P(y=-1)}{\lambda_1 P(y=1)} \exp -(\lambda_{-1} - \lambda_1) f\Big)^{-1} \\
&= \Big(1 + \exp\left[-(\lambda_{-1} - \lambda_1) f + \ln(\frac{\lambda_{-1} P(y=-1)}{\lambda_1 P(y=1)})\right]\Big)^{-1} \\
&= \Big(1 + \exp(Af + B)\Big)^{-1}
\end{aligned}$$

De parameters A en B worden gefit door een maximum likelihood schatting van de training set (f_i, t_i), waarbij

$$t_i = \frac{y_i + 1}{2}$$

en worden gevonden door de negatieve log likelihood van de trainingdata uit te rekenen. Wat op zijn beurt in feite niets anders is dan een cross-entropy functie[4]:

$$\min - \sum_i t_i \log(p_i) + (1 - t_i) \log(1 - p_i) \tag{3.27}$$

[4]Volledige hoe en waarom is na te lezen in [4, 43], maar kortweg meet deze expressie verschil tussen twee verdelingen. In ons geval een Bernouilli verdeling dat de labels toekent en een logistieke verdeling die de kans op een bepaald label uitdrukt gegeven een welbepaalde output. Als we deze uitdrukking minimaliseren, zeggen we dat we willen dat deze zich hetzelfde gedrag vertonen.

waarbij

$$p_i = \frac{1}{1 + \exp(Af_i + B)}.$$

Lemma 3.6. *In het geval van* hard-margin *of* least-squares *SVMs, bekomen we steeds een $A \leq 0$ als optimum van (3.27).*

Bewijs. Stel immers dat $A > 0$, dan kunnen we de waarde van de negatieve log likelihood (3.27) nog naar beneden halen door A gelijk aan 0 te stellen. We kunnen immers de objectieve functie op volgende manier noteren, waarbij i respectievelijk j loopt over de positieve, respectievelijk negatieve samples:

$$\begin{aligned}
&- \Big(\sum_i t_i \log(p_i) + (1 - t_i) \log(1 - p_i) + \sum_j t_j \log(p_j) + (1 - t_j) \log(1 - p_j) \Big) \\
&= -\Big(\sum_i t_i \log(p_i) + \sum_j (1 - t_j) \log(1 - p_j) \Big) \\
&= -\Big(\sum_i \log(p_i) + \sum_j \log(1 - p_j) \Big) \\
&\geq -\Big(\sum_i \log(p'_i) + \sum_j \log(1 - p'_j) \Big).
\end{aligned}$$

De ongelijkheid geldt omdat voor elk van de postieve objecten, geldt dat

$$\log(p'_i) > \log(p_i).$$

Immers,

$$\begin{aligned}
& \log \frac{1}{1 + \exp(B)} > \log \frac{1}{1 + \exp(Af_i + B)} \\
\iff & \frac{1}{1 + \exp(B)} > \frac{1}{1 + \exp(Af_i + B)} \\
\iff & 1 + \exp(B) < 1 + \exp(Af_i + B) \\
\iff & \exp(B) < \exp(Af_i) \exp(B) \\
\iff & 1 < \exp(Af_i) \\
\iff & 0 < Af_i \\
\iff & 0 < A,
\end{aligned}$$

waarbij we in de laatste stap gebruik maken van het feit dat we werken met *hard-margin* of *least-squares* SVMs. Met behulp van een analoge redenering kan men aantonen dat ook voor de negatieve objecten, geldt dat

$$\log(1 - p'_j) > \log(1 - p_i).$$

□

Lemma 3.6 toont ons dat na uitwerking van (3.27) $P(y = 1|f)$ weldegelijk een cumulatieve distributie heeft, en we geen kunstgreep moeten uithalen opdat de coëfficiënt A negatief zou zijn.

Originele pseudo-code en een lichtjes gewijzigde (stabielere) variant om dit minimalisatie probleem op te lossen, zijn na te lezen in [44] respectievelijk [39].

3.4 Modelselectie bij single-class classificatie

We keren in deze sectie terug op de one-class classificatie methoden uit Hoofdstuk 2 om de verschuldigde uitleg over de betekenis van parameter C uit de doeken te doen en waarna we enkele heuristieken beschrijven om de kernel parameter σ in de Gaussische RBF kernel af te stellen.

3.4.1 Target error estimate – betekenis parameter C

Zoals eerder vermeld, zorgt de parameter C voor een *trade-off* tussen het volume van de sfeer en het aantal objecten uit de target set dat als té ver beschouwd wordt. Toch willen we geen genoegen nemen met deze vage betekenis en willen we de lezer met een meer theoretische uitleg verblijden.

Wanneer een object van de target set verworpen wordt door de geconstrueerde omschrijving van de data, wordt dit een type II fout genoemd, ook wel $\mathcal{E}_{II}$ genoteerd, aangezien ten onrechte een element van de target set als outlier werd bestempeld (standaard terminologie: *false negative*).

Om een bovengrens voor deze fout in te voeren, combineren we een aantal besproken theoretische resultaten. Het belangrijkste werktuig vormt het verband tussen binaire classificatie en de data description methoden. Dit laat ons immers toe resultaten over de generalisatie fout uit Sectie 3.2 te hergebruiken in de context van single-class problemen. Voor de verklaring van parameter C zal echter *Support Vector Count* (Stelling 3.2), waarbij het aantal support vectors een aanwijzing is voor de verwachte target error, volstaan.

Intuïtief kan men deze stelling als volgt begrijpen. Wanneer we in onze data beschrijving een trainingsobject weglaten dat niet tot de verzameling van de support vectors behoort, dan zal er bij weglating niets veranderen en zal bij de test procedure dit object correct geclassificeerd worden als non-outlier. Verklaring hiervoor is dat bij de expansie (2.10) van het centrum van de sfeer in termen van de $\boldsymbol{x}_i$'s, $\alpha = 0$ geen bijdrage leverde.

Bij het verwijderen van een support vector, onderscheiden we 3 gevallen: *in-bound*, *bound* en *essential* support vectors. Het verschil hangt af van de waarde van bijbehorende α_i, zoals Tabel 3.1 aangeeft. Het is enkel wanneer

we een essential SV verwijderen dat dit een fout veroorzaakt bij de leave-one-out procedure, want enkel in dat geval zullen we een kleinere sfeer vinden en wordt dit object bij de test als outlier aanzien.

Tabel 3.1: Overgenomen uit [56] waarbij we hier α_{max} dienen te interpreteren als C, *on margin* als "op de rand van de sfeer" en *in margin* het gebied buiten de sfeer.

Type SV	Definition	Properties
(standard) SV	$0 < \alpha_i$	lies on or in margin
in-bound SV	$0 < \alpha_i < \alpha_{max}$	lies on margin
bound SV	$\alpha_i = \alpha_{max}$	usually lies in margin ("margin error")
essential SV	appears in all possible expansions of solution	becomes margin error when left out

De strikte ongelijkheid in Stelling 3.2 geldt wanneer niet alle support vectors essentieel zijn en dus uit de omschrijving kunnen weggelaten worden. Dit is bijvoorbeeld het geval wanneer we werken in een 2-dimensionale ruimte waarbij we slechts 3 punten[5] nodig hebben om een cirkel te beschrijven.

We voeren opnieuw de notatie $\nu = 1/(mC)$ omdat deze – zoals in een ogenblik duidelijk wordt – een meer natuurlijke betekenis heeft, namelijk de fractie aan outliers (deel van target data dat als type II fout bestempeld zal worden).

Stelling 3.7 (ν-property)**.** *Beschouwen we het* soft sphere probleem *(2.6) en (2.7) met $\nu = 1/(mC)$. Dan gelden volgende eigenschappen:*

1. *ν is een bovengrens voor het percentage outliers.*
2. *ν is een ondergrens voor het percentage SVs.*

Bewijs.

1. Stel N_E het aantal errors (i.e. $\alpha_i = C$) dan construeren we een boven-

[5]Soms volstaan zelfs 2 wanneer zowel centrum en de 2 punten collineair zijn.

grens op N_E gebruik makend van de nevenvoorwaarden (2.9) en (2.12).

$$\begin{aligned}
&\sum_i^m \alpha_i = 1 \\
&\Rightarrow N_E C + \sum_i^{m-N_E} \alpha_i = 1 \\
&\Rightarrow N_E = \frac{1 - \sum_i^{m-N_E} \alpha_i}{C} \\
&\Rightarrow N_E \leq \frac{1}{C} \\
&\Rightarrow \frac{N_E}{m} \leq \frac{1}{mC} = \nu \\
&\Rightarrow \text{fraction of errors} \leq \nu
\end{aligned}$$

2. Stel N_{SV} het aantal SVs (i.e. $\alpha_i > 0$), dan construeren we een ondergrens op N_{SV} gebruik makend van de nevenvoorwaarden (2.9) en (2.12).

$$\begin{aligned}
&\sum_i^m \alpha_i = 1 \\
&\Rightarrow \sum_i^{N_{SV}} \alpha_i = 1 \\
&\Rightarrow N_{SV} C \geq 1 \\
&\Rightarrow N_{SV} \geq \frac{1}{C} \\
&\Rightarrow \frac{N_{SV}}{m} \geq \frac{1}{Cm} = \nu \\
&\Rightarrow \text{fraction of SVs} \geq \nu
\end{aligned}$$

□

Hierdoor kunnen we het gevolg van de keuze van parameter C beter in schatten. Merk vooreerst op dat we $C \geq 1/m$ dienen te nemen. Anders staat in (2.9) een voorwaarde die nooit vervuld kan worden en dus nooit tot een oplossing van het probleem kan leiden. Anderzijds zullen we voor $C \geq 1$ steeds een oplossing vinden, waarbij we geen fouten toestaan; dit komt overeen met de *rigid sphere* of *hard margin* oplossing. Immers, als er zich dan een fout voor doet, $\alpha_i = C = 1$ dan kunnen we wegens voorwaarde (2.9) geen andere support vectors meer vinden en kunnen we dus ook geen beschrijving van een sfeer opstellen.

Samenvattend beperkt de keuze van de parameter C zich tot het interval $[\frac{1}{m}, 1]$ en dus varieert van uiterst tolerant (maximaal 100% outliers) tot een nultolerantie wat de outliers betreft.

Wanneer we bij negative SVDD ook samples toelaten die we als outlier wensen te beschouwen, kunnen ook type I fouten ($\mathcal{E}_I$) voorkomen. Dit verschijnsel zal optreden wanneer een outlier object ten onrechte als target data bestempeld wordt, men spreekt dan over een zogenaamde *false positive*. Op analoge manier zijn de betekenis van de parameters C_1 en C_2 te verklaren en hun keuze zal nu het percentage false negatives respectievelijk false positives bepalen.

3.4.2 Heuristiek keuze kernel parameter

Wanneer men op voorhand een schatting levert voor de type II fout die gemaakt mag worden op de training set, kan men de hyperparameter σ^2 met behulp van volgend iteratief schema [60] optimaliseren.

Bereken de initiële data beschrijving met σ^2 afgesteld op een kleine waarde (bijvoorbeeld kleinste afstand tussen 2 verschillende objecten). Zolang het percentage support vectors groter is dan de vooraf gedefinieerde bovengrens ν, blijven we de σ^2 verhogen en berekenen we telkens opnieuw het optimale model. Wanneer σ^2 groter wordt dan de grootste afstand tussen 2 objecten, staken we het iteratieve proces eveneens, want dan wordt sowieso toch het sferische model bekomen.

Een andere methode [31] gebruikt een gelijkaardig iteratief schema waarbij enkel het criterium om te stoppen verschilt. Voor kleine waarden van de gewichtsparameter σ^2 zijn alle trainingspunten support vectors, het algoritme memoriseert gewoon de data en zal niet goed generaliseren. Als we σ^2 verhogen, verlaagt het aantal SVs. Als eenvoudige heuristiek kunnen we dus starten met een kleine σ^2 en deze blijven verhogen totdat het aantal SVs niet verder afneemt.

Methoden om zowel de kernel parameters als de regularisatie parameter C te kiezen staan nog in de kinderschoenen. Het grootste probleem vormt het feit dat we enkel over de target set beschikken om onze data omschrijving te controleren. Omwille van deze reden is het moeilijk om het generaliserende karakter van de methode in te schatten.

In [63, 64] wordt dit probleem getackeld door een extra validatie set te gebruiken, bestaande uit een aantal voorbeelden representatief voor de target set en gaat men tevens ook een grote verzameling artificiële variabelen genereren, hetzij in een box hetzij in een sfeer, rondom de target set. Vervolgens gaat men de validatie error trachten te minimaliseren, bijvoorbeeld door te parameterruimte af te speuren met een gridsearch. Spijtig genoeg lijdt

deze methode direct onder de vloek die rust op hoge dimensionaliteit ("curse of dimensionality"): bij (lineaire) toename van de dimensie, stijgt het aantal benodigde samples om eenzelfde bedekking te hebben van de ruimte exponentieel.

Hoofdstuk 4

Feature selectie

Aangezien de rekenkracht van computers en de rand- en meetapparatuur steeds geavanceerder worden, is high throughput data-analysis geen wens maar een must. Om het hoofd te bieden aan data sets met duizenden of veeleer tienduizenden variabelen vormt, naast parameterselectie, feature selectie een niet te onderschatten onderdeel van modelselectie. Applicaties die staan te popelen om de vruchten te plukken van onderzoek op dit topic zijn ondermeer de automatische informatie extractie uit tekstdocumenten (afkomstig van het internet), analyse van genexpressies en alles wat te maken heeft bioinformatica (bijvoorbeeld de computationele tak binnen de chemische industrie).

Dit hoofdstuk ziet er als volgt uit. Na een korte inleiding om de meest fundamentele vragen omtrent feature selectie – wat, waarom en hoe? – te beantwoorden, nemen we een kijkje hoe support vector machines ons hierbij zouden kunnen helpen en welke voordelen ze er eventueel zelf uit kunnen halen.

4.1 Inleiding

Wanneer in 1997 een speciale editie uit het tijdschrift *Artificial Intelligence* gewijd werd aan het topic over variabele selectie [5, 37], gebruikten slechts enkele onderzoeksdomeinen meer dan 40 features. Deze situatie is echter de laatste jaren drastisch veranderd, de meeste papers verkennen domeinen waarin data sets met honderden tot tienduizenden variabelen eerder de regel dan de uitzondering vormen. Nieuwe technieken om taken met vele irrelevante, en dus overbodige, variabelen en met in vergelijking een kleine hoeveelheid aan trainingsvoorbeelden het hoofd te bieden, schieten als paddenstoelen uit de grond.

Typische voorbeelden van nieuwe toepassingen die snakken naar deze technieken zijn ondermeer genselectie van *micro-array data* [24, 26, 42], classificatie van tekstdocumenten en *peak selection* bij gas chromatografie en massa spectroscopie [21, 23, 69].

In het genselectie probleem corresponderen de input variabelen met de genexpressie coëfficiënten van mRNA[1] uit een sample van een aantal patiënten. Een standaard classificatie probleem vormt het scheiden van gezonde patiënten van kankerpatiënten op basis van hun gen profiel. Meestal zijn er slechts een honderdtal voorbeelden (patiënten) beschikbaar voor training- en testfase tezamen. Het aantal variabelen in de data varieert daarentegen echter tussen 6000 en 60 000.

In het tekstclassificatie probleem worden documenten veelal voorgesteld door een *bag-of-words*, een vector waarvan de dimensie gelijk is aan het aantal verschillende woorden uit de trainingcorpora en waarin de frequentie van voorkomen wordt vermeld. Woordenschatten van enkele honderdduizenden woorden zijn hierbij schering en inslag. Typische taken die men achter af met deze data uithaalt is het automatisch sorteren van URLs in een web directory of detectie van spam/ham op mailservers en/of mailclients.

Naast classificatie van stoffen aan de hand van data afkomstig van gas chromatografie en massa spectroscopie, zou een andere mogelijke toepassing er uit kunnen bestaan de effectieve verschillen tussen samples uit verschillende batchen te melden. Aangezien de herkenning van pieken hierbij een cruciale rol speelt, zal ook hier een reductie aan gegevens het werk van chemici kunnen besparen en wie weet wel kunnen automatiseren.

4.1.1 Wat?

Ondanks het verschil in terminologie *feature* en *variabele* [28], waarbij features uit de input variabelen worden bekomen door één of andere preprocessing stap toe te passen, en we bovendien in het kader van kernel methoden een impliciete mapping uitvoeren van input naar feature ruimte, zullen we in het vervolg van dit hoofdstuk geen onderscheid maken tussen beide begrippen en plaatsen we alles onder de noemer *feature selectie*. Immers, de features worden steeds met behulp van een kernel berekend op basis van de input variabelen en vormen dus geen tastbaar gegeven. Daarom concentreren de meeste methoden zich op selectie van de input variabelen en slechts in uitzonderlijke gevallen (zoals bijvoorbeeld in [72]) op de reductie van expliciete features.

[1] [75] Messenger RNA, dat meestal mRNA genoemd wordt, speelt een centrale rol in het tot expressie brengen van genetische informatie. mRNA is een vorm van RNA, welke als 'boodschapper' (messenger) twee processen met elkaar verbindt: de transcriptie, waarbij een stuk DNA (een gen) overgeschreven wordt tot mRNA, en de translatie, waarbij het mRNA wordt vertaald naar een keten van aminozuren (een eiwit).

Het feature selectie probleem kunnen we op volgende twee manieren opvatten:

- gegeven een vaste $k \ll n$, vind de k features die aanleiding geven tot de kleinste generalisatie fout; of
- gegeven een maximale generalisatie fout, vind de kleinste k.

Aangezien in beide formuleringen de generalisatie fout vanzelfsprekend onbekend is, zullen we deze moeten schatten bijvoorbeeld, gebruikmakende van de besproken criteria uit het vorige hoofdstuk.

Vooraleer we een kort overzicht verschaffen van de algemene methoden waarmee we dit combinatorische en, zoals aangetoond in [1], NP-complete probleem aanvatten, geven we eerst aan waarom we in de eerste plaats geïnteresseerd zijn om het feature selectie probleem in de praktijk op te lossen.

4.1.2 Waarom?

Het waarom kunnen we samenvatten door volgende drie doelstellingen te formuleren: het verbeteren van de voorspellingsperformantie van de predictor variabelen, het vinden van snellere en kostefficiëntere predictoren, en het bevorderen van het begrip over het onderliggende proces dat de data genereert.

Immers, als we in staat zijn ons te beperken tot juist die verzameling aan features die het probleem karakteriseren, zal dit de generalisatie fout verminderen. We zijn dan immers bezig met het modelleren van het kernidee en vergeten alle mogelijke ruisfactoren. Desondanks we weten dat SVMs uitstekend kunnen omgaan met een hoge dimensionaliteit van de data, zijn ook deze tools onderhevig aan de gevolgen van de problemen die zich afspelen in ruimten waarin vele features irrelevant zijn. In Sectie 5.2.1 wordt dit geïllustreerd aan de hand van twee artificieel gegeneerde problemen.

Met een kleiner aantal input variabelen neemt ook de benodigde tijd en plaats van het model af, waardoor ze ingeschakeld kunnen worden in applicaties onderhevig aan strikte tijd of geheugen beperkingen. We kunnen dan een aantal software of hardware sensors uitschakelen en de uit te voeren berekeningen tot een minimum herleiden.

Als we bovendien op zoek zijn naar verklaringen, bijvoorbeeld om geschikte medicatie te ontwikkelen in farmacogenetica (*pharmacogenomics*) [22, 48], is het nodig om tot het juiste selecte clubje van genen door te dringen. Wanneer daarenboven we achteraf eveneens zaken willen opsteken van de classifier, brengt een subset van een tiental variabelen ons mogelijk sneller tot inzicht dan wanneer we steeds moeten kijken naar de waarden van honderden, zo niet duizenden, verschillende metingen.

4.1.3 Hoe?

We schetsen nu een kort overzicht van enkele intrinsiek van elkaar verschillende aanpakken samen met hun mogelijke voor- en nadelen. Het spreekt voor zich dat deze lijst onvolledig is en de nodige diepgang ontbreekt; ze dient enkel om de nadien besproken methoden, SVM-RFE en scaled feature selection, in een breder kader te plaatsen.

Feature ranking versus subset selectie

Vele feature selectie algoritmen hebben op één of andere manier het rangschikken van variabelen als hoofd- of hulpmechanisme omwille van de eenvoud, schaalbaarheid en goede empirische resultaten.

Tot deze taxonomie horen ondermeer de verschillende (lineaire) feature ranking criteria op basis van correlatie coëfficiënten, zoals daar bijvoorbeeld zijn de *Pearson correlatie coëfficiënt*

$$R(i) = \frac{\mathrm{cov}(X_i, Y)}{\sqrt{\mathrm{var}(X_i)\mathrm{var}(Y)}}$$

of de *Fisher score*,

$$F(i) = \left| \frac{\mu_i^+ - \mu_i^-}{\sigma_i^{+2} - \sigma_i^{-2}} \right|$$

waar $\mu_i^{\pm}$ de gemiddelde waarde is van het i-de feature in de positieve/negatieve klasse en $\sigma_i^{\pm}$ de standaard afwijking, die de verhouding uitdrukt tussen de variantie tussen de klassen onderling en de variantie binnen eenzelfde klasse.

Het nadeel van feature selectie gebaseerd op ranking van afzonderlijke features, is dat ze enkel geëvalueerd worden ten opzichte van een welbepaalde responsvariabele, waardoor we in de uiteindelijke subset enkel die variabelen opnemen die weliswaar afzonderlijk behoorlijk presteren, maar eventueel nog reduntanties vertonen en we mogelijk hierdoor complementaire features uit het oog verliezen die samen tot betere resultaten zouden leiden.

In [28] wordt aan de hand van een drietal eenvoudige voorbeelden uit de doeken gedaan waarom

1. *noise reduction and consequently better class separation may be obtained by adding variables that are presumably redundant (Figuur A.1);*

2. *perfectly correlated variables are truly redundant in the sense that no additional information is gained by adding them, but very high variable correlation (or anti-correlation) does not mean absence of variable complementarity (Figuur A.2);*

3. *a variable that is completely useless by itself can provide a significant performance improvement when taken with others or two variables that are useless by themselves can be useful together (Figuur A.3).*

Dit laatste puntje toont aan waarom het beschouwen van deelverzamelingen van variabelen die samen een goede voorspellingskracht hebben, in tegenstelling tot het rangschikken van variabelen volgens hun individuele sterkte, vaak de voorkeur geniet.

Echter het beschouwen van alle mogelijke deelverzamelingen is zoals we weten een combinatorisch probleem, aangezien het aantal mogelijke deelverzamelingen gelijk is aan het aantal elementen van de machtsverzameling[2] en dus gelijk is aan 2^n.

In de literatuur [5, 37] maakt men onderscheid tussen drie types van methoden om het probleem op te lossen: de zogenaamde *filter*, *wrapper* en *embedded* methoden.

Filter methoden

Filter methoden worden gedefinieerd als een preprocessing stap met als taak de irrelevante attributen te verwijderen alvorens we de data doorgeven aan een inductie methode. Ze trachten hierbij te streven naar een compacte representatie, onafhankelijk van de later te gebruiken methode om het model op te stellen.

Het ultieme voordeel is dat zij vaak zeer eenvoudig te implementeren en daarenboven snel te evalueren zijn. Zij zijn echter suboptimaal[3] aangezien zij geen rekening houden met de uiteindelijk gebruikte leermethode.

Als voorbeeld van een filter subset selectie procedure gebaseerd op correlatie coëfficiënten – vandaar de naam – die vaak in de wereld van machine learning wordt gebruikt, vermelden we *correlation-based feature selection* (CFS). Deze methode, uitvoerig beschreven in [30], berust op het idee om iteratief predictorvariabelen te selecteren die enerzijds een hoge correlatie hebben met de responsvariabelen en die anderzijds eerder ongecorreleerd zijn met de reeds geselecteerde variabelen. Intuïtief kunnen we dit eenvoudig inzien: aangezien correlatie een maat is voor empirische afhankelijkheid, selecteren we dus juist die relaties die zo goed mogelijk de output trachten te verklaren en anderzijds proberen we geen variabelen toe te voegen die ons geen extra inzicht bieden.

[2] Wegens het binomium van Newton geldt, $\sum_k \binom{n}{k} = (1+1)^n = 2^n$.

[3] In sommige omstandigheden zijn ze optimaal, bijvoorbeeld als we Fisher score gebruiken om de variabelen te ordenen alvorens we deze aan een lineaire discrimant analyse (LDA) routine doorgeven [74].

Wrapper methoden

Wrapper methoden beschouwen de gebruikte machine als een perfecte *black box* en gebruiken de voorspellingsperformantie ervan om de relatieve bruikbaarheid van subsets van variabelen te bepalen. In de praktijk, moet men het volgende specifiëren:

1. hoe de ruimte van alle mogelijke deelverzamelingen te doorzoeken;
2. hoe de performantie van het model te bepalen om de zoektocht te sturen en te stoppen;
3. welke predictor te gebruiken.

Het meten van de performantie wordt, net zoals in het vorige hoofdstuk over het optimaliseren van de hyperparameters, meestal gedaan door gebruik te maken van een onafhankelijke validatie set of door gebruik te maken van cross-validatie. Als predictor kan men elke mogelijke leertechniek gebruiken zoals bijvoorbeeld beslissingsbomen, naïve (ofwel *idiot*) Bayes [76], of zoals wij zullen doen, support vector machines.

De zoektocht wordt al snel computationeel onaantrekkelijk, het aantal deelverzamelingen groeit immers exponentieel met het aantal features. Vandaar dat men buiten *exhaustive search* vaak andere zoekstrategieën gebruikt zoals daar zijn *best-first*, *branch-and-bound*, *simulated annealing* of genetische algoritmen. Een overzicht wordt verschaft in de spraakmakende paper over de wrapper methodologie [37].

Om het nadeel van deze methode, dat er vinger dik opligt namelijk het verslinden van computer resources, nog verder te reduceren, stapt men vaak over naar *forward selection* of *backward elimination*, zogenaamde *greedy* zoekstrategieën; de "hebzucht" in deze context verwijst naar het feit dat er geen *back-tracking* mechanisme is ingebouwd en er dus geen mogelijkheid is om op eerder gemaakte beslissingen terug te komen. Beide methoden bouwen iteratief geneste feature verzamelingen $F_1 \subset F_2 \subset \ldots \subset F_n$ op, waarbij we in forward selection telkens de meest belovende variabele toevoegen, terwijl we bij backward elimination de variabele met de minste invloed uit de voorgaande verzameling verwijderen.

Embedded methoden

Embedded methoden vormen een compromis tussen de filter en wrapper methoden door het beste van beide werelden te combineren. Ze voeren de variabele selectie uit als onderdeel van het trainingsprocess en zijn specifiek voor bepaalde leermachines. Ultieme voorbeelden zijn expert systemen

gebaseerd op tree inductie, zoals bijvoorbeeld ID3, C4.5 of CART (Classification and Regression Trees, [7]) die één of andere vorm van onzuiverheidscriterium gebruiken, zoals bijvoorbeeld entropie waarden, Gini index of informatie winst, om de volgende split tijdens het opbouwen van de boom te bepalen.

Eventueel zal de groeifase worden gevolgd door een snoeifase (het *prunen* van de boom), waarbij we wel een vorm van validatie zullen moeten gebruiken (hetzij in de vorm van een aparte validatie set, hetzij door toepassing van k-fold cross validatie).

De feature selectie zit dus intrinsiek in het leerproces ingebakken, waarbij men er van uitgaat dat enkel die variabelen overblijven in de uiteindelijke boom of, meer algemeen, het model die het belangrijkst zijn en het probleem het best omschrijven.

Er zijn nog twee speficieke types embedded methoden die onze extra aandacht verdienen, *nested subset* methoden en *direct objective optimization*, aangezien de specifieke algoritmen in combinatie met support vector machines besproken in dit eindwerk onder deze noemers vallen.

Nested subset methoden laten zich tijdens de zoektocht doorheen de ruimte van feature verzamelingen leiden door een schatting van de wijziging van de waarde van de objectieve functie. In combinatie met greedy zoekstrategieën (backward elimination of forward selection) bekomen ze geneste deelverzamelingen van variabelen.

Deze methoden verschillen onderling in elkaar op de manier waarop ze de verandering in de waarde van de objectieve functie $J(k)$, waarbij k duidt op het aantal variabelen, voorspellen:

1. *Finite difference calculation*: Het verschil tussen $J(k)$ en $J(k+1)$ of $J(k-1)$ wordt berekend voor de variabelen die we willen toevoegen of verwijderen.

2. *Optimum brain damage (OBD)*: werd oorspronkelijk geformuleerd om de gewichten w_i in neurale netwerken te *prunen* [38], maar kan eveneens gebruikt worden bij backward elimination om een tweede orde benadering van J te definieren die het verschil in functie waarde ΔJ_i berekent bij verwijdering van het i-de feature

$$\Delta J_i = \frac{\partial J}{\partial w_i}\Delta w_i + \frac{1}{2}\frac{\partial^2 J}{\partial^2 w_i}(\Delta w_i)^2,$$

   waarbij het verschil in gewicht $\Delta w_i = w_i$ overeenkomt met het verwijderen van het i-de feature. In het optimum van J kunnen we de eerste

orde term verwaarlozen, dit resulteert uiteindelijk in

$$\Delta J_i = \frac{1}{2}\frac{\partial^2 J}{\partial^2 w_i}(\Delta w_i)^2.$$

3. *Sensitivity of the objective function calculation*: Hierbij wordt de absolute waarde of het kwadraat van de afgeleide van J naar x_i (of w_i) gebruikt.

Kortom ze verschillen onderling door het feit of ze een nulde, eerste of tweede orde benadering gebruiken als maat voor de verandering in objectieve functie.

Direct objective optimization houdt in dat men effectief de objectieve functie van het variabele selectie probleem formeel tracht uit te drukken en algoritmen te vinden om deze te optimaliseren. In het algemeen, bestaat de objectieve functie uit twee termen die elkaar bestrijden: enerzijds het deel dat *goodness-of-fit* uit drukt en dat we willen maximaliseren en anderzijds het aantal variabelen, waarvan we er zo min mogelijk willen gebruiken. Zonder de details uit te werken, die overigens te vinden zijn in [72], past dit perfect in het kader van het support vector framework, waarbij we nu niet de l_1- of l_2-norm gaan gebruiken, maar de nul-norm van $\boldsymbol{w}$

$$||\boldsymbol{w}||_0^0 = \text{card}\{w_i | w_i \neq 0\}.$$

Dit probleem wordt benaderend opgelost en valt uiteindelijk samen met de SVM-RFE methode, besproken in volgende sectie.

We naderen stilaan het hoogtepunt van dit hoofdstuk. In volgende twee secties bespreken we twee embedded algoritmen die gebruik maken van specifieke eigenschappen van support vector machines om geneste subsets van features te construeren.

4.2 SVM-RFE

Als eerste bespreken we *Support Vector Machine Recursive Feature Elimination* (SVM-RFE) [29], waar de geneste deelverzamelingen van variabelen sequentieel worden geselecteerd door achterwaartse eliminatie toe te passen, startende met alle features en telkens feature per feature te verwijderen. Op die manier zullen uiteindelijk alle features gerangschikt worden.

In het geval van lineaire support vector machines, worden in elke stap de coëfficiënten van de gewichtsvector $\boldsymbol{w}$ gebruikt als feature ranking criterium. Dit geeft aanleiding tot de routine beschreven in Algoritme 8.

Algoritme 8 Recursieve feature eliminatie procedure voor lineaire SVMs

start: gerangschikte features $R = [\,]$; geselecteerde subset $S = [1, \ldots, n]$;
repeat
  train een lineaire SVM met alle trainingdata en variabelen in S;
  bereken de gewichtsvector $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x}_i$;
  bereken de ranking score van de features in S: $c_i = w_i^2$;
  vind het feature met de kleinste ranking score: $e = \min_i c_i$;
  update R: $R = R[e; R]$;
  update S: $S = S - [e]$;
until alle features gerangschikt zijn
output: gerangschikte lijst met features R.

Wanneer snelheid primeert, kunnen we dit algoritme aanpassen door meer dan één feature per stap te verwijderen. Let wel op dat, door bijvoorbeeld telkens de helft te elimineren, de classificatie performantie mogelijkerwijze achteruit kan gaan, aangezien we dan immers een minder fijne zoektocht uitvoeren.

We gebruiken w_i^2 als ranking score aangezien dit overeenstemt met het verwijderen van de variabele die de objectieve functie, in geval van support vector machines

$$J = \frac{1}{2} ||\boldsymbol{w}||^2 ,$$

het minst wijzigt. We kunnen het gebruik ervan rechtvaardigen door gebruik te maken van de optimal brain damage (OBD) benadering. Vermits

$$\Delta J_i = \frac{1}{2} \frac{\partial^2 J}{\partial^2 w_i} (\Delta w_i)^2 = \frac{1}{2} (\Delta w_i)^2 = \frac{1}{2} w_i^2,$$

waarbij de voorfactor 1/2 natuurlijk geen rol van betekenis speelt.

Wanneer we dit algoritme willen gebruiken om feature selectie uit te voeren voor een willekeurige kernel functie in plaats van de lineaire kernel, zullen we een ander ranking criterium moeten hanteren. Immers, de gewichtsvector $\boldsymbol{w}$ wordt in het algemeen gekarakteriseerd door

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \Phi(\boldsymbol{x}_i)$$

en ligt dus in de feature ruimte en kent, voor zover bekend, geen representant in de input ruimte. Volgens [72] kunnen we dit algoritme voor bepaalde kernels wel gebruiken om expliciete feature selectie in feature space uit te voeren, maar dan wel enkel voor polynomiale kernels en zeker niet voor de oneindig dimensionale mapping gekarakteriseerd door een Gaussische kernel, om zo het opgestelde SVM-model te vereenvoudigen.

Aangezien we eigenlijk meer geïnteresseerd zijn in variabele selectie in de input ruimte, gaan we op zoek naar die welbepaalde feature die de objectieve functie zo min mogelijk wijzigt. Wanneer we dit met de *finite difference calculation* uitdrukken, levert dit ons ranking criterium c_i voor het i-de attribuut

$$\begin{aligned} c_i &= \frac{1}{2}\Big| \, ||\boldsymbol{w}||^2 - \Big|\Big|\boldsymbol{w}^{(i)}\Big|\Big|^2 \, \Big| \\ &= \frac{1}{2}\Big| \sum_{k,l} \alpha_k^* \alpha_l^* y_k y_l K(\boldsymbol{x}_k, \boldsymbol{x}_l) - \sum_{k,l} \alpha_k^{*(i)} \alpha_l^{*(i)} y_k y_l K^{(i)}(\boldsymbol{x}_k, \boldsymbol{x}_l) \Big|, \end{aligned}$$

waarbij $K^{(i)}(\boldsymbol{x}_k, \boldsymbol{x}_l)$ bekomen wordt door juist het i-de attribuut weg te laten bij de berekening van de Gram matrix. Aangezien uit experimentele resultaten in [45] blijkt dat het opnieuw trainen van een SVM om $\boldsymbol{\alpha}^{*(i)}$ te bekomen nauwelijks verbeteringen inhoudt, bespaart men zich de moeite om dit verschil exact te bepalen en berekent met het ranking criterium c_i vaak gewoon als

$$\begin{aligned} c_i &= \frac{1}{2}\Big| \sum_{k,l} \alpha_k^* \alpha_l^* y_k y_l K(\boldsymbol{x}_k, \boldsymbol{x}_l) - \sum_{k,l} \alpha_k^* \alpha_l^* y_k y_l K^{(i)}(\boldsymbol{x}_k, \boldsymbol{x}_l) \Big| \\ &= \frac{1}{2}\Big| \sum_{k,l} \alpha_k^* \alpha_l^* y_k y_l \big(K(\boldsymbol{x}_k, \boldsymbol{x}_l) - K^{(i)}(\boldsymbol{x}_k, \boldsymbol{x}_l) \big) \Big|. \end{aligned}$$

Een andere mogelijkheid [45] bestaat eruit een gevoeligheidsanalyse van de objectieve functie voor een welbepaalde variabele uit te voeren. Hiervoor introduceren we een virtuele schalingsfactor $\boldsymbol{\nu}$ en berekenen dan de gradiënt van de objectieve functie naar deze schalingsfactor. Deze laatste gedraagt zich als een componentsgewijze vermenigvuldiging (met waarde $\mathbf{1}$) en $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ herleidt zich dan tot

$$K(\boldsymbol{\nu} * \boldsymbol{x}_i, \boldsymbol{\nu} * \boldsymbol{x}_j),$$

waarbij $*$ het componentsgewijze product uitdrukt.

De eerste orde ranking score, *weight vector gradient*, wordt dan gegeven door

$$\begin{aligned} c_i &= \Big| \frac{\partial \, \big|\big| \frac{1}{2}\boldsymbol{w} \big|\big|^2}{\partial \nu_i} \Big| \\ &= \frac{1}{2}\Big| \sum_{k,l} \alpha_k^* \alpha_l^* y_k y_l \frac{\partial K(\boldsymbol{\nu} * \boldsymbol{x}_k, \boldsymbol{\nu} * \boldsymbol{x}_l)}{\partial \nu_i} \Big|. \end{aligned} \tag{4.1}$$

In het geval van de Gaussische kernel, herleidt de partiële afgeleide zich onder de voorwaarde $\nu_i = 1$ tot

$$\frac{\partial K(\boldsymbol{\nu} * \boldsymbol{x}_k, \boldsymbol{\nu} * \boldsymbol{x}_l)}{\partial \nu_i} = -\frac{1}{\sigma^2}(\nu_i \boldsymbol{x}_{ki} - \nu_i \boldsymbol{x}_{li})^2 K(\boldsymbol{x}_k, \boldsymbol{x}_l) = -\frac{1}{\sigma^2}(\boldsymbol{x}_{ki} - \boldsymbol{x}_{li})^2 K(\boldsymbol{x}_k, \boldsymbol{x}_l)$$

en in het geval van de linaire kernel tot

$$\frac{\partial K(\boldsymbol{\nu} * \boldsymbol{x}_k, \boldsymbol{\nu} * \boldsymbol{x}_l)}{\partial \nu_i} = 2\nu_i \boldsymbol{x}_{ki} \boldsymbol{x}_{li} = 2\boldsymbol{x}_{ki} \boldsymbol{x}_{li}.$$

Wanneer we deze laatste uitdrukking substitueren in (4.1), kunnen we opmerken dat we terug uitkomen in ons basis geval, waardoor we een tweede verklaring vinden waarom w_i^2 weldegelijk een goede ordening oplevert. Immers[4],

$$c_i = \Big| \sum_{k,l} \alpha_k^* \alpha_l^* y_k y_l \boldsymbol{x}_{ki} \boldsymbol{x}_{li} \Big| = |w_i^2| = w_i^2.$$

We kunnen nu reeds de link maken met de methode uit volgende sectie, in het lineaire geval geeft SVM-RFE immers hetzelfde resultaat als feature scaling met behulp van gradient descent, waarin we ons telkens beperken tot het nemen van slechts één enkele stap.

4.3 Feature scaling methode

De virtuele scale parameters uit het laatste deel van vorige sectie, doen een belletje rinkelen bij de aandachtige lezer. Zoals beloofd in het hoofdstuk over het optimaliseren van de kernel parameters, kunnen we de schaalfactoren gebruiken als maat voor gewicht van de belangrijkheid van de input variabelen. De weg ligt dus open om de technieken uit Sectie 3.3 te gaan gebruiken om feature selectie uit te voeren.

De methode, die we kunnen classifiëren onder het mom van *direct objective optimization*, ziet er uit als volgt. Definieer kernel

$$K_{\boldsymbol{\sigma}}(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{\sigma} * \boldsymbol{x}, \boldsymbol{\sigma} * \boldsymbol{y}),$$

waar $*$ de componentsgewijze vermenigvuldiging is en $\boldsymbol{\sigma} \in \{0,1\}^n$ een binaire vector. Als σ_i gelijk is aan 1, betekent dit dat we de i-de component in rekening brengen en we dus de i-de input variabele behouden.

Om nu de k beste features te selecteren, gebruiken we als *goodness-of-fit* functie één van de schattingen van generalisatie fout uit Sectie 3.2, bijvoorbeeld de radius/margin bound. Het feature selection probleem kunnen we dan als volgend optimalisatieprobleem herformuleren

$$\arg\min_{\boldsymbol{\sigma}} \Big\{ R^2 W^2(\boldsymbol{\sigma}) \mid \sum_i \sigma_i = k \Big\}.$$

Om het minimum vinden van R^2W^2 over $\boldsymbol{\sigma}$, moeten we alle mogelijke deelverzamelingen van features doorzoeken waarvan we weten dat dit een combinatorisch probleem is en dus computationeel onaantrekkelijk. Om dit

[4] $\sum_i w_i^2 = ||\boldsymbol{w}||^2 = \sum_{k,l} \alpha_k^* \alpha_l^* y_k y_l \sum_i \boldsymbol{x}_{ki} \boldsymbol{x}_{li} = \sum_i \sum_{k,l} \alpha_k^* \alpha_l^* y_k y_l \boldsymbol{x}_{ki} \boldsymbol{x}_{li}$

probleem enigszins in te dijken, kunnen we onze zoektocht beperken tot het sequentieel toevoegen of verwijderen van variabelen.

Als alternatief kan men ook de techniek, beschreven in [12, 73], gebruiken waar we de binaire vector $\boldsymbol{\sigma} \in \{0,1\}^n$ benaderen door een vector met reële waarden $\boldsymbol{\sigma} \in [0,1]^n \subset \mathbb{R}^n$. Om dan de optimale waarde van $\boldsymbol{\sigma}$ te vinden, minimaliseert men de radius/margin bound, of menig andere differentieerbare criteria, door een standaard optimalisatie techniek, zoals bijvoorbeeld gradient descent, toe te passen om volgend benaderend integer programming probleem op te lossen:

$$\begin{aligned} &\underset{\boldsymbol{\sigma}\in\mathbb{R}^n}{\text{minimaliseer}} \quad R^2W^2(\boldsymbol{\sigma}) + \lambda \sum_i (\sigma_i)^p \\ &\text{met} \sum_i \sigma_i = k, \quad \sigma_i \geq 0, \qquad i = 1, \ldots, m. \end{aligned}$$

Wanneer λ groot genoeg gekozen wordt en $p \to 0$ zullen slechts k elementen groter zijn dan 0 en dus een oplossing van het feature selectie probleem benaderen.

Om de berekeningen nog te vereenvoudigen, kan men dit idee transformeren naar een recursieve eliminatie procedure en volgend feature ranking algoritme gebruiken:

Algoritme 9 Scaled feature selectie algoritme

1: Initialiseer $\boldsymbol{\sigma} = (1, \ldots, 1)$.
2: Los het SVM optimalisatieprobleem op.
3: Minimaliseer de schatting van de fout T, in functie van $\boldsymbol{\sigma}$ met een gradient stap.
4: Zolang we geen lokaal minimum van T bereiken, herhaal stappen 2 en 3.
5: Negeer de dimensies die overeenstemmen met kleine waarden σ_i door deze op 0 te zetten en ga naar stap 2.

We kunnen hetzelfde algoritme, mits een kleine aanpassing door te voeren, gebruiken om feature selectie uit te voeren in de ruimte van de principale componenten [14]. In plaats van de input vectoren, herschalen we nu de principale componenten door volgende kernel,

$$K_{\boldsymbol{\sigma}}(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{\sigma} * \boldsymbol{\Sigma}\boldsymbol{x}, \boldsymbol{\sigma} * \boldsymbol{\Sigma}\boldsymbol{y}),$$

te gebruiken, waarbij $\boldsymbol{\Sigma}$ de matrix is met principale componenten die we berekenen met het standaard principal component analysis (PCA) algoritme alvorens we de SVM trainen.

Het ultieme voordeel van deze methode gebaseerd op feature scaling is dat we door toepassing van de *ridge truc*, i.e. door parameter C (of ν) op te

tellen bij de diagonaal van de Gram matrix, we de optimale keuze ervan kunnen bepalen samen met de uitvoering om de optimale feature set te bepalen. Wanneer we combinaties van verschillende kernels toelaten, zoals in [58], zouden we ook de parameter voor de lineaire interpolatie telkens – zonder enige moeite – mee kunnen optimaliseren. Daar tegenover staat dat men bij SVM-RFE in beide gevallen de optimale parameters met een aparte routine moet bepalen, maar minder – evenveel als het aantal features – SVM modellen moet trainen tijdens de recursieve eliminatie procedure.

Hoofdstuk 5

Experimentele resultaten

We sluiten af met een omvangrijk gedeelte dat de theorie toetst aan de praktijk.

5.1 Selectie van (hyper)parameters

In dit eerste deel kijken we hoe de methode om de modelparameters te bepalen, besproken in Sectie 3.3.2, i.e. door de radius/margin bound te minimaliseren met behulp van een gradiënt-gebaseerde optimalisatie routine, het ervan afbrengt in vergelijking met een exhaustive grid search in combinatie met k-fold cross-validatie als criterium om de generalisatie eigenschappen te voorspellen.

5.1.1 Experimenten

Om het experiment te voltrekken, maakte ik gebruik van een aangepaste versie van LIBSVM [10], waaraan de berekening van de gradiënt van de Gaussian RBF kernel werd toegevoegd, en een bijpassende tool om de radius/margin bound te berekenen. In navolging van [17, 35], gebruikte ik de naar Python geporteerde variant van L-BFGS-B [8, 79], een quasi-Newton methode, opgenomen in het, door Prof. Dr. Chih-Jen Lin per e-mail aangeleverde, Python script om de scaled feature selectie, besproken in Sectie 4.3, te voltooien. Aan dit script voegde ik de aangepaste radius/margin bound uit Sectie 3.2.2 toe. Dit alles werd aan elkaar gelast door een, eveens in Python geschreven, driver routine dat aansluitend eveneens de analyse van de experimenten voor zijn rekening nam.

Deze testopstelling werd gevoed met benchmark data, beschikbaar gesteld op `http://ida.first.fhg.de/projects/bench/benchmarks.htm`, waarbij ik mij beperkte tot de 10 data sets met aanvaardbare grote (< 70 MB) om

binnen aanzienlijke tijd de gewenste resultaten te verkrijgen. Elk van de data sets bestaat uit een collectie van 100 realisaties, met uitzondering van *image* en *splice* waar slechts 20 realisaties voorhanden zijn, opgesplist in trainingsvoorbeelden en testobjecten van diverse grootte. Een overzicht van de verscheidenheid wordt verschaft in Tabel 5.1.

data set	train grootte	test grootte	input dimensie	realisaties
banana	400	4900	2	100
breast-cancer	200	77	9	100
diabetis	468	300	8	100
flare-solar	666	400	9	100
german	700	300	20	100
heart	170	100	13	100
image	1300	1010	18	20
splice	1000	2175	60	20
thyroid	140	75	5	100
titanic	150	2051	3	100

Tabel 5.1: Informatie benchmark data.

In dit experiment (zie Tabel 5.2) speelden we de keuze in norm tussen de verschillende modellen uit (L1-SVM versus L2-SVM), zowel voor de standaard Gaussische RBF kernel als de gewogen variant, waar we de keuze tussen één globale spreidingsparameter σ of één schalingsparameter per attribuut doorlichten. Hierbij werd nauwlettend het gemiddelde aantal SVM trainingen, een groffe maat om de duur van de parameterselectie te taxeren, alsook de gemiddelde error rate over het aantal realisaties per data set in de gaten gehouden. We vergeleken de bekomen resultaten met de baseline uit [46], waar de modelparameter C en de spreidingsparameter σ geselecteerd werden uit een parameterruimte met elks 10 elementen met behulp van een exhaustive grid search in combinatie met 5-fold crossvalidatie.

	norm	kernel	methode	criterium
A	L1-SVM	Gaussian RBF	gridsearch	5-fold CV
B	L1-SVM	Gaussian RBF	quasi-Newton	R^2W^2
C	L2-SVM	Gaussian RBF	quasi-Newton	R^2W^2
D	L1-SVM	weighted Gaussian RBF	quasi-Newton	R^2W^2
E	L2-SVM	weighted Gaussian RBF	quasi-Newton	R^2W^2

Tabel 5.2: Legende experiment.

5.1.2 Resultaten

De verkregen resultaten worden samengevat in Tabel 5.3 en Tabel 5.4, terwijl dezelfde resultaten aangevuld met informatie over de standaardafwijking bijgesloten werden in Appendix B.

data set	A	B	C	D	E
banana	500	9.73	8.38	12.78	10.57
breast-cancer	500	10.43	11.80	11.66	19.20
diabetis	500	14.92	9.84	16.05	18.64
flare-solar	500	9.80	8.36	14.81	10.17
german	500	13.81	11.05	33.45	28.48
heart	500	14.14	11.00	21.56	24.61
image	500	14.05	13.45	18.20	18.95
splice	500	13.45	19.40	41.25	14.80
thyroid	500	15.08	11.25	23.37	18.74
titanic	500	8.06	6.76	10.11	7.23
gemiddeld	500	12.32	11.13	20.32	17.14

Tabel 5.3: Gemiddeld aantal SVM trainingen.

Wanneer we Tabel 5.3 bekijken, merken we een spectaculaire tijdsbesparing, uitgedrukt in het gemiddeld aantal benodigde SVM trainingen om tot de optimale parameters te komen. Voor de grid search met 5-fold CV is deze constant en, aangezien we telkens 2 parameters moeten afstellen met elks 10 verschillende keuzemogelijkheden, moeten we in totaal $5 \times 10 \times 10$ combinaties doorzoeken, wat de aanwezigheid van 500 in de tabel verklaart. Daar tegenover staat de quasi-Newton methode met gemiddeld 12 (L1-SVM) en 11 (L2-SVM) trainingen voor de standaard Gaussische kernel, wat een reductie van 97% teweegbrengt in het aantal benodigde modellen tijdens de selectie van de hyperparameters.

Bovendien zien we dat we minder dan het dubbele aantal nodig hebben, wanneer we de weighted Gaussian willen tweaken. Dit is slechts een druppel op een warme plaat in vergelijking met de explosie in verspilde tijd, die we zouden teweegbrengen, wanneer we deze kernel willen gebruiken in combinatie met de grid search. Voor de kleinste data set, *banana*, zou dit resulteren in 5000 te trainen SVM modellen. Echter, wanneer we hetzelfde willen realiseren met de grootste data set, *splice*, zouden we $5 \times 10 \times 10^{60}$ modellen moeten opstellen, alvorens we de optimale keuze kunnen doorvoeren. Het spreekt voor zich dat een computer vrijgesteld wordt van deze onbegonnen klus en de vrijgekomen computertijd beter inzet om nuttiger werk te verrichten.

Bij het analyseren van Tabel 5.4 kunnen we ondanks de enorme reductie in trainingstijd geen noemenswaardige stijgingen in de gemiddelde error rate

vaststellen. In tegendeel! Op 6 van de 10 data sets kunnen we grens verscherpen en bij *heart* is de stijging niet significant. Enkel bij *breast-cancer*, *german* en *titanic* moeten de methoden hun meerdere erkennen in de brute force techniek.

data set	A	B	C	D	E
banana	11.53	10.50	**10.42**	10.54	10.46
breast-cancer	**26.04**	28.81	26.48	28.81	28.79
diabetis	23.53	34.78	**23.34**	34.78	23.75
flare-solar	32.43	35.64	35.32	**32.33**	32.43
german	**23.61**	29.06	24.68	24.50	25.28
heart	**15.95**	22.62	16.02	17.31	15.97
image	2.96	4.25	3.71	**2.30**	3.85
splice	10.88	10.99	10.92	11.14	**10.69**
thyroid	4.80	5.07	4.75	**3.87**	4.07
titanic	**22.42**	23.38	23.09	22.98	22.99
gemiddeld	**17.41**	20.51	17.87	18.86	17.83

Tabel 5.4: Gemiddelde error rate.

5.1.3 Discussie

We kunnen besluiten dat de winst, die geboekt wordt door gebruik te maken van de quasi-Newton methode om de radius/margin bound te optimaliseren, zich voornamelijk situeert in de enorme reductie aan het benodigde aantal SVM trainingen zonder dat het verlies aan nauwkeurigheid de spuigaten uitloopt. We kunnen echter geen van 4 combinaties naar voor schuiven om de brute kracht van de grid search samen met cross-validatie van de troon te stoten. Enkel L1-SVM met de standaard Gaussische RBF kernel lijkt, net zoals in [17] aangegeven, eerder in negatieve zin uit te blinken. Toch moeten we voorzichtig omspringen met de uitspraak om deze overboord te gooien, aangezien in vele gevallen de prestaties wel naar behoren zijn en het gebruik van de l_1-norm in het algemeen aanleiding geeft tot zuinige modellen, i.e. modellen met minder support vectoren.

Zelfs wanneer we op veilig spelen en telkens alle vier de methoden testen, zouden we gemiddeld gezien ook slechts – voor de beschouwde benchmark data – gemiddeld 60 SVMs moeten trainen, wat nog steeds overeenkomt met een reductie van meer dan 87% in vergelijking met grid search in combinatie met cross-validatie.

We verwachten niet de trainingtijd nog naar beneden te halen of de nauwkeurigheid op te krikken door bijvoorbeeld een echte Newton methode in te schakelen, aangezien quasi-Newton methoden de beste trade-off wat betreft snelheid en precisie leveren en de meeste bovengrenzen voor de generalisatie

fout niet tweemaal differentieerbaar zijn, zoals aangegven in [17]. Bovendien wordt in dezelfde paper aangetoond dat het niet nodig is om de exacte locatie van het optimum te vinden, maar dat het aanduiden van een regio met goede parameters belangrijker is. Tot een gelijkaardige conclusie kwam men in [35], waar bovendien verschillende (differentieerbare) bovengrenzen tegen elkaar werden opgezet.

Wat betreft het gebruik van de gewogen Gaussische kernel, kunnen we twee zaken besluiten. Enderzijds lijden de modellen, ondanks een groter aantal in te stellen parameters, niet aan overfittingsverschijnselen en anderzijds gaat de vlieger om elke variabele van een individuele scale factor te voorzien om spectaculaire verbeteringen te bereiken niet op. Dit was ook niet verwacht en geheel in de trend van [14].

5.2 Feature selectie

In deze sectie verschaf ik een overzicht van een vergelijkende studie tussen verschillende feature selectie algoritmen, met name een feature ranking methode berustend op de toekenning van Fisher scores aan de verschillende input variabelen, SVM-RFE en de feature scaling methode gebaseerd op het minimaliseren van de radius/margin bound. Aangezien ik geen enkele toolbox of library vond die alle methodes tezamen aanbood, gebruikte ik een combinatie van de Matlab toolbox Spider[1], Gist-fselect[2] en, om van een *weighted* RBF kernel gebruik te kunnen maken, een door mezelf uitgebreide versie van het door Prof. Dr. Chih-Jen Lin toegestuurd scaled feature selectie algoritme dat gebruik maakt van LIBSVM [10].

We analyseren hoe de verschillende variabele selectie routines het er vanaf brengen bij twee artificiële problemen en bij classificatie van random netwerken.

5.2.1 Artificiële data

In navolging van [73] gebruikte ik volgende twee artificiële data sets om na te gaan hoe de verschillende algoritmen zich gedragen in de aanwezigheid van irrelevante en overbodige features.

Experimenten

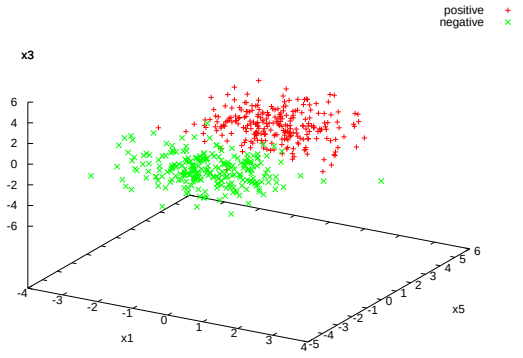
Het experiment bestond eruit om na te gaan of de verschillende methoden in staat waren tussen al de ruis de relevante variabelen te selecteren en te

[1] `http://www.kyb.tuebingen.mpg.de/bs/people/spider/`: R2W2_sel (weighted linear, weighted poly), RFE (linear, poly, rbf).

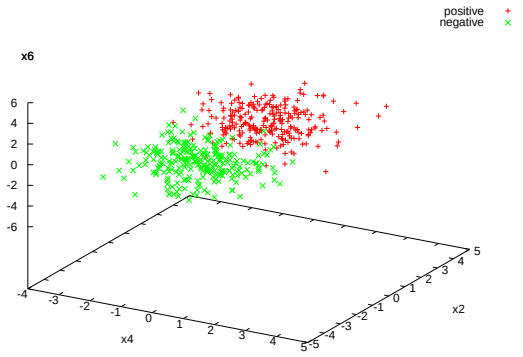
[2] `http://benzer.ubic.ca/gist/fselect.html`: Fisher criterion score.

aanschouwen of de nauwkeurigheid toeneemt naarmate we de door feature selectie procedure aangeduide minder relevante variabelen achterwege laten. We beschouwen volgende twee problemen.

Lineair probleem In het eerste voorbeeld (Figuur 5.1), zijn slechts zes dimensies van de 50 relevant. De kans op klasse label $y = 1$ of -1 is gelijk. Met een kans gelijk aan 0.7 werden de eerste drie features $\{x_1, x_2, x_3\}$ getrokken uit $x_i = yN(i, 1)$, waarbij $N(\mu, \sigma)$ duidt op een normaal verdeling met gemiddelde μ en standaardafwijking σ, en de volgende drie variabelen $\{x_4, x_5, x_6\}$ als $x_i = N(0, 1)$. Zo niet, werden de rollen omgekeerd en de eerste drie gegenereerd uit $x_i = N(0, 1)$ en de variabelen vier tot en met zes als $x_i = yN(i - 3, 1)$. De overige dimensies stelden ruis voor: $x_i = N(0, 20)$, $i = 7, \ldots, 50$.



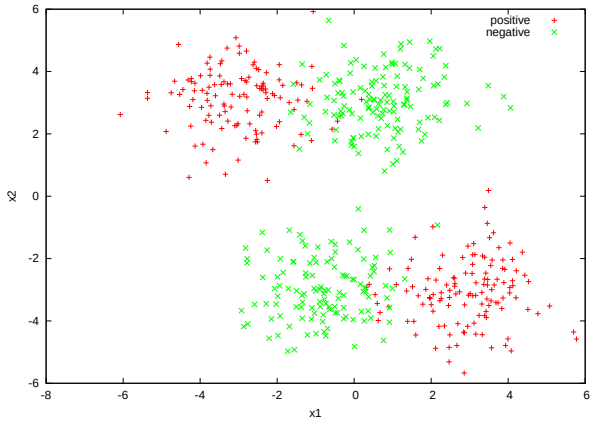
(a)



(b)

Figuur 5.1: **Lineair probleem**. (a) projectie (x_1,x_5,x_3); (b) projectie (x_4,x_2,x_6).

Niet-lineair probleem Voor het tweede voorbeeld (Figuur 5.2), waren twee van de 50 dimensies relevant. Opnieuw was de kans op $y = 1$ of -1 gelijk. De data werd echter gegenereerd op volgende manier: als $y = -1$ dan was met een gelijke kans $\{x_1, x_2\}$ afkomstig uit een bivariate normaal verdeling $N(\mu_1, \Sigma)$ of $N(\mu_2, \Sigma)$, waarbij $\mu_1 = \{-\frac{3}{4}, -3\}$, $\mu_2 = \{\frac{3}{4}, 3\}$ en $\Sigma = I_2$; als $y = 1$ werden ze op gelijkaardige manier vastgelegd, maar dan met $\mu_1 = \{3, -3\}$ en $\mu_2 = \{-3, 3\}$. De overige features $x_i = N(0, 20)$, $i = 7, \ldots, 50$ zijn ruis.



Figuur 5.2: **Niet-lineair probleem**. Projectie (x_1,x_2).

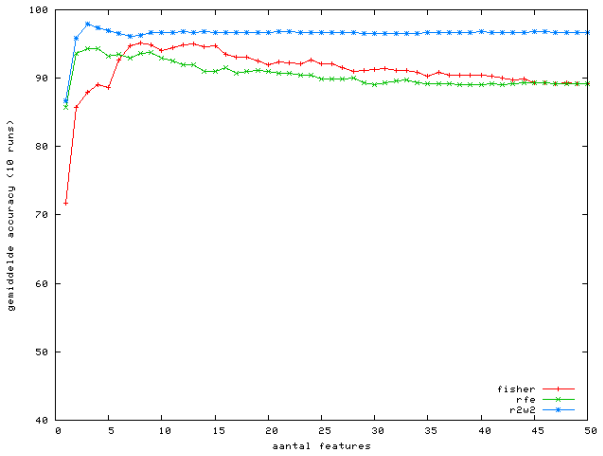
We lieten de drie methoden lopen op training sets met variërende grootte, respectievelijk 10, 20 en 30 voorbeelden, en evalueerden de geconstrueerde classifiers met een test set bestaande uit 500 objecten, die we – net als de training sets – telkens eerst standaardiseren door van elke input variabele het gemiddelde uit de training data af te trekken en te delen door de standaardafwijking, wederom bepaald door de trainingsvoorbeelden.

Voor het lineaire probleem beperkten we ons tot het aanleren van een lineaire beslissingsfunctie in de inputruimte, terwijl we bij het niet-lineaire probleem polynomiale kernels van graad twee uitspeelden tegenover Gaussische RBF kernels.

Resultaten

In Appendix C worden de resultaten, bekomen door elke combinatie van traininggrootte en feature selectie methode tienmaal te herhalen, uitgezet in grafieken die de gemiddelde nauwkeurigheid (*accuracy*) tonen wanneer we de verzameling met features iteratief inkrimpen.

In Figuur 5.3 kunnen we opmerken dat door het afstellen van meerdere schaal factoren bij de methode met een gewogen lineaire kernel, we minder



Figuur 5.3: Lineair probleem – linear kernel – trainingsize 30.

invloed hebben van de irrelevante features en we ook de beste performantie halen wanneer we ons beperken tot een zeer klein aantal features. SVM-RFE presteert nagenoeg gelijkaardig aan de methode waar we eerst de features preprocessen door de Fisher scores te rangschikken, behalve wanneer we het aantal features sterk laten afnemen dan gaat de Fisher-methode sneller de mist in. Ook als we Tabel 5.5 bekijken zien we dat R^2W^2 er het best in slaagt om twee relevante features te selecteren, wat meteen de accuracy resultaten staaft.

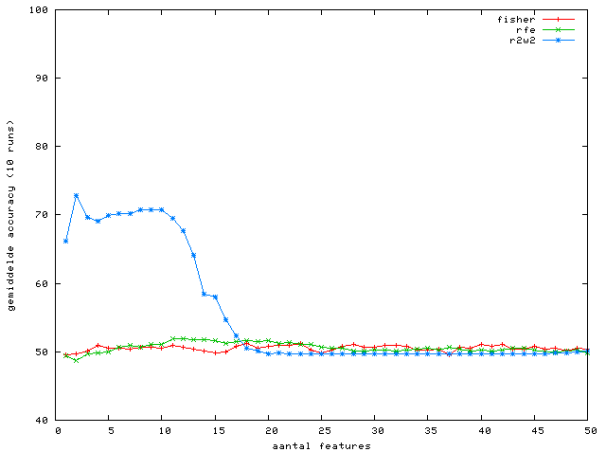
Lineair probleem (linear)	10	20	30	50	100
Fisher score	**4**	5	5	–	–
SVM-RFE	**4**	6	9	–	–
R^2W^2	3	**8**	**10**	–	–

Niet-lineair probleem (poly)	10	20	30	50	100
Fisher	**1**	0	0	–	–
SVM-RFE	**1**	0	**3**	**9**	**8**
R^2W^2	**1**	0	0	–	–

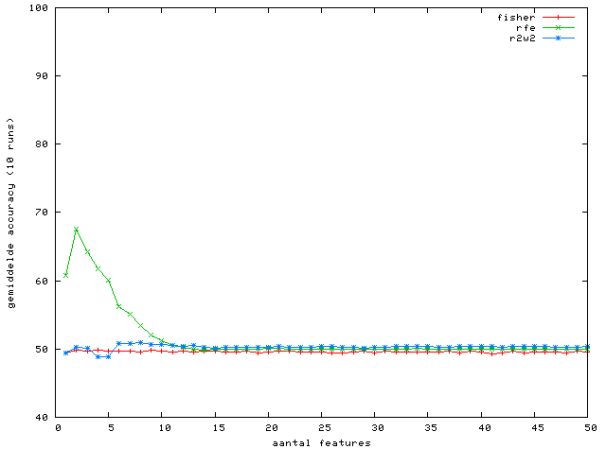
Niet-lineair probleem (rbf)	10	20	30	50	100
Fisher	**1**	0	0	–	–
SVM	0	0	0	**6**	2
R^2W^2	**1**	**1**	**3**	5	**5**

Tabel 5.5: Aantal keer dat twee relevante features overbleven.

Als we kijken naar het niet-lineaire probleem, merken we sterke verschillen bij het gebruik van verschillende kernels. De beste resultaten worden



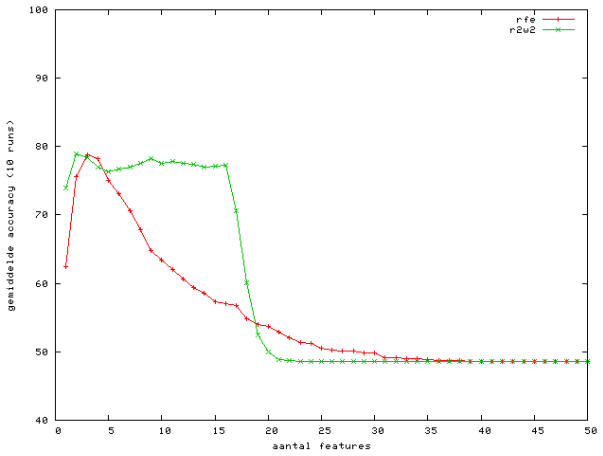
Figuur 5.4: Niet-lineair probleem – Gaussian RBF kernel – trainingsize 30.



Figuur 5.5: Niet-lineair probleem – polynomial kernel – trainingsize 30.

geboekt bij een Gaussische kernel (Figuur 5.4) en met feature scaling als selectie criterium; terwijl we in het geval van een polynomiale kernel (Figuur 5.5) van graad twee beter af zijn met het recursieve feature eliminatie algoritme.

Wanneer we echter 50 trainingssamples (Figuur 5.6) nemen, begint SVM-RFE een waardige concurrent te vormen voor R^2W^2.



Figuur 5.6: Niet-lineair probleem – Gaussian RBF kernel – trainingsize 50.

Discussie

Ondanks de goede performantie van support vector machines bij data met een hoogdimensionaal karakter, leert dit experiment ons dat feature selectie de nauwkeurigheid verhoogt. De prestaties van de verschillende feature selectie routines hangen af van de trainingsgrootte; wat ons geenszins mag verbazen. Immers, wanneer we meer voorbeelden aanbieden, zal de ruis steeds minder significant worden en slechts sporadisch als zinvol worden aangeschouwd.

We kunnen besluiten dat voor de lineaire problemen, we er baat aan hebben om verschillende features in combinatie met anderen te beschouwen, aangegeven door de prestaties van zowel SVM-RFE als R^2W^2, in tegenstelling tot de Fisher ranking score die slechts eenmalig de ranking uitvoert en deze niet meer herziet naarmate er minder ruisfactoren aanwezig zijn.

Op het niet-lineaire probleem gaat deze simplistische methode, onafhankelijk van de gekozen kernel, totaal de mist in; dit vormt evenmin een verrassing aangezien Fisher ranking criterium enkel kijkt naar een individuele variabele en dus geen niet-lineaire verbanden kan herkennen en daarenboven totaal geen rekening houdt met de specifieke kernel. SVM-RFE levert behoorlijke resultaten, al laat dit zich bij de Gaussische kernel pas met een groter aantal voorbeelden in de verf zetten.

We merken dat R^2W^2 superieur is aan de overige methoden. Enkel bij de polynomiale kernel laat deze een steekje vallen, zinvolle verklaring voor dit fenomeen ontbreekt vooralsnog en lijkt af te wijken met de resultaten uit [14, 72, 73].

5.2.2 Classificatie van random netwerken

In het kader van het huidige onderzoek binnen ISLAB, leek het ons nuttig om te kijken wat support vector machines en de geassocieerde feature selectie methoden ons zouden leren, wanneer we ze loslaten op een probleem om random gegenereerde netwerken/grafen te classificeren.

We beperkten ons tot het maken van een onderscheid tussen enerzijds *directed scale-free* (DSF) grafen en anderzijds *small-world* (SW) netwerken. Small-world netwerken [70, 78] vormen een subset van random grafen die voldoen aan het small-world fenomeen, wat inhoudt dat, ondanks het feit dat nodes met een klein aantal buren rechtstreeks gekoppeld zijn, elke node bereikt kan worden door een willekeurige andere node in een klein aantal stappen. Om te spreken van een directed scale-free graaf [6, 77], moeten de meeste nodes in de graaf een lage graad hebben en zijn er slechts enkele nodes aanwezig met een zeer hoge graad, de zogenaamde *highly connected hubs*.

Ondanks dat er al kernel functies ontwikkeld zijn – bijvoorbeeld [33, 34] – die de (lokale) *simulariteit* uitdrukken tussen twee grafen, hebben we bij gebrek aan een publiek beschikbare implementatie gekozen om de classificatie uit te voeren door de drie meest gebruikte kernels – lineaire, polynomiale en Gaussische RBF kernel – los te laten op enkele (globale) statistieken.

Experimenten

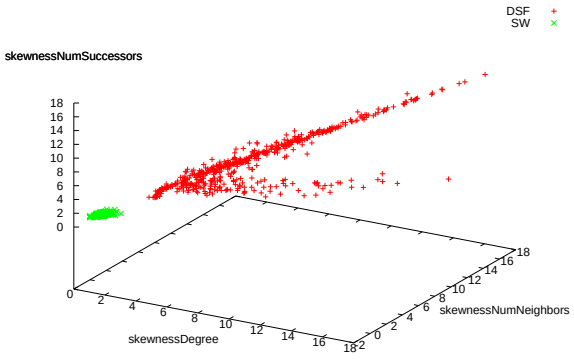
Om het experiment te voltrekken, gebruiken we een genereerde data set bestaande uit 330 small-world en 519 directed scale-free grafen en splitsen deze op in training sets, en corresponderende complementaire verzamelingen aan testgegevens, met variërende grootte (10, 25, 50, 100 en respectievelijk 250 voorbeelden) waarbij we de verhoudingen tussen het aantal DSFs en SWs behouden. We laten de drie verschillende feature selectie methoden – vermeld in het begin van deze sectie – in combinatie met de drie verschillende kernels los op zowel een ongeschaalde versie als een tussen [-1,1] geschaalde versie van de data sets.

We maken ook een verschil qua initiële verzameling aan globale statistieken. In één testcase beschouwen we slechts de tien meest intuïtieve features: gemiddelde in en/of out graad, gemiddeld aantal voorgangers, nakomelingen en buren, diameter, gemiddelde padlengte, clusteringcoëfficiënt (zowel met als zonder lussen in rekening te brengen). De volledige data set bestaat daarentegen uit 72 features waar naast het gemiddelde ook andere statistieken aanwezig waren, zoals daar zijn waarden voor variantie, scheefheid, minimum, maximum, mediaan en som (al dan niet van gekwadrateerde termen).

Resultaten

We bespreken hieronder de verschillende waargenomen tendensen aan de hand van figuren opgenomen in Appendix D en Appendix E die, net zoals bij het experiment op de artificiële problemen, de gemiddelde nauwkeurigheid over 10 runs uitzetten. In plaats van histogrammen, die de relatieve belangrijkheid, bepaald door een gewogen gemiddelde te nemen waarbij de hoogst gerangschikte features een zwaarder gewicht toebedeeld krijgen, van de verschillende features tonen, is daar waar zinvol geacht een plot van de drie meest beduidende, en dus laatst overblijvende, input variabelen toegevoegd.

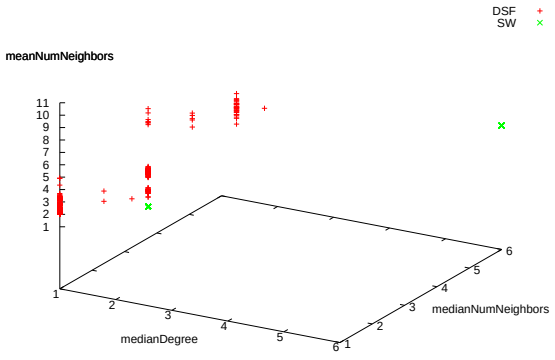
DSF versus SW (72 features) In het geval van de ongeschaalde variant met alle 72 features, zien we dat de twee verschillende klassen grafen perfect lineair te scheiden zijn en dat de beste resultaten dan ook geboekt worden met een lineaire kernel in combinatie met een feature ranking gebaseerd op Fisher scores. Zelfs met slechts 10 trainingsvoorbeelden halen we een nauwkeurigheid van bijna 100 procent.



Figuur 5.7: skewness : Degree – NumNeighbors – NumSuccessors

Uit histogrammen leren we dat de meeste informatie geput wordt uit de *skewness*, een maat voor de asymmetrie van een verdeling, in dit geval van een sterk gecorreleerde verzameling die ons iets leert over het aantal direct geconnecteerde nodes. Figuur 5.7 toont de perfect lineaire scheiding op basis van deze features, die over de variërende grootte uiterst stabiel bleven, en leert ons dat direct-scale free grafen langere uitlopers naar rechts hebben, terwijl de small-world netwerken sterk symmetrische eigenschappen vertonen.

Naarmate we het aantal trainingsvoorbeelden laten toenemen, zien we ook we een betere prestatie van de polynomiale kernel, wederom in combinatie



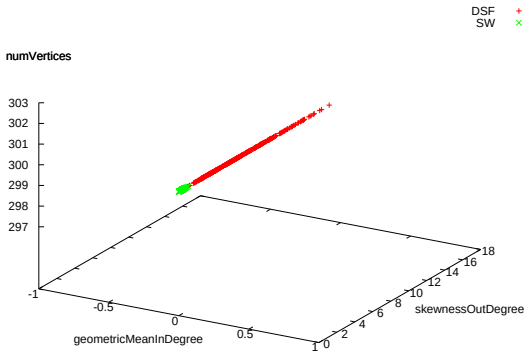
Figuur 5.8: medianDegree – medianNum- en meanNumNeighbors

met Fisher ranking. Wanneer we 250 voorbeelden aanbieden, merken we ook verbetering van de feature scaling methode berustend op de Gaussische RBF kernel, waarbij we – vooruitblikkend op later – een aantal variabelen ('meanNumSuccessors', 'meanNumPredecessors', 'meanNumNeighbors') uit de gereduceerde feature verzameling naar boven zien borrelen. De drie variabelen die echter het langst stand houden, worden getoond op Figuur 5.8, waar we tevens zien dat de SWs twee eilandjes vormen, terwijl er bij de DSFs meer variatie optreedt.

De resultaten voor de geschaalde versie, nog steeds over de volledige verzameling aan features, zijn meer spraakmakend. Hier kunnen we als belangrijkste conclusie formuleren dat na herschaling de methodes over het algemeen (veel) beter presteren en dat de prestaties nagenoeg identiek zijn aan die van de lineaire kernel voor om het even welke feature selectie methode. Enkel wanneer we stapsgewijs de variabelen met de laagste Fisher score elimineren, verliezen we aan nauwkeurigheid bij de polynomiale kernel.

Opmerkelijk is dat zowel de Fisher ranking scores als SVM-RFE (met alle drie de kernels!) ongeveer dezelfde features als in vorige geval selecteren.

Wanneer we kijken naar de performantie van de RBF kernels, leren we dat het afstellen van de regulatieparameter en de kernelparameter met behulp van een grid search logischerwijze betere prestaties aflevert dan de default waarden met zowel C als γ gelijk aan $1/2$. Echter, zoals blijkt uit de resultaten, kan de computationeel onaantrekkelijke zoektocht vermeden worden, door gebruik te maken van de feature scaling methode gebaseerd op de radius/margin bound, waarbij gelijktijdig eveneens de regulatieparameter C bepaald wordt. De drie meest frequent voorkomende features worden getoond op Figuur 5.9 en verschillen klaarblijkelijk sterk met de feature verzamelingen bekomen door het loslaten van de overige combinaties.



Figuur 5.9: geometricMeanInDegree – skewnessOutDegree – numVertices

DSF versus SW (10 features) We kunnen het principe achter kernel methodes, namelijk dat elk probleem lineair te scheiden is en dat het er enkel op neerkomt op de juiste manier naar de data te kijken, ook in de omgekeerde richting proberen uit te buiten. Hiervoor gebruiken we de data sets met beperkt aantal features (10 om precies te zijn) en gaan we na of we met behulp van het inpluggen van een gepaste kernel tot gelijkaardige resultaten komen als de lineaire kernel losgelaten op de volledige verzameling aan input variabelen.

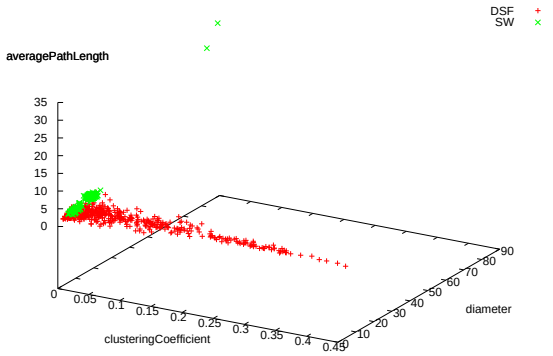
Zowel voor de geschaalde als de ongeschaalde data, merken we dat de lineaire kernel vrij matig presteert. Dit mag ons geenszins verbazen, immers met een lineaire kernel voeren we geen impliciete mapping naar een ruimte met (niet-)lineaire features uit en nemen we de input variabelen zoals ze zijn.

Andere negatieve uitschieters zijn in het geval van de unscaled data de RBF kernel (al dan niet getuned door het uitvoeren van een exhaustive grid search) en in het geval van de scaled data de polynomiale kernel, beiden in combinatie met feature ranking gebaseerd op het Fisher ranking criterium.

Opnieuw merken we de tendens dat wanneer we een groter aantal voorbeelden gebruiken om ons model te trainen, dit in het algemeen tot betere resultaten leidt.

In het geval van de ongeschaalde data, merken we dat polynomiale kernel de beste resultaten boekt wanneer we de feature ranking uitvoeren door de recursieve eliminatie procedure of door gebruik maken van de Fisher scores. Wat de feature scaling methoden op basis van gradiënt informatie afkomstig van de bovengrens op de generalisatiefout betreft, zien we dat de RBF kernel beter geschikt is dan de polynomiale kernel en dat deze de uitstekende resultaten van de andere methoden (in combinatie met de polynomiale kernel) kan bijbenen wanneer we minstens 50 trainingsvoorbeelden gebruiken.

Voor de geschaalde data merken we gelijkaardige resultaten, met uitzondering van de buitengewoon goede resultaten van de weighted polynomiale kernel waarbij we de gewichten optimaliseren door de radius/margin bound te minimaliseren. Ook de RBF kernel scoort met dezelfde optimalisatieroutine naar behoren.



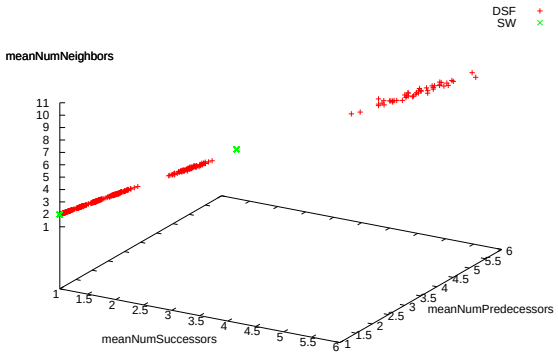
Figuur 5.10: clusteringCoefficient – diameter – averagePathLength

Niettegenstaande we niet dezelfde hoge classificatie rate halen dan wanneer we naar de volledige verzameling aan features kijken, kunnen we uit dit experiment wel een aantal extra opmerkingen afleiden in verband met de keuze van features. Wanneer we de globale tendens beschouwen, merken we pas een serieuze terugval op, wanneer we drie of minder input variabelen beschouwen. Bovendien merken we een verschil in de "beste" features voor de verschillende kernels. Wanneer we polynomiale/lineaire kernel hanteren in samenspraak met SVM-RFE, zien we dat features op Figuur 5.10, aangevuld met 'meanNumNeighbours', schijnbaar de meeste informatie leveren. Nemen we een kijkje naar de Fisher ranking, merken we dat 'clusteringCoefficient' 'meanNumNeighbours' vervangt.

Geheel andere resultaten verkrijgen we wederom bij de weighted RBF kernel, waar de sterk niet-lineaire features getoond op Figuur 5.11 de vaandel dragen, afhankelijk van het aantal trainingsvoorbeelden aangevuld met informatie over 'clusteringCoefficient' of 'diameter' van de graaf.

Discussie

De belangrijkste conclusie die we kunnen formuleren, is dat de small-world netwerken en de directed-scale free grafen, perfect lineair te scheiden zijn op basis van symmetrie kenmerken in de verdeling van buren. Dit verbaast ons niet, aangezien de definitie van directed-scale free grafen postuleert dat



Figuur 5.11: mean : NumSuccessors – NumPredecessors – NumNeighbors

buiten vele nodes met een kleine graad ook enkele *higly-connected hubs* aanwezig zijn, die de langere staarten in de verdeling van het aantal directe buren verklaren.

De bevindingen op de dataset met gereduceerde features staven de resultaten uit [70], zij stellen immers dat de small-world netwerken worden gekaraktiseerd door een lage clusteringcoëfficiënt. De scheiding op basis van dit criterium biedt geen 100 procent uitsluitsel, wat te wijten is aan het feit dat er ook onder de DSFs grafen zijn die eveneens aan het small-world fenomeen voldoen.

Dat de Gaussische RBF kernel totaal andere features naar voren schuift, heeft te maken met de lokale eigenschappen van deze kernel en deze in staat is om geïsoleerde punten, die vaak voorkwamen bij de SWs, van meer uitgespreide clusters te scheiden en de beslissingsfunctie als het ware uiteenvalt in een aantal disjuncte sferen, die telkens data van een welbepaalde klasse bevatten.

Besluit

Tot slot zetten we de belangrijkste bevindingen uit voorgaande hoofdstukken op een rijtje.

Eerst en vooral hebben we, met de *gradiënt-gebaseerde methoden* om de generalisatie fout te minimaliseren, uitstekende werktuigen ter beschikking om op een snelle én accurate manier modelparameters te selecteren om binaire classificatie tot een goed einde te brengen. Zij openen deuren die met traditionele brute force zoekstrategieën vooralsnog gesloten blijven. Het beschouwen van complexere kernels met een groter aantal parameters, zoals bijvoorbeeld de gewogen kernels of het samennemen van een lineaire combinatie van kernels, wordt nu computationeel aantrekkelijk.

Voorts hebben we gezien dat we zowel *SVM-RFE* als *scaled feature selection* kunnen gebruiken om feature selectie door te voeren. De experimentele resultaten bevestigen bovendien dat het elimineren van irrelevante input variabelen, hetzij om de nauwkeurigheid op te drijven hetzij om het aantal metingen bij tijdskritieke toepassingen tot een minimum te beperken, de prestaties van support vector machines kan verbeteren. De beste resultaten worden neergezet door het *scaled feature selection* algoritme, waarmee we daarenboven gelijktijdig de optimale modelparameters kunnen bepalen zonder een sprong te moeten maken naar een andere tijdrovende routine.

De kernel methode *support vector data description* werd besproken als mogelijke piste om het hoofd te bieden aan problemen met één dominante klasse, om naderhand ingezet te worden bij outlier detectie of monitoring. Het verband met binaire classificatie reikt hopelijk in de (nabije) toekomst een oplossing aan voor het vooralsnog open probleem van parameterselectie bij de single-class technieken.

We hebben er bovendien het volste vertrouwen in dat de goede resultaten, zowel bij parameter optimalisatie als feature selectie, neergezet bij binaire classificatie, te evenaren zijn bij het opstellen van regressie modellen met behulp van gelijkaardige technieken. Deze toekomstbeelden tonen dat deze tak in machine learning nog steeds in volle bloei is en de grenzen nog lang niet bereikt zijn.

Wordt vervolgd...

Bibliografie

[1] Amaldi, E., and Kann, V. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science 209*, 1–2 (1998), 237–260.

[2] Ayat, N., Cheriet, M., and Suen, C. Empirical error based optimization of SVM kernels: Application to digit image recognition. In *Proceedings of the International Workshop on Frontiers in Handwriting Recognition* (2002).

[3] Ayat, N., Cheriet, M., and Suen, C. Optimization of the SVM kernels using an empirical error minimization scheme. In *Proc. of the International Workshop on Pattern Recognition with Support Vector Machine* (2002), pp. 354–369.

[4] Bierens, H. J. The logit model: estimation, testing and interpretation, 2004. Lecture notes undergraduate econometrics.

[5] Blum, A., and Langley, P. Selection of relevant features and examples in machine learning. *Artificial Intelligence 97* (1997), 245–271.

[6] Bollobás, B., Borgsy, C., Chayesz, J., and Riordan, O. Directed scale-free graphs. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms* (2003).

[7] Breiman, L., Friedman, J., Olshen, R., and Stone, C. *Classification and Regression Trees.* Chapman & Hall/CRC, Boca Raton, Fla, 1998.

[8] Byrd, R. H., Lu, P., and Nocedal, J. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing 16*, 5 (1995), 1190–1208.

[9] Cauwenberghs, G., and Poggio, T. Incremental and decremental support vector machine learning. In *NIPS* (2000), pp. 409–415.

[10] Chang, C.-C., and Lin, C.-J. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[11] Chang, M.-W., and Lin, C.-J. Leave-one-out bounds for support vector regression model selection. *Neural Computation, 17* (2005), 1188–1222.

[12] Chapelle, O. Feature selection for support vector machines, February 2005.

[13] Chapelle, O., and Vapnik, V. Model selection for support vector machines. In *Advances in Neural Information Processing Systems* (2000).

[14] Chapelle, O., Vapnik, V., Bousquet, O., and Mukhherjee, S. Choosing multiple parameters for support vector machines. *Machine Learning 46 (1)* (2001), 131–159.

[15] Cherkassy, V., and Ma, Y. Practical selection of SVM parameters and noise estimation for svm regression. *Neural Networks 17 (1)* (2004), 113–26.

[16] Christmann, A., Luebke, K., Marin-Galiano, M., and Rüping, S. Determination of hyper-parameters for kernel based classification and regression. 2005.

[17] Chung, K.-M., Kao, W.-C., Sun, T., Wang, L.-L., and Lin, C.-J. Radius margin bounds for support vector machines with the RBF kernel. *Neural Computation 15* (2003), 2643–2681.

[18] Cortes, C., and Vapnik, V. Support-vector networks. *Machine Learning 20 (3)* (1995), 273–297.

[19] Cristianini, N., and Shawe-Taylor, J. *An Introduction to Support Vector Machines (and Other Kernel-Based Learning Methods)*. Cambridge University Press, 2000.

[20] Duan, K., Keerthi, S. S., and Poo, A. N. Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing 51* (2002), 41–59.

[21] Duan, K., and Rajapakse, J. C. SVM-RFE peak selection for cancer classification with mass spectrometry data. In *APBC* (2005), pp. 191–200.

[22] Evans, W., and McLeod, H. Pharmacogenomics – drug disposition, drug targets, and side effects. *The New England Journal of Medicine 348* (2003), 358–349.

[23] Fananapazir, N., Li, M., Spentzos, D., and Aliferis, C. Formative evaluation of a prototype system for automated analysis of mass spectrometry data. In *AMIA Symposium* (2005).

[24] Furey, T. S., Cristianini, N., Duffy, N., Bednarski, D. W., Schummer, M., and Haussler, D. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics 16*, 10 (October 2000), 906–914.

[25] Gold, C., and Sollich, P. Model selection for support vector machine classification. *Neurocomputing 55* (2003), 221–249.

[26] Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D., and Lander, E. S. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science 286*, 5439 (October 1999), 531–537.

[27] Graf, A. B., Smola, A., and Borer, S. Classification in a normalized feature space using support vector machines. *IEEE Transactions on Neural Networks 14 (3)* (2003), 597–605.

[28] Guyon, I., and Elisseeff, A. An introduction to variable and feature selection. *Journal of Machine Learning Research 3* (March 2003), 1157–1182.

[29] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. Gene selection for cancer classification using support vector machines. *Machine Learning 46*, 1-3 (2002), 389–422. Erratum, http://www.clopinet.com/isabelle/Papers/RFE-erratum.html.

[30] Hall, M. A. *Correlation-based Feature Subset Selection for Machine Learning*. PhD dissertation, Department of Computer Science, University of Waikato, 1999.

[31] Hayton, P., Schölkopf, B., Tarassenko, L., and Anuzis, P. Support vector novelty detection applied to jet engine vibration spectra. In *NIPS* (2000), pp. 946–952.

[32] Hertz, J., Krogh, A., and Palmer, R. G. *Introduction to the theory of neural computation*. Persues Books Publishing, Reading, Massachusetts, 1991.

[33] Kashima, H., and Inokuchi, A. Kernels for graph classification. In *1st ICDM Workshop on Active Mining (AM-2002)* (Maebashi, Japan, 2002).

[34] Kashima, H., Tsuda, K., and Inokuchi, A. *Kernel Methods in Computational Biology*. MIT Press, 2004, ch. Kernels for Graphs.

[35] Keerthi, S. S. Efficient tuning of SVM hyperparameters using radius/margin bound and iterative algorithms. *Tech Report CD-01-02* (2001).

[36] KiYoung, L., Dae-Won, K., Doheon, L., and Kwang, L. H. Improving support vector data description using local density degree. *Pattern Recognition 38 (10)* (2005), 1768–1771.

[37] Kohavi, R., and John, G. H. Wrappers for feature subset selection. *Artificial Intelligence 97*, 1-2 (1997), 273–324.

[38] LeCun, Y., Denker, J., Solla, S., Howard, R. E., and Jackel, L. D. Optimal brain damage. In *Advances in Neural Information Processing Systems II* (San Mateo, CA, 1990), D. S.Touretzky, Ed., Morgan Kauffman.

[39] Lin, H.-T., Lin, C.-J., and Weng, R. C. A note on Platt's probabilistic outputs for support vector machines. 2003.

[40] Moya, M., and Hush, D. Network constraints and multi-objective optimization for one-class classification. *Neural Networks 9*, 3 (1996), 463–474.

[41] Moya, M. R., Koch, M. W., and Hostetler, L. D. One-class classifier networks for target recognition applications. In *Proc. World Congress on Neural Networks* (1993), International Neural Network Society (INNS), pp. 797–801.

[42] Mukherjee, S., Tamayo, P., Slonim, D., Verri, A., Golub, T., Mesirov, J., and Poggio, T. Support vector machine classification of microarray data. Tech. rep., AI Memo 1677, Massachusetts Institute of Technology, 1999.

[43] Niven, R. K. Combinatorial information theory: I. philosophical basis of cross-entropy and entropy, 2006.

[44] Platt, J. C. Probabilities for support vector machines. In *Advances in Large Margin Classifiers* (1999), A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, Eds., MIT Press.

[45] Rakotomamonjy, A. Variable selection using SVM based criteria. *JMLR Special Issue on Variable and Feature Selection 3* (March 2003), 1357–1370.

[46] Rätsch, G., Onoda, T., and Müller, K.-R. Soft margins for adaboost. *Machine Learning 42(3)* (2001), 287–320.

[47] Ripley, B. D. *Pattern recognition and neural networks.* Cambridge University Press, 1996.

[48] Roses, A. D. Voor elke patiënt de juiste pil. *Natuur & Techniek 69*, 9 (2001).

[49] Rüping, S. Incremental learning with support vector machines. Tech. rep., Universität Dortmund, 2002.

[50] Sarle, W. S. comp.ai.neural-nets FAQ, 2006. [Online; accessed 15-May-2006].

[51] Saunders, G., Gammerman, A., and Vovk, V. Ridge regression learning algorithm in dual variables. In *Proc. 15th International Conf. on Machine Learning* (1998), Morgan Kaufmann, San Francisco, CA, pp. 515–521.

[52] Schittkowski, K. Optimal parameter selection in support vector machines. *Journal of Industrial and Management Optimization 1 (4)* (2005), 465–476. to appear: Journal of Industrial and Management Optimization (2005).

[53] Schölkopf, B., Burges, C., and Vapnik, V. Extracting support data for a given task. In *First International Conference on Knowledge Discovery & Data Mining* (Menlo Park, 1995), U. Fayyad and R. Uthurusamy, Eds., AAAI Press.

[54] Schölkopf, B., Platt, J., and Smola, A. Kernel method for percentile feature extraction. Tech. Rep. TR MSR 2000-22, Microsoft Research, Redmond, WA, 2000.

[55] Schölkopf, B., Platt, J. C., Shawe-Taylor, B. J., Smola, A. J., and Williamson, R. C. Estimating the support of a high-dimensional distribution. *Microsoft Research TR 87* (1999).

[56] Schölkopf, B., and Smola, A. J. *Learning with Kernels (Support Vector Machines, Regularization, Optimization, and Beyond).* The MIT Press, 2002.

[57] Shawe-Taylor, J., and Cristianini, N. On the generalization of soft margin algorithms. *IEEE Transactions on Information Theory* (1999).

[58] Smits, G. F., and Jordaan, E. M. Using mixtures of polynomial and RBF kernels for support vector regression. In *Proceedings of the 2002 IEEE World Conference on Computational Intelligence* (2002), pp. 2192–2198.

[59] Syed, N., Liu, H., and Sung, K. Incremental learning with support vector machines. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI-99)* (1999).

[60] Tax, D. *One-class classification; Concept-learning in the absence of counter-examples*. PhD thesis, Delft University of Technology, 2001.

[61] Tax, D., and Duin, R. Data domain description using support vectors. In *Proc. of the European Symposium on Artificial Neural Networks '99* (1999).

[62] Tax, D., and Duin, R. Support vector domain description. *Pattern Recognition Letters 20 (10-13)* (1999), 1191–1199.

[63] Tax, D., and Duin, R. Outliers and data descriptions. In *Proc. ASCI 2001, 7th Annual Conf. of the Advanced School for Computing and Imaging (Heijen, NL, May 30-June 1)* (2001), R. Lagendijk, J. Heijnsdijk, A. Pimentel, and M. Wilkinson, Eds., pp. 234–241.

[64] Tax, D., and Duin, R. Uniform object generation for optimizing one-class classifiers. *Journal of Machine Learning Research, Special Issue on Kernel Methods 2 (2)* (2002), 155–173.

[65] Tax, D., and Duin, R. Support vector data description. *Machine Learning 54 (1)* (2004), 45–66.

[66] Tax, D., Ypma, A., and Duin, R. P. W. Support vector data description applied to machine vibration analysis. In *Proceedings of ASCI'99* (1999).

[67] Tsuda, K., Rätsch, G., Mika, S., and Müller, K.-R. Learning to predict the leave-one-out error of kernel based classifiers. *Lecture Notes in Computer Science 2130* (2001).

[68] Vapnik, V. *Statistical Learning Theory*. Wiley, New York, 1998.

[69] Wagner, M., Naik, D., and Pothen, A. Protocols for disease classification from mass spectrometry data. *Proteomics 3*, 9 (2003), 1692–1698.

[70] Watts, D. J., and Strogatz, S. H. Collective dynamics of 'small-world' networks. *Nature 393(6684)* (1998), 440–2.

[71] Wellens, P. Een introductie tot support vector machines. Master's thesis, Universiteit Antwerpen (Departement Wiskunde-Informatica), 2005.

[72] Weston, J., Elisseeff, A., Schölkopf, B., and Tipping, M. Use of the zero-norm with linear models and kernel methods. *Journal of Machine Learning Research 3* (March 2003), 1439–1461.

[73] Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., and Vapnik, V. Feature selection for SVMs. In *NIPS* (2000), pp. 668–674.

[74] Wikipedia. Linear discriminant analysis — Wikipedia, the free encyclopedia, 2006. [Online; accessed 6-April-2006].

[75] Wikipedia. Messenger RNA — Wikipedia, the free encyclopedia, 2006. [Online; accessed 15-April-2006].

[76] Wikipedia. Naive Bayes classifier — Wikipedia, the free encyclopedia, 2006. [Online; accessed 25-March-2006].

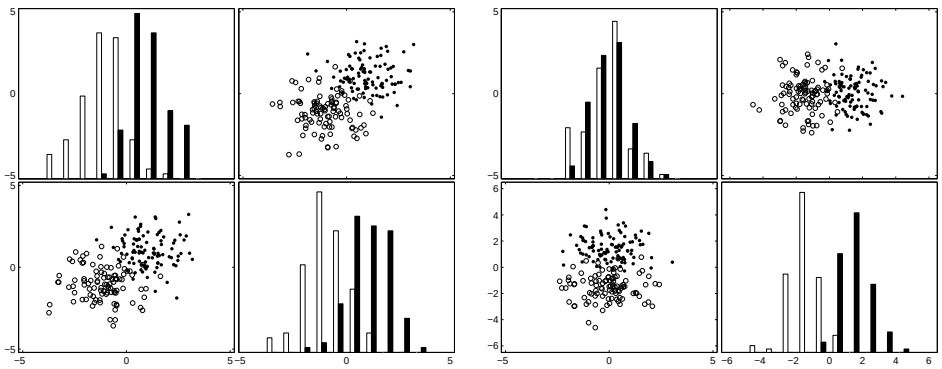
[77] Wikipedia. Scale-free network — Wikipedia, the free encyclopedia, 2006. [Online; accessed 25-April-2006].

[78] Wikipedia. Small-world network — Wikipedia, the free encyclopedia, 2006. [Online; accessed 25-April-2006].

[79] Zhu, C., Byrd, R. H., and Nocedal, J. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. *ACM Transactions on Mathematical Software 23*, 4 (1997), 550–560.

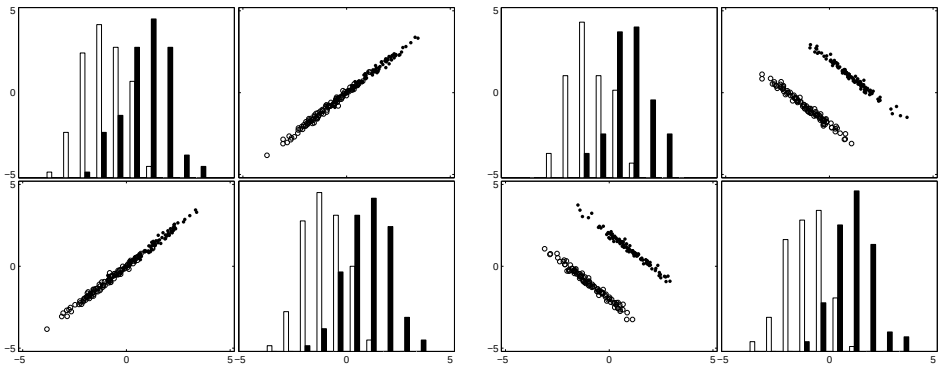
Bijlage A

Small but Revealing Examples [28]



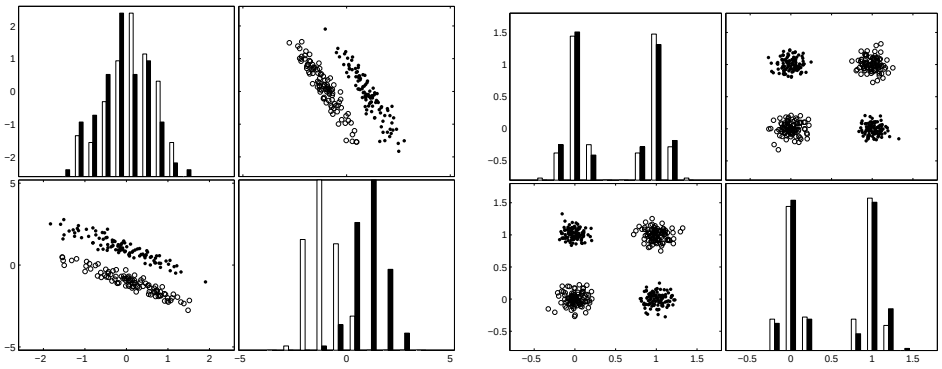
(a) (b)

Figuur A.1: **Information gain from presumably redundant variables.** (a) A two class problem with independently and identically distributed (i.i.d.) variables. Each class has a Gaussian distribution with no covariance. (b) The same example after a 45 degree rotation showing that a combination of the two variables yields a separation improvement by a factor $\sqrt{2}$. I.i.d. variables are not truly redundant.



(a) (b)

Figuur A.2: **Intra-class covariance.** In projection on the axes, the distributions of the two variables are the same as in the previous example. (a) The class conditional distributions have a high covariance in the direction of the line of the two class centers. There is no significant gain in separation by using two variables instead of just one. (b) The class conditional distributions have a high covariance in the direction perpendicular to the line of the two class centers. An important separation gain is obtained by using two variables instead of one.



(a) (b)

Figuur A.3: **A variable useless by itself can be useful together with others.** (a) One variable has completely overlapping class conditional densities. Still, using it jointly with the other variable improves class separability compared to using the other variable alone. (b) XOR-like or chessboardlike problems. The classes consist of disjoint clumps such that in projection on the axes the class conditional densities overlap perfectly. Therefore, individual variables have no separation power. Still, taken together, the variables provide good class separability.

Bijlage B

Resultaten optimaliseren van (hyper)parameters

banana		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	11.53 ± 0.66
L1-SVM rm rbf	9.73	10.50 ± 0.44
L2-SVM rm rbf	8.38	10.42 ± 0.46
L1-SVM rm weighted rbf	12.78	10.54 ± 0.48
L2-SVM rm weighted rbf	10.57	10.46 ± 0.48

Tabel B.1: Resultaten voor data set banana

breast-cancer		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	26.04 ± 4.74
L1-SVM rm rbf	10.43	28.81 ± 4.56
L2-SVM rm rbf	11.80	26.48 ± 4.73
L1-SVM rm weighted rbf	11.66	28.81 ± 4.56
L2-SVM rm weighted rbf	19.20	28.79 ± 4.56

Tabel B.2: Resultaten voor data set breast-cancer

diabetis		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	23.53 ± 1.73
L1-SVM rm rbf	14.92	34.78 ± 2.17
L2-SVM rm rbf	9.84	23.34 ± 1.75
L1-SVM rm weighted rbf	16.05	34.78 ± 2.17
L2-SVM rm weighted rbf	18.64	23.75 ± 1.60

Tabel B.3: Resultaten voor data set diabetis

flare-solar		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	32.43 ± 1.82
L1-SVM rm rbf	9.80	35.64 ± 1.63
L2-SVM rm rbf	8.36	35.32 ± 1.72
L1-SVM rm weighted rbf	14.81	32.33 ± 1.82
L2-SVM rm weighted rbf	10.17	32.43 ± 1.65

Tabel B.4: Resultaten voor data set flare-solar

german		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	23.61 ± 2.07
L1-SVM rm rbf	13.81	29.06 ± 2.06
L2-SVM rm rbf	11.05	24.68 ± 2.54
L1-SVM rm weighted rbf	33.45	24.50 ± 2.21
L2-SVM rm weighted rbf	28.48	25.28 ± 2.27

Tabel B.5: Resultaten voor data set german

heart		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	15.95 ± 3.26
L1-SVM rm rbf	14.14	22.62 ± 3.89
L2-SVM rm rbf	11.00	16.02 ± 3.25
L1-SVM rm weighted rbf	21.56	17.31 ± 3.26
L2-SVM rm weighted rbf	24.61	15.97 ± 3.52

Tabel B.6: Resultaten voor data set heart

image		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	2.96 ± 0.60
L1-SVM rm rbf	14.05	4.25 ± 0.71
L2-SVM rm rbf	13.45	3.71 ± 0.65
L1-SVM rm weighted rbf	18.20	2.30 ± 0.51
L2-SVM rm weighted rbf	18.95	3.85 ± 0.48

Tabel B.7: Resultaten voor data set image

splice		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	10.88 ± 0.66
L1-SVM rm rbf	13.45	10.99 ± 0.70
L2-SVM rm rbf	19.40	10.92 ± 0.72
L1-SVM rm weighted rbf	41.25	11.14 ± 0.56
L2-SVM rm weighted rbf	14.80	10.69 ± 0.83

Tabel B.8: Resultaten voor data set splice

thyroid		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	4.80 ± 2.19
L1-SVM rm rbf	15.08	5.07 ± 2.08
L2-SVM rm rbf	11.25	4.75 ± 2.05
L1-SVM rm weighted rbf	23.37	3.87 ± 1.71
L2-SVM rm weighted rbf	18.74	4.07 ± 2.06

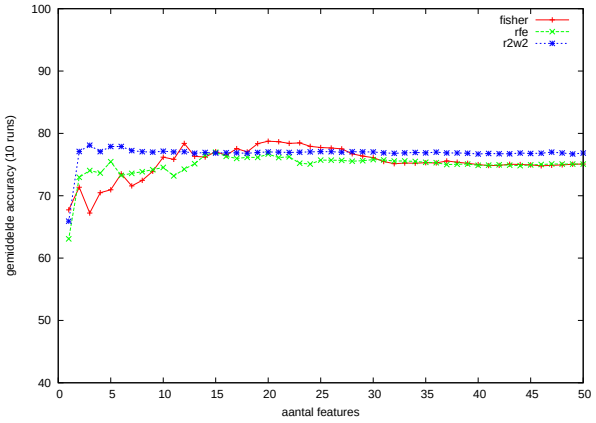
Tabel B.9: Resultaten voor data set thyroid

titanic		
	average nr SVM trainings	average error rate ± std
L1-SVM grid rbf	500.00	22.42 ± 1.02
L1-SVM rm rbf	8.06	23.38 ± 1.65
L2-SVM rm rbf	6.76	23.09 ± 1.21
L1-SVM rm weighted rbf	10.11	22.98 ± 1.49
L2-SVM rm weighted rbf	7.23	22.99 ± 1.17

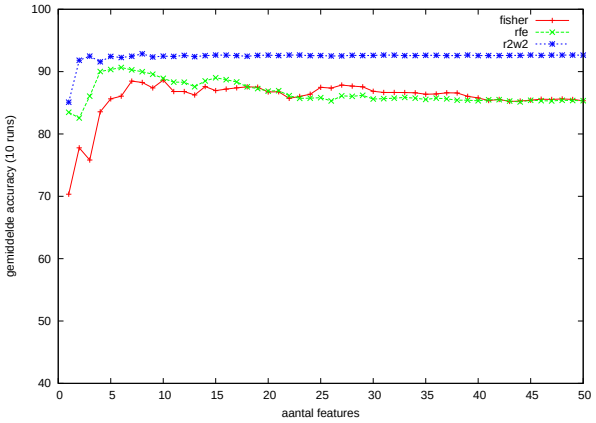
Tabel B.10: Resultaten voor data set titanic

Bijlage C

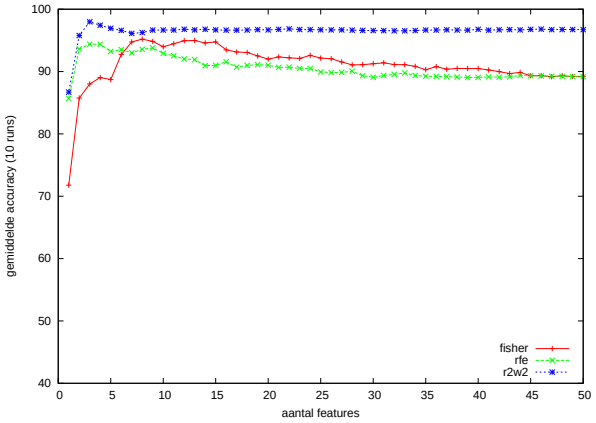
Resultaten artificiële problemen



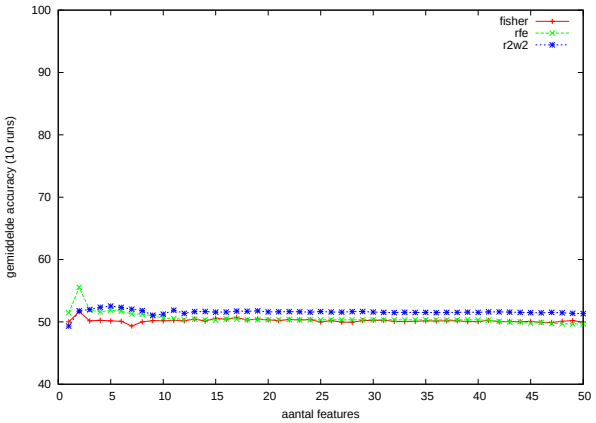
Figuur C.1: Lineair probleem – linear kernel – trainingsize 10.



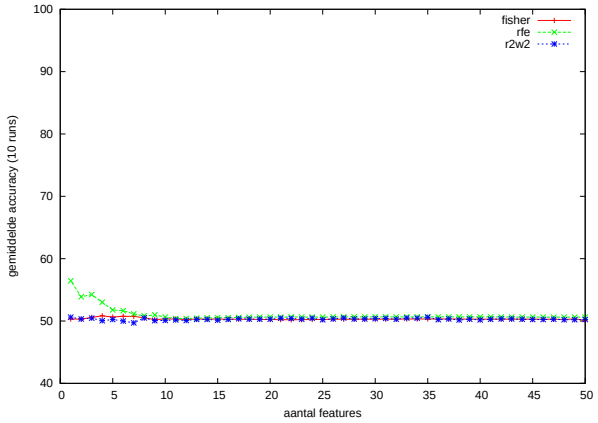
Figuur C.2: Lineair probleem – linear kernel – trainingsize 20.



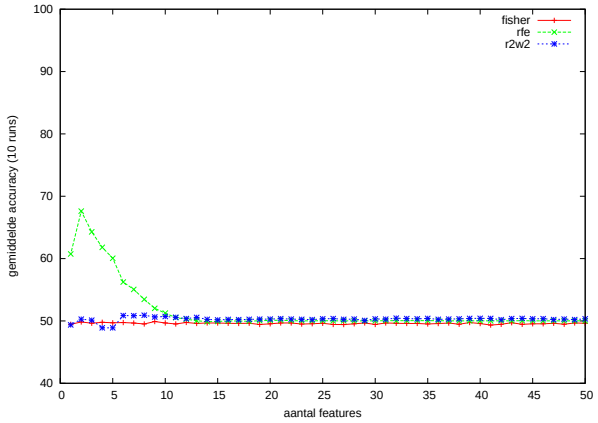
Figuur C.3: Lineair probleem – linear kernel – trainingsize 30.



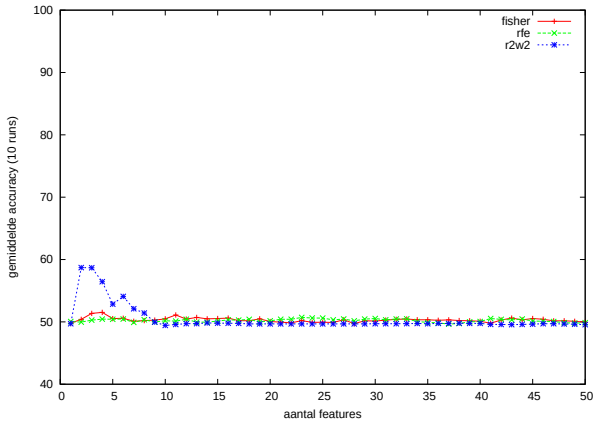
Figuur C.4: Niet-lineair probleem – polynomial kernel – trainingsize 10.



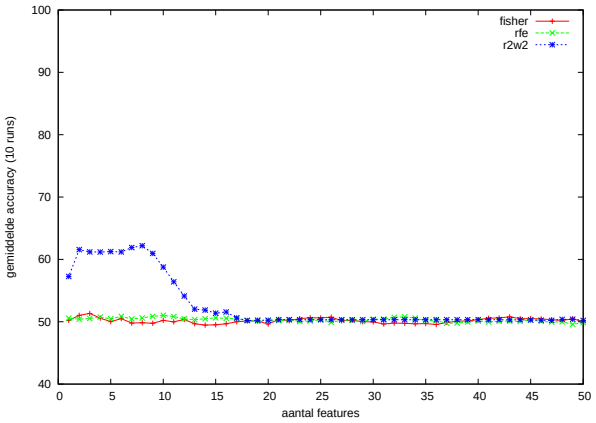
Figuur C.5: Niet-lineair probleem – polynomial kernel – trainingsize 20.



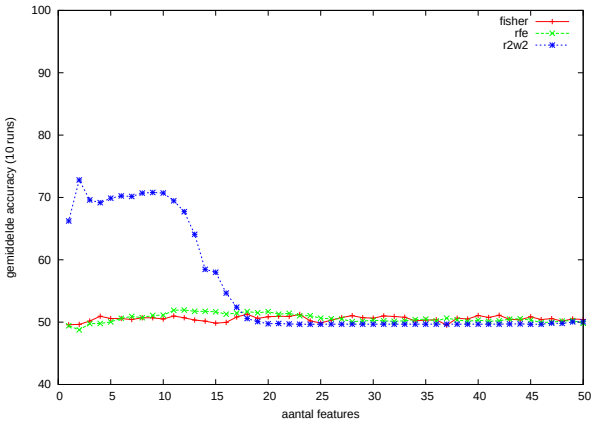
Figuur C.6: Niet-lineair probleem – polynomial kernel – trainingsize 30.



Figuur C.7: Niet-lineair probleem – Gaussian RBF kernel – trainingsize 10.



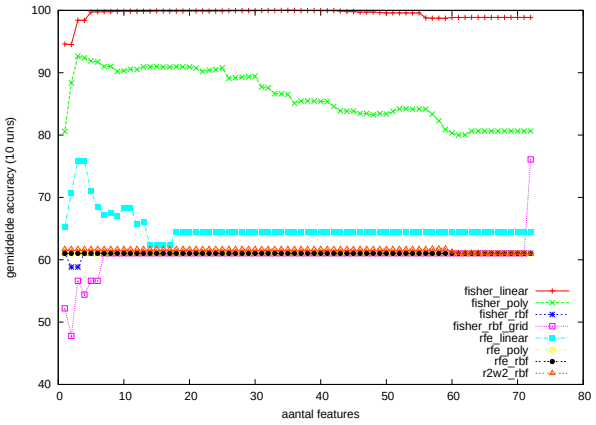
Figuur C.8: Niet-lineair probleem – Gaussian RBF kernel – trainingsize 20.



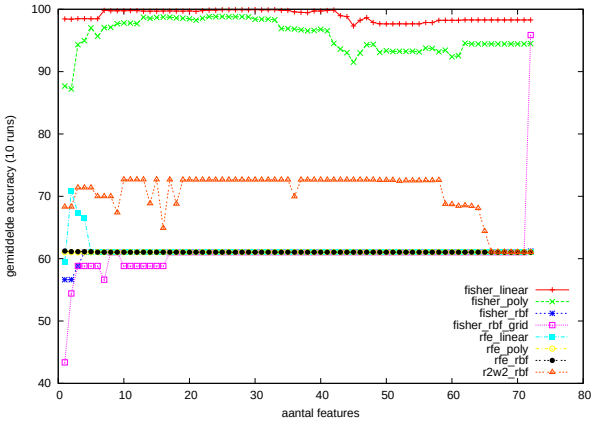
Figuur C.9: Niet-lineair probleem – Gaussian RBF kernel – trainingsize 30.

Bijlage D

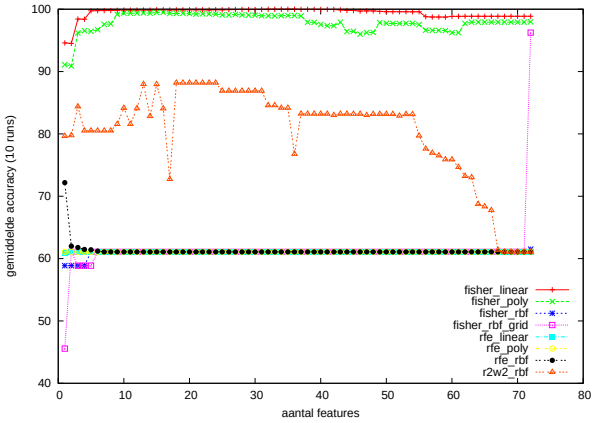
Resultaten DSF versus SW (72 features)



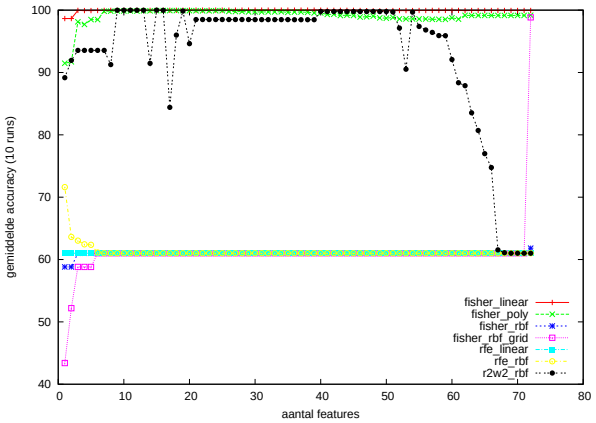
Figuur D.1: DSF versus SW – not scaled – trainingsize 10.



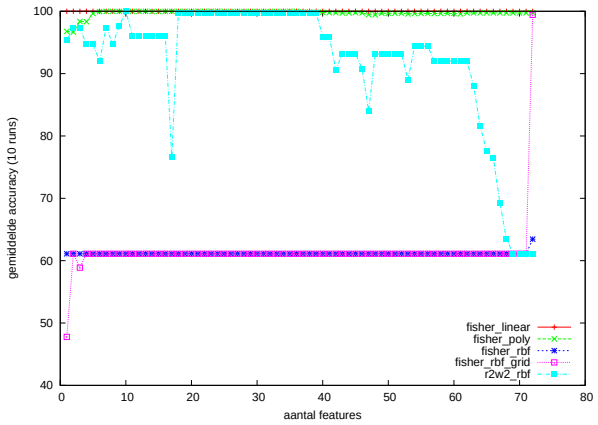
Figuur D.2: DSF versus SW – not scaled – trainingsize 25.



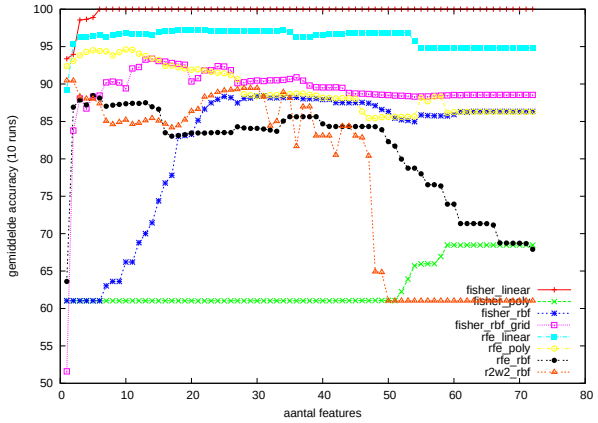
Figuur D.3: DSF versus SW – not scaled – trainingsize 50.



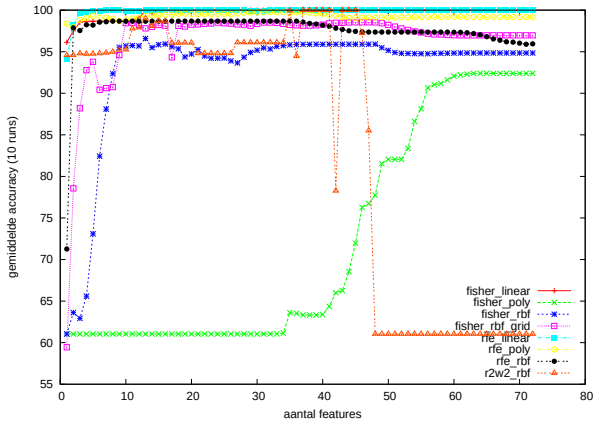
Figuur D.4: DSF versus SW – not scaled – trainingsize 100.



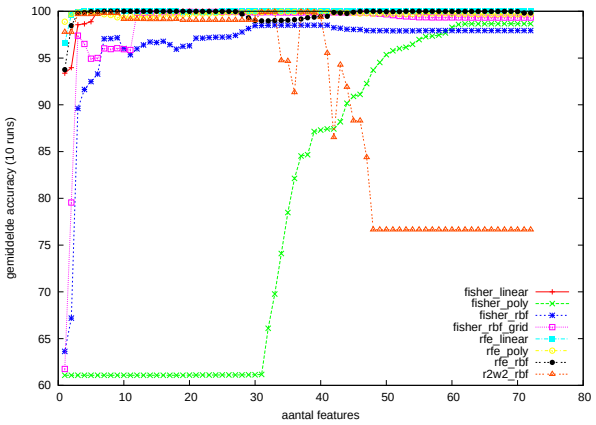
Figuur D.5: DSF versus SW – not scaled – trainingsize 250.



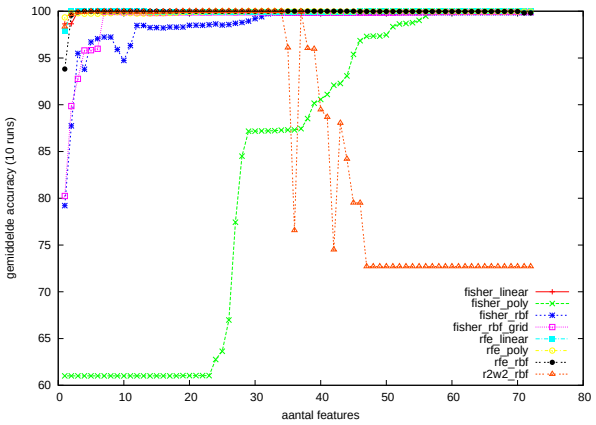
Figuur D.6: DSF versus SW – scaled – trainingsize 10.



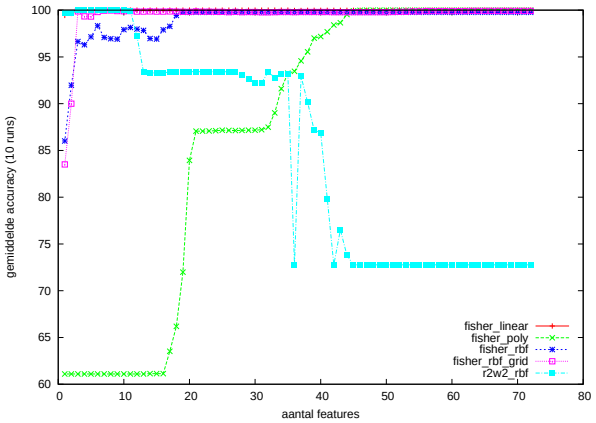
Figuur D.7: DSF versus SW – scaled – trainingsize 25.



Figuur D.8: DSF versus SW – scaled – trainingsize 50.



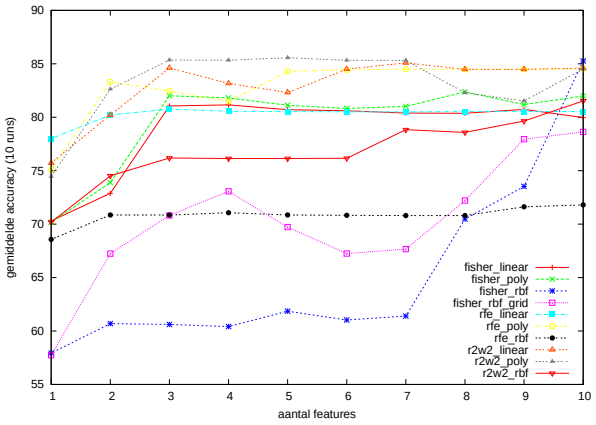
Figuur D.9: DSF versus SW – scaled – trainingsize 100.



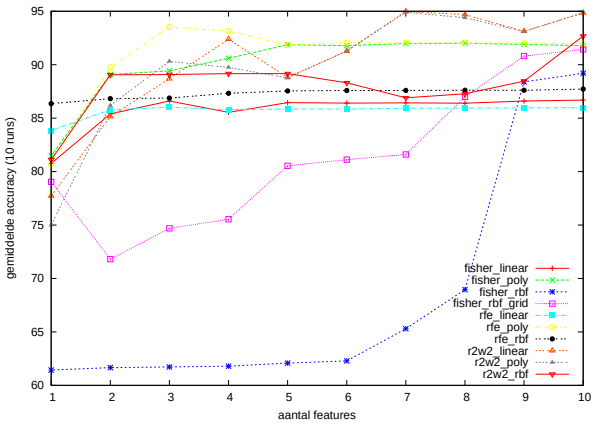
Figuur D.10: DSF versus SW – scaled – trainingsize 250.

Bijlage E

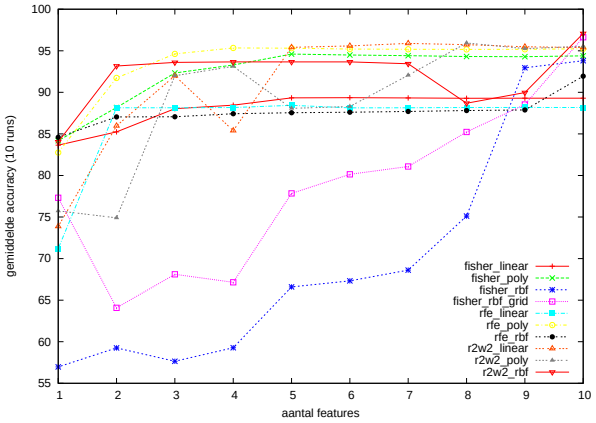
Resultaten DSF versus SW (10 features)



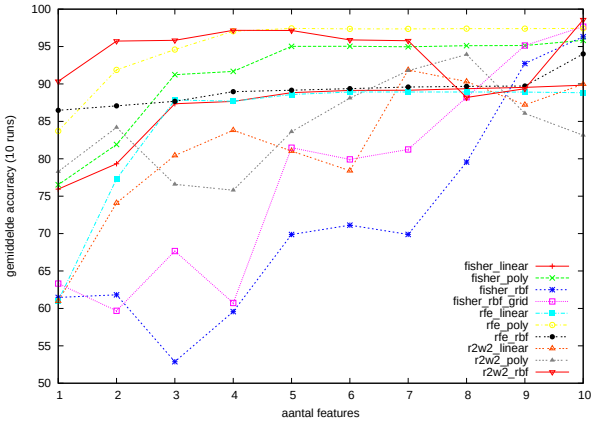
Figuur E.1: DSF versus SW – not scaled – trainingsize 10.



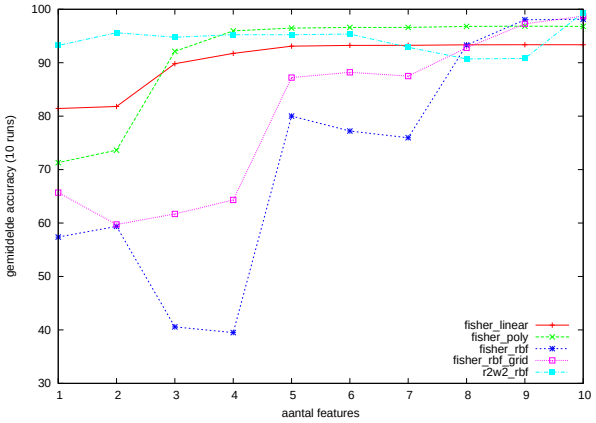
Figuur E.2: DSF versus SW – not scaled – trainingsize 25.



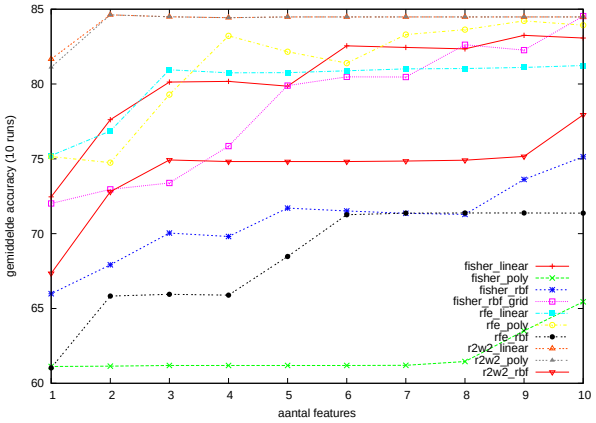
Figuur E.3: DSF versus SW – not scaled – trainingsize 50.



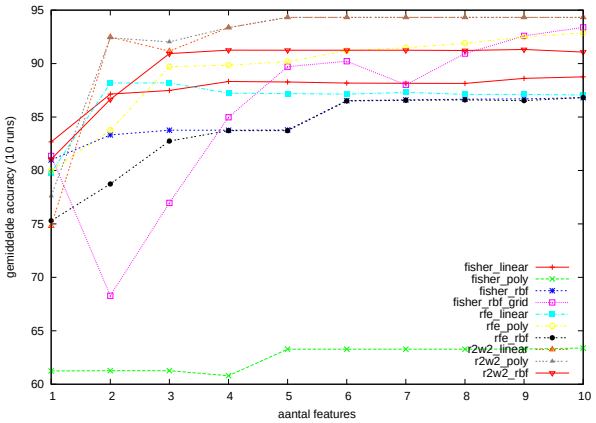
Figuur E.4: DSF versus SW – not scaled – trainingsize 100.



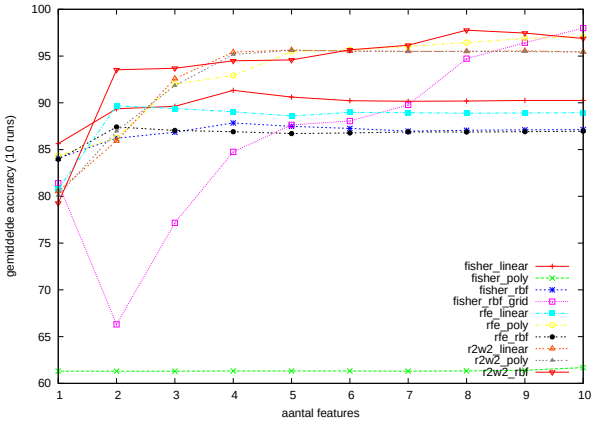
Figuur E.5: DSF versus SW – not scaled – trainingsize 250.



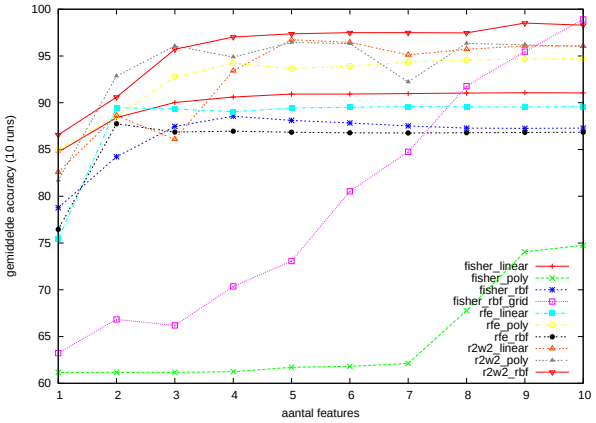
Figuur E.6: DSF versus SW – scaled – trainingsize 10.



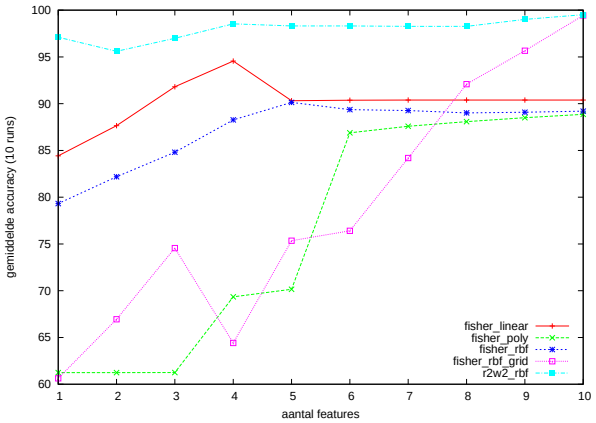
Figuur E.7: DSF versus SW – scaled – trainingsize 25.



Figuur E.8: DSF versus SW – scaled – trainingsize 50.



Figuur E.9: DSF versus SW – scaled – trainingsize 100.



Figuur E.10: DSF versus SW – scaled – trainingsize 250.