Universitetet i Oslo
Institutt for informatikk

Optimeringsmetoder anvendt på klassifikasjon av hyperspektrale data

Hovedoppgave

Bjørn Terjei Austenaa

2. mai 2006

Forord

Formålet med denne oppgaven har vært å prøve ut Support Vector Machine (SVM) baserte klassifikatorer på hyperspektrale data, samt å sette seg inn i teorien i kvadratisk optimering.

Dette har jeg gjort ved å løse det kvadratiske programmet C-SVM ved hjelp av indrepunktsmetoden. Jeg har også benyttet meg av kjernefunksjoner.

Jeg har implementert fire forskjellige algoritmer som løser C-SVM problemet og nærliggende problemer. Jeg har brukt dem til å klassifisere det hyperspektrale datasettet Fontainebleau og sammenlignet klassifikasjonsnøyaktighetene og kjøretidene.

Jeg tar også opp spørsmålet om en egenskapsreduksjon kan ha noe for seg i form av bedre klassifikasjonsnøyaktighet og kjøretid.

Jeg vil takke mine veiledere Professor Geir Dahl og Førsteamanuensis Anne Schistad Solberg for gode idéer og verdifulle kommentarer.

Bjørn Austenaa. Oslo, 1. mai 2006.

Innhold

1 Hyperspektrale data/bilder - problemstillinger og metoder 1
1.1 Introduksjon 1
1.2 Klassifikasjon fra høydimensjonale rom 2
1.2.1 Klassifikasjon ved Nevrale nettverk 5
1.2.2 Klassifikasjon ved Bayes regel 6
1.2.3 Klassifikasjon med Support Vector Machine 6
1.3 Egenskapsseleksjon 9
1.3.1 Foroverseleksjon 10
1.3.2 Bakoverseleksjon 10
1.3.3 Branch and Bound 10
1.4 Egenskapstransformasjon 11
1.4.1 Prinsipalkomponentanalyse (PCA) 11
1.4.2 Fishers lineære diskriminant (FLD) 11
1.4.3 Fouriertransform som egenskapsuttrekning 14

2 Kvadratisk optimering og indrepunktsmetoden 15
2.1 Optimering 15
2.2 Lagrange relaksasjon 16
2.2.1 Indikator funksjonen 16
2.2.2 Nedre grense på optimal verdi 17
2.2.3 Lagrange duale problem 17
2.2.4 Karush-Kuhn-Tucker optimalitetskrav 18
2.3 Indrepunktsmetoden anvendt på lineært problem 18
2.4 Indrepunktsmetoden anvendt på kvadratisk problem 20
2.4.1 Newtons metode 21
2.4.2 Barriere parameter 22
2.4.3 Skritt lengde 22
2.4.4 Stoppkriterier 22

3 Klassifikasjonsteori 23
3.1 Klassifikasjon 23
3.2 Valg av diskriminantfunksjon 23
3.3 Kjernefunksjoner 23

3.4 VC-Teori . . . 25
3.5 Trening og validering . . . 25
3.6 Klassifikasjon av testsett . . . 26
3.7 En enkel klassifikasjonsmetode: senter problemet . . . 26

4 Klassifikasjon med Support Vector Machine 29
4.1 Hard margin problemet . . . 29
4.1.1 Lagrange relaksasjon . . . 29
4.1.2 Karush-Kuhn-Tucker's komplemetæritetskrav . . . 30
4.1.3 Det dual problemet . . . 31
4.2 Myk margin: C-SVM problemet . . . 31
4.2.1 Lagrange relaksasjon . . . 31
4.2.2 Beregning av b . . . 33
4.3 Myk margin 2: ν-SVM . . . 33
4.4 "Proximal" eller regularisert SVM . . . 34
4.5 Indrepunktsmetoden anvendt på det duale av SVM-problemet 35
4.6 Multiklasse problemet . . . 36
4.7 Beskrivelse av algoritmer . . . 37
4.7.1 Senter algoritmen . . . 37
4.7.2 PathQuad algoritmen . . . 37
4.7.3 OnLine algoritmen . . . 37
4.7.4 QuadProg1 algoritmen . . . 37
4.7.5 QuadProg2 algoritmen . . . 38

5 Klassifikasjon med kjernefunksjoner 39
5.1 Klassifikasjon med forskjellige kjernefunksjoner av sjakkbrettdatasettet . . . 39
5.2 Klassifikasjon med gaussisk kjernefunksjon av virvelfunksjonen 40

6 Klassifikasjon av Fontainebleau datasettet 44
6.1 Fontainebleau datasettet . . . 44
6.2 Normalisering . . . 44
6.3 Validering av parameterverdier . . . 45
6.4 Forvirringsmatrise og klassifikasjonsmål . . . 45
6.5 Validering: Innstilling av parameterverdier . . . 46
6.6 Klassifikasjonsresultater . . . 47
6.7 Egenskapstransform med Fishers Lineære Diskriminant (FLD) 49
6.8 Egenskapstransform med prinsipalkomponentanalyse . . . 54

7 Oppsummering og videre arbeide 58

Appendices 61

A Programmkode **62**
A.1 Klassifikatorer 62
A.1.1 Indrepunktsmetoden 62
A.1.2 Matlabs kvadratiske optimerer 66
A.1.3 On-line algoritmen 66
A.1.4 Senter algoritmen 68
A.1.5 Proximal SVM algoritmen 71
A.2 Kjernefunksjoner 72
A.2.1 Gaussisk kjerne 72
A.2.2 Sigmoid kjerne 73
A.2.3 Polynomisk kjerne 74
A.2.4 Spektral Angle kjerne 74
A.3 Preprosessering 74
A.3.1 Normalisering 74
A.3.2 Egenskapsutvelgelse med Fisher Lineare Diskriminant 75
A.3.3 Egenskapsutvelgelse med Prinsipal Komponent Analyse 76
A.4 Kjøringsalgoritmer 77
A.4.1 Datautvelgelse og preprosessering for Fontainebleau datasettet kalt rosis joined i matlabfilene 77
A.4.2 Datasettutvelgesle og preprosessering for andre datasett 82
A.4.3 Kjerneutregning, Klassifikasjon og utskrifter 84

Kapittel 1

Hyperspektrale data/bilder - problemstillinger og metoder

1.1 Introduksjon

Hyper- og multispektrale data får vi fra satellitter og fly for å overvåke været og jordens overflate. Bilder med ca 8-10 spektrale bånd kalles multispektrale bilder, mens bilder med flere spektrale bånd kalles hyperspektrale bilder. I hyperspektrale bilder kan det være opptil flere hundre spektrale bånd.

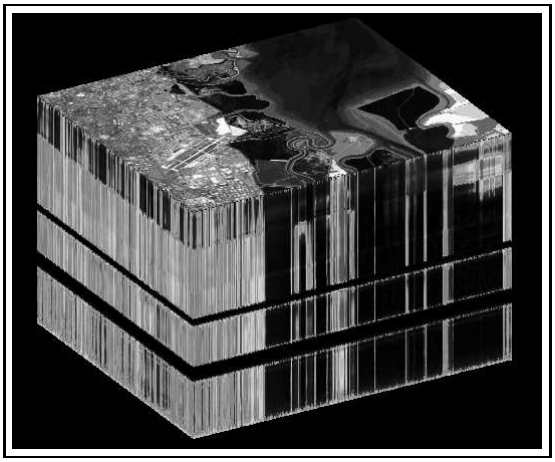
Hvert "bånd" kan man se på som et gråtonebilde for en viss frekvens med bølgelengder fra ca 400nm til ca 700nm. Det tilsvarer frekvenser fra de ultrafiolette og til de infrarøde frekvensene.

Bilder tatt fra satellitter har en oppløsning fra 1km til 30m. Bilder tatt med fly kan ha en oppløsning helt ned til 1 meter.

Setter man i sammen alle båndene får man en bilde-kube som vi kan se i figur 1.1. For hvert piksel i bildet får vi en verdi for hvert bånd. Disse verdiene kaller vi for egenskaper. For eksempel for vanlige fargebilder ville vi hatt 3 bånd eller tre egenskaper, ett for hver av primærfargene rødt, grønt og blått. Samlet sett utgjør egenskapene til et piksel en egenskapsvektor. Egenskapsvektoren kan man se på som en spektral signatur til pikselet.

I motsetning til radar, som er en aktiv sensor i det at den sender ut energi som den måler refleksjonen på, er de hyperspektrale sensorene ikke aktive i at de ikke sender ut energi, men bare leser av refleksjonen av sollyset fra bakken. Dette gjør at man er avhengig av å ta bilder i godt vær da skyene ellers ville absorbert mye av strålingen.

Det er tre måter man kan se på hyperspektrale data på. Det første er bilde rommet. Det er den mest naturlige måten å se bildet på. Man kan se bildet enten som et gråtone bilde for ett bånd om gangen, eller som et fargebilde for tre bånd om gangen der man gir båndene fargene rødt grønt og blått. I figur 1.2 ser vi et gråtone bilde over Oslo. Her kan man sammenligne nabopiksler og deres romlige variasjon.



Figur 1.1: En bildekube.

Den andre måten er spektralrommet. Her ser man på hvert piksel individuelt som en funksjon av bølgelengden. Hvis man kan bruke responsen på bølgelengden til å klassifisere pikslene gir det en enkel klassifikasjonsmetode, men derimot hvis pikslene er en del av en tekstur vil nabopikslene også ha noe å si for hvilken klasse pikselen tilhører og da kan man ikke bare basere seg på spektral rommet. Se figur 1.3 for plott av spektralrommet til et piksel med 81 egenskaper/frekvenser.

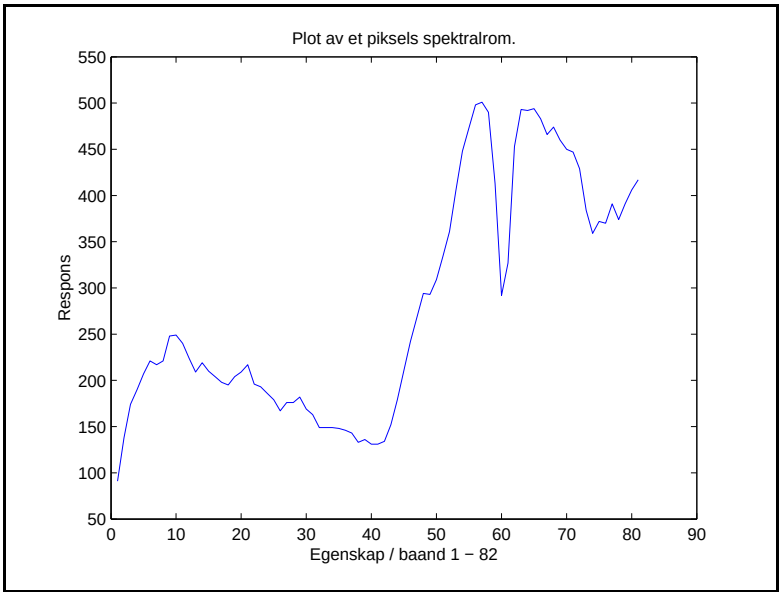
Den tredje måten å se på dataene på er i egenskapsrommet. Man ser på hvert bånd som en egenskap og en dimensjon i egenskapsrommet. Hvis man har to egenskaper kan man plotte de mot hverandre i et spredningsplott. Ved å lete etter samlinger kan man med stor sannsynlighet si at de pikslene som havner på samme sted i spredningsplottet tilhører samme klasse. For et eksempel på et spredningsplott med tre klasser i to dimensjoner se figur 1.4. Man ser fra figuren at en av klassene skilles godt ut mens to av dem ligger over hverandre.

1.2 Klassifikasjon fra høydimensjonale rom

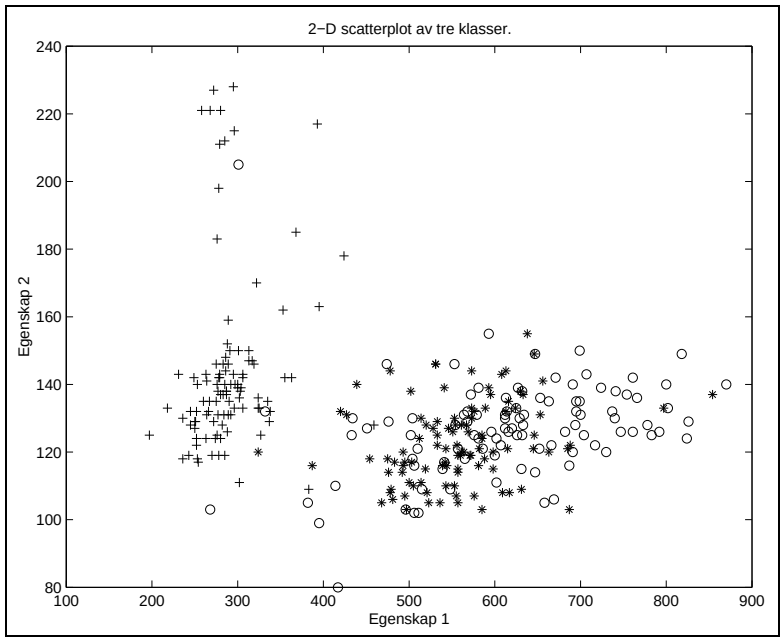
I klassifikasjon ønsker man å bestemme for hvert piksel en tilhørighet til bestemte klasser. For eksempel i et bilde over et skogområde kan man ønske å bestemme hvilke vegetasjonstyper man har. Man velger ut deler av bildet til å representere de forskjellige vegetasjonstypene. Disse brukes så til å trene opp en klassifikasjonsalgoritme. For å få et godt resultat er det viktig at dette treningssettet oppfyller tre kriterier:

- Utfyllende: Hvert piksel må ha en logisk tilhørighet til en klasse i treningssettet.

Figur 1.2: Bilderommet



Figur 1.3: Plott av et piksels egenskaper som en funksjon i spektralrommet.



Figur 1.4: Et spredningsplott av tre klasser med to egenskaper. Vi ser at en av klassene kan skilles nesten helt fra de to andre, mens de to siste overlapper hverandre. Kanskje disse tre klassene kunne skilles helt i et høyere dimensjonalt rom?

- Separerbart: Klassene må kunne skilles ved hjelp av egenskapene.
- Informasjons verdi: Klassene må bære den informasjonen brukeren ønsker.

Klassifikasjonsalgoritmen består av funksjoner som skiller klassene kalt diskriminantfunksjoner. Hvis vi antar at det hyperspektrale bildet har N bånd kan vi for hvert piksel danne en vektor X som inneholder responsen til pikselen i de N båndene. Hvis vi har M forskjellige klasser trenger vi M funksjoner $\{g_1(X), g_2(X), \ldots, g_M(X)\}$. De må være laget slik at hvis X tilhører klasse i vil $g_i(X)$ gi størst verdi. Da får vi følgende bestemmelses regel: Hvis ω_i betegner klasse i kan vi bestemme at X tilhører klasse ω_i hvis og bare hvis $g_i(X) \geq g_j(X)$ for alle $j = 1, 2, ...M$.

Spørsmålet er hvordan finne de M diskriminantfunksjonene. To av de mest vanlige metodene er en iterativ metode som benytter seg av nevrale nettverk og en statistisk metode som bygger på Bayes regel. Disse to metodene er hentet fra kilde [8]. En tredje metode som har blitt populær i de siste årene kalles Support Vector Machine.

1.2.1 Klassifikasjon ved Nevrale nettverk

Den iterative metoden trenger et sett med forhåndsklassifiserte piksler som den skal bruke til å bestemme diskriminantfunksjonene. Disse velger man en parametrisk form til som f. eks.:

$$g_1(X) = a_{11}x_1 + a_{12}x_2 + b_1$$

$$g_2(X) = a_{21}x_1 + a_{22}x_2 + b_2$$

I starten setter man a'ene og b'ene tilfeldig. Så setter man treningsdataene inn i funksjonene og ser om de klassifiserer riktig. Hvis de klassifiserer riktig gjør man ingen ting, men hvis de klassifiserer feil øker man eller "belønner" diskriminantfunksjonen for den riktige klassen og minsker eller "straffer" diskriminantfunksjonene for de andre klassene. For eksempel hvis X tilhører klasse ω_1, men $g_2(X) \geq g_1(X)$ øker vi g_1 og minker g_2 ved å sette:

$$\begin{array}{lll} a'_{11} = a_{11} + \lambda x_1 & a'_{12} = a_{12} + \lambda x_2 & b'_1 = b_1 + \lambda \\ a'_{21} = a_{21} - \lambda x_1 & a'_{22} = a_{22} - \lambda x_2 & b'_2 = b_2 - \lambda \end{array}$$

Dette repeterer man helt til det ikke er flere eller lite nok feilklassifiseringer. Hvis man har mange egenskaper blir det mange variable og tidkrevende utregninger. Siden det er en heuristisk metode er det vanskelig å estimere hvor mange iterasjoner som er nødvendig. Vanligvis bruker den mye lenger tid enn en ikke-iterativ metode og egner seg da ikke så godt til problemer der en må gjøre treningen flere ganger.

1.2.2 Klassifikasjon ved Bayes regel

Den statistiske metoden bruker treningsdataene til å estimere en sannsynlighetsmodell for hver klasse i steden for en empirisk iterativ metode. Hvert piksel blir tildelt den klassen som er mest sannsynlig etter Bayes regel. Gitt apriori sannsynligheter $P(\omega_i)$ for hver klasse ω_i $i = 1, \ldots, M$ og de betingede sannsynligheten $P(X|\omega_i)$, som vil si sannsynligheten for X gitt at X tilhører klasse ω_i, får vi de aposteriori sannsynlighetene eller bestemmelsesfunksjonene $P(X|\omega_i)P(\omega_i)$. Da får vi at X tilhører klasse ω_i hvis $P(X|\omega_i)P(\omega_i) \geq P(X|\omega_j)P(\omega_j)$ for alle $j \neq i$.

Ofte kan man anta en normal- eller Gauss-fordelt tetthetsfunksjon. Da får man

$$P(X|\omega_i) = \frac{1}{\sqrt{(2\pi)^N |\Sigma_i|}} e^{-1/2(X-\overline{X}_i)^T \Sigma_i^{-1}(X-\overline{X}_i)} \; i = 1, \ldots, M$$

hvor N er dimensjonen, $\overline{X}_i$ er middelverdi(vektor)en til klasse ω_i, Σ_i er kovariansmatrisen, $|\Sigma_i|$ er determinanten til Σ_i, Σ_i^{-1} er den inverse og M er antall klasser.

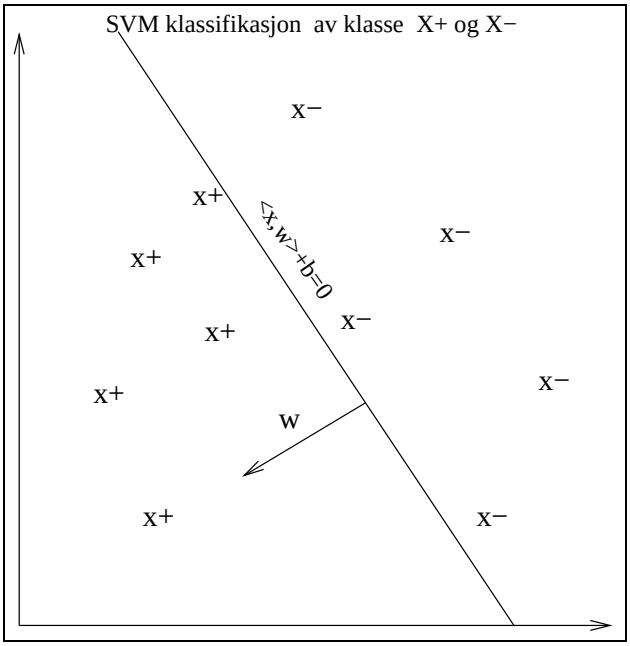
Hvis det blir mange egenskaper og få treningspiksler som det ofte er i hyperspektrale bilder vil $\overline{X}_i$ og Σ_i bli store og unøyaktige. For N egenskaper må man for hver klasse estimere N middelverdier i $\overline{X}_i$ og $(N^2 + N)/2$ for kovariansmatrisen da den er symmetrisk. For eksempel for et 5 klasse system gir dette:

N	$5N$	$5(N^2 + N)/2$
5	25	75
10	50	275
20	100	1050
50	250	6375
200	1000	100500

Vi ser at det er antall parametre i kovariansmatrisen som dominerer. Ettersom antall egenskaper øker blir beregnings kompleksiteten stor. Hughes fenomen sier at med et begrenset antall treningspiksler vil klassifikasjonsnøyaktigheten nå et maksimum når man øker beregningskompleksiteten for så å minke igjen. Dette kaller man for *"dimensjonsforbannelsen"*. Man har for få treningspiksler i forhold til dimensjonen og unøyaktigheten til estimatene forvirrer klassifikasjonen. Hvis man ikke kan få flere treningspiksler må man redusere antall egenskaper ved en utvelgelse eller en projeksjon til en lavere dimensjon.

1.2.3 Klassifikasjon med Support Vector Machine

Support Vector Machine (SVM) er en rask klassifikator for 2-klasse problemer. I SVM ønsker man å finne et hyperplan som skiller de to klassene med størst mulig margin.



Figur 1.5: SVM klassifikasjon av to klasser $x+$ og $x-$. w er normalvektoren til diskriminantplanet, og b bestemmer avstanden fra origo.

Hyperplanet representeres ved en normalvektor w og en avstand b, slik at hyperplanet er gitt ved de x slik at $\langle w, x\rangle + b = 0$. Se figur 1.5.

Det er vanlig å skalere w og b slik at de nærmeste punktene til hyperplanet tilfredsstiller $|\langle w, x_i\rangle + b| = 1$. Dette gir en "kanonisk" form til hyperplanet slik at $y_i(\langle w, x_i\rangle + b) \geq 1$ med $y \in \{-1, +1\}$ for hhv. $x-$ og $x+$.

Avstanden fra de nærmeste punktene, kalt støttevektorer (Support Vector), og til hyperplanet blir da $1/||w||$. Se figur 1.6. For å finne størst mulig margin kan man maksimere $1/||w||$ eller man kan minimere $||w||$. I SVM-litteraturen er det vanlig å minimere $\frac{1}{2}||w||^2$ siden dette gir et noe enklere problem å jobbe videre med.

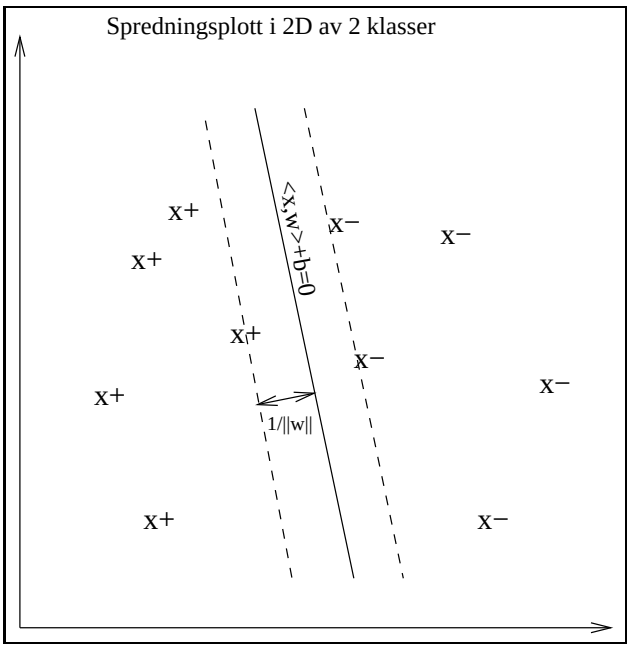
Optimeringsproblemet for m punkter blir da:

$$\begin{aligned} \min \quad & \frac{1}{2}||w||^2 \\ \text{forutsatt at} \quad & y_i(\langle w, x_i\rangle + b) \geq 1, \quad i = 1..m \end{aligned} \tag{1.1}$$

For en optimal w kan vi klassifisere en vilkårlig vektor x til klasse $+1$ eller -1 slik:

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

der sign-funksjonen er fortegnsfunksjonen som gir $+1$ for positive verdier, -1 for negative verdier og 0 kan settes vilkårlig til $+1$ eller -1.



Figur 1.6: Avstanden mellom planet $\langle x, w\rangle + b = 0$ og de nærmeste x er $1/\|w\|$. Vi kan se at når marginen maksimeres treffer den to $x-$ punkter på høyre side og ett $x+$ punkt på venstre side. Disse stopper marginen fra å utvide seg mer og fungerer som støtter på hver side slik at hyperplanet låses i posisjon i midten.

En svakhet med denne metoden er at den krever at punktene x_i skal kunne skilles med et hyperplan noe som ofte ikke er tilfelle. Dette vil vi se nærmere på senere.

Noe av kraften i SVM er at den kan skrives om til å bruke kjernefunksjoner (kernels). Kjernefunksjonen projiserer dataene opp i et høyeredimensjonalt rom hvor dataene i større grad kan separeres lineært. Man skulle tro at denne høyere dimensjonen skulle være en ulempe pga. "dimensjonsforbannelsen", men dette er ikke helt riktig. Dette skyldes den enkle separasjonsfunksjonen, hyperplanet. Den krever ikke at man må estimere så mange parametre som hos Bayes metode som gjør at en høy dimensjon gir unøyaktige estimater ved få treningspunkter.

SVM klassifikatoren er allikevel ikke helt uproblematisk. Valget av en kjernefunksjon, dvs. projeksjonen, er avgjørende. Den må ha riktig kompleksitet. Har funksjonen for høy kompleksitet blir dataene overtilpasset og tilfeldige variasjoner i treningsettet som ikke finnes i testsettet kan få for stor betydning. Generaliserings feilen øker. Man kan si at når kompleksiteten blir stor blir dataene memorert av klassifikatoren. I tillegg kan det være uteliggere i datasettet som da kan gi en uheldig effekt. Dette er illustrert i [11].

1.3 Egenskapsseleksjon

I egenskapsseleksjon ønsker man å redusere antall egenskaper eller bånd slik at kompleksiteten reduseres, men samtidig beholde nok egenskaper for en god klassifikasjon. Problemet blir å finne de egenskapene som gir den beste klassifikasjonen, ikke for mange og ikke for få. Man kan velge de egenskapene som individuelt sett virker best, men ofte er det slik at også kombinasjonen av egenskaper har noe å si på grunn av kovariansen mellom egenskapene.

Hvis man ønsker å velge ut en delmengde k av m egenskaper og skal teste alle mulige valg, gir dette $\binom{m}{k} = \frac{m!}{(m-k)!k!}$ kombinasjoner som kan bli et svært stort tall. For eksempel hvis du ønsker å velge ut 10 av 50 egenskaper blir dette: $\binom{50}{10} = \frac{50!}{(50-10)!10!)} = 10.272.278.170$ som er over 10 milliarder kombinasjoner, 10 av 100 egenskaper gir: $\frac{100!}{(100-10)!10!)} = 17.310.309.456.440$ over 17 billioner kombinasjoner! Det blir fort svært upraktisk å teste alle mulige kombinasjoner.

Målet er å finne de egenskapene som sammen gir den beste klassifikasjonen innen en rimelig tid. I egenskapsseleksjonen må man ha en seleksjonsalgoritme og en kriteriefunksjon. Seleksjonsalgoritmen lager forskjellige subsett og velger det beste ved hjelp av kriteriefunksjonen. Det finnes både optimale og suboptimale/heuristiske seleksjonsalgoritmer. De optimale kan bare brukes så lenge det ikke er mere enn noen få titals egenskaper, ellers blir den kombinatoriske kompleksiteten for stor, derfor skal vi bare se på suboptimale metoder.

Noen familier av disse er Foroverseleksjon, Bakoverseleksjon og Flytende søk [15]. "Branch and Bound" er an annen metode som finner den globalt optimale verdien uten å prøve alle kombinasjoner, men den krever en monoton testfunksjon [3] [12].

1.3.1 Foroverseleksjon

I forover seleksjon velges først den beste individuelle egenskapen. Så velger man den som klassifiserer best sammen med den første. Slik fortsetter man og velger den egenskapen som klassifiserer best helt til man har valgt nok egenskaper ved at enten klassifikasjonen er god nok eller man har nådd en begrensning på antall egenskaper.

1.3.2 Bakoverseleksjon

I bakover seleksjon gjør man det motsatt ved at man begynner med alle egenskapene og så velger bort den egenskapen som ga største forbedring eller minste loss i testfunksjonen. Dette gjentar man til egenskapsrommet er redusert nok og klassifikasjonen er god.

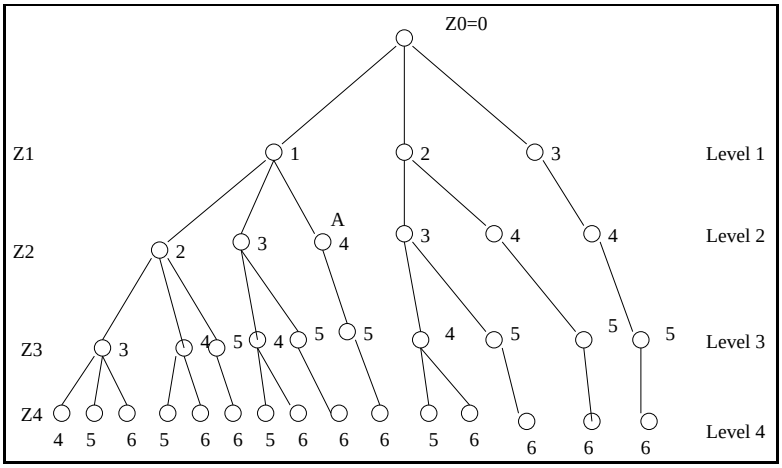
Et problem med forover og bakover seleksjon, er at når en egenskap først er valgt eller valgt bort kan den ikke velges tilbake igjen. Dette er det Flytende søk metoden tar hensyn til. Det er to versjoner av flytende søk, en som bygger på forover seleksjon og en som bygger på bakover seleksjon.

1.3.3 Branch and Bound

Branch and Bound brukes for å finne optimale løsninger til kombinatoriske optimeringsproblemer uten å prøve alle mulige løsninger. Dette krever en monoton testfunksjon. Isteden for å nummerere opp alle kombinasjoner av en delmengde m av n egenskaper, nummereres kombinasjonene til de $\overline{m} = m-n$ egenskapene som velges bort. På den måten minner den litt om bakover seleksjon ved at man tar bort egenskaper.

$(z_1, ..., z_{\overline{m}})$ er settet med ikke valgte egenskaper der z_i tar verdier fra 1 til n. Siden permutasjoner av disse ikke gir noen nye løsninger kan vi anta at de ordnes i stigende rekkefølge. Hvis $J_{\overline{m}}(z_1, ..., z_{\overline{m}})$ er kriteriefunksjonen der $z_1, ..., z_{\overline{m}}$ er slettet fra de n egenskapene da er problemet å finne den optimale sekvensen $(z_1^*, ..., z_{\overline{m}}^*)$ slik at $J_{\overline{m}}(z_1^*, ..., z_{\overline{m}}^*) = maks\, J_{\overline{m}}(z_1, ..., z_{\overline{m}})$. Alle enumerasjoner av $(z_1, ..., z_{\overline{m}})$ kan skrives som et løsningstre. Se figur 1.7.

Antar at $J_1(z_1) \geq J_2(z_1, z_2) \geq ... \geq J_{\overline{m}}$ siden J er monoton. Hvis den beste kombinasjonen man hittil har funnet gir $J_{\overline{m}}(z_1, ..., z_{\overline{m}}) = \alpha$ og man søker videre og finner en $J_{\overline{m}}(z_1, ..., z_k) \leq \alpha$ for $k < \overline{m}$ har man allerede en dårligere løsning som etter antagelsen vil bare bli dårligere ved å ta bort flere fra løsningen. Derfor trenger man ikke å søke videre på det treet. Hvis



Figur 1.7: Branch and Bound tre.

egenskapene er ordnet riktig i dette treet kan man unngå å måtte søke igjennom store deler av treet.

1.4 Egenskapstransformasjon

En annen måte å redusere antall egenskaper enn en seleksjon er å lage nye egenskaper som en lineær kombinasjon av de gamle. For eksempel $y = A^T x$ der x er en vektor med de gamle egenskapene, A er en projeksjonsmatrise og y er en vektor med de nye egenskapene som en lineær kombinasjon av de gamle. To slike metoder er Prinsipalkomponentanalyse (PCA) og Fishers lineære diskriminant (FLD) [15].

1.4.1 Prinsipalkomponentanalyse (PCA)

Prinsipalkomponentanalyse eller Karhunen-Loève transform går ut på å finne projeksjonsmatrisen A slik at $y = A^T x$ er ukorrelert. Dette får vi når korrelasjonsmatrisen $R_y = E[yy^T]$ er diagonal. Vi har:

$$R_y = E[yy^T] = E[A^T xx^T A] = E[A^T R_x A]$$

Der R_x er korrelasjonsmatrisen til x. For at R_y skal være diagonal må A diagonalisere R_x. Dette får vi ved å velge søylevektorene i A som de ortogonale egenvektorene til R_x siden R_x er symmetrisk.

1.4.2 Fishers lineære diskriminant (FLD)

Fishers lineære diskriminant bygger på Fishers diskriminant mål (FDM). FDM var opprinnelig utviklet for to klasser, men den kan også generaliseres

til å gjelde for flere enn to klasser. FDM måler avstanden mellom klassene (between class scatter) delt på variansen innenfor klassene (within class scatter). For to klasser har vi:

$$FDR = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

Den gir et mål på hvor godt egenskapene klarer å separere klassene. Fishers lineære diskriminant benytter seg av dette målet generalisert for flere klasser, til å lage en egenskapstransformasjon som reduserer antall egenskaper til én mindre enn antall klasser mens den samtidig maksimerer dette målet.

For å generalisere FDM til M klasser definerer vi innenfor-klasse-variansen:

$$S_w = \sum_{i=1}^{M} P_i S_i$$

hvor S_i er kovariansmatrisen for klasse i

$$S_i = E[(x - \mu_i)(x - \mu_i)^T]$$

og P_i er a priori sannsynligheten for klasse i: $P_i = n_i/N$ der n_i er antall punkter fra klasse i.

Mellom-klasse-avstanden defineres som:

$$S_b = \sum_{i=1}^{M} P_i(\mu_i - \mu_0)(\mu_i - \mu_0)^T$$

der μ_i er middelverdi-vektor for klasse i og μ_0 er den globale middelverdi-vektoren:

$$\mu_0 = \sum_{i=1}^{M} P_i \mu_i.$$

Ut i fra S_w og S_b danner vi M-klassegeneraliseringen av FDM:

$$J = trasen\{S_w^{-1} S_b\}$$

Vi ønsker å finne den projeksjonen $y = A^T x$ fra $\Re^m$ til $\Re^l$ som maksimerer klasseseparasjonsmålet J. For y har vi $S_{yw} = A^T S_{xw} A$ og $S_{yb} = A^T S_{xb} A$, som gir:

$$\begin{aligned} J_y &= trasen\{S_{yw}^{-1} S_{yb}\} \\ &= trasen\{(A^T S_{xw} A)^{-1} (A^T S_{yb} A)\}, \\ &\quad \text{som vi kaller } J_y(A). \end{aligned}$$

Maksimum for $J_y(A)$ finner vi ved $\frac{\delta J_y(A)}{\delta A} = 0$. Som kan forkortes til:

$$S_{xw}^{-1} S_{xb} A = A S_{yw}^{-1} S_{yb} \tag{1.2}$$

Videre er det vist i [15] at det finnes en B slik at:

$$B^T S_{yw} B = I \text{ og } B^T S_{yb} B = D, \tag{1.3}$$

der D er en diagonalmatrise. Da kan man skrive om 1.2 ved hjelp av 1.3 til:

$$S_{xw}^{-1} S_{xb} C = CD, \tag{1.4}$$

der $C = AB$ er en $m \times l$ matrise. Dette ser vi er et egenvektor - egenverdi problem og vi lar søylene i C være egenvektorene tilhørende de l største egenverdiene til $S_{xw}^{-1} S_{xb}$. Siden S_{xb} har rang $M-1$ har $S_{xw}^{-1} S_{xb}$ rang $M-1$ og $M-1$ ikke-null egenverdier og vi lar $l \leq M-1$. Hvis vi setter $\hat{y} = B^T y$ og $y = A^T X$, får vi $\hat{y} = B^T A^T x = C^T x$.

$$\begin{aligned} \text{Da } J_{\hat{y}} &= trasen\{S_{\hat{y}w}^{-1} S_{\hat{y}b}\} \\ &= trasen\{(B^T S_{yw} B)^{-1} (B^T S_{yb} B)\} \\ &= trasen\{B^{-1} S_{yw}^{-1} S_{yb} B\} \\ &= trasen\{S_{yw}^{-1} S_{yb} B^{-1} B\} = J_y \end{aligned}$$

ser vi at vi ikke mister noe av klasseseparerbarheten ved å gå fra y til $\hat{y}$. Vi har videre at

$$J_x = trasen\{S_{xw}^{-1} S_{xb}\} = \lambda_1 + \lambda_2 + \ldots + \lambda_{M+1} + 0$$

og

$$J_{\hat{y}} = trasen\{(C^T S_{xw} C)^{-1} (C^T S_{xb} C)\} \tag{1.5}$$

Ved å skrive om 1.4 til

$$C^T S_{xb} C = C^T S_{xw} CD$$

og sette den inn i 1.5 får vi:

$$J_{\hat{y}} = trasen\{D\} = \lambda_1 + \lambda_2 + \ldots + \lambda_{M+1} = J_x$$

Dermed har vi vist at klasseseparerbarheten for $\hat{y}$ er like stor som for x målt med klasseseparasjonsmålet J, mens antall egenskaper er redusert fra m til l, der $l \leq M-1$, og C er den optimale transformasjonsmatrisen med hensyn på J.

1.4.3 Fouriertransform som egenskapsuttrekning

Når man ser på egenskapene som en respons-funksjon i det elektromagnetiske spekteret, er det nærliggende å tenke at fourierkoeffisientene kan være en naturlig representasjon av kurven. Dette er det flere som har benyttet seg av. I tillegg er det brukt sinus- og cosinustransform som basis for egenskapsuttrekning. (For eksempel Marius Ingjer sin hovedoppgave "Hyperspectral image classification. Feature extraction using orthogonal frequency transformations",IfI, UiO, juni 2005.)

Kapittel 2

Kvadratisk optimering og indrepunktsmetoden

2.1 Optimering

Optimering består i å finne en optimal verdi på en funksjon innenfor et definert område. Dette skriver vi gjerne slik:
Maksimer $f(x)$
Forutsatt at $x \in D$.
Hvis $f(x)$ er en lineær funksjon og D kan beskrives ved affine funksjoner kaller vi det lineær optimering eller lineær programmering (LP). Disse programmene kan vi sette opp slik:

$$\begin{aligned} \text{Maksimer} \quad & f(x) = c'x \\ \text{Forutsatt at} \quad & Ax \leq b \\ \text{og} \quad & Bx = d \end{aligned}$$

Hvis $f(x)$, eller objektivfunksjonen som den kalles, er kvadratisk mens begrensningene fortsatt er affine kaller vi det et kvadratisk program (QP). Dette skriver vi slik:

$$\begin{aligned} \text{Minimer} \quad & f(x) = x'Qx + c'x + d \\ \text{Forutsatt at} \quad & Ax \leq b \\ \text{og} \quad & Bx = d \end{aligned}$$

I dette problemet kreves det gjerne at Q skal være en (symmetrisk) positiv semidefinit matrise.

Hvis både objektivfunksjonen og begrensningene er kvadratiske, kaller vi det et kvadratisk begrenset, kvadratisk program (KBKP). Dette kan se

slik ut:

$$\begin{aligned} \text{Minimer} \quad & f(x) = x'Qx + c'x + d \\ \text{Forutsatt at} \quad & x'Px + q'x + r \leq 0 \\ \text{og} \quad & Bx = d \end{aligned}$$

I KBKP minimerer vi en (konveks) kvadratisk funksjon over snittet av ellipsoider da det igjen kreves at både P og Q er positiv semidefinite matriser.

I denne oppgaven skal vi se på KP problemer nærmere bestemt "Support Vector" problemet og metoder for å løse dette.

2.2 Lagrange relaksasjon

En metode for å relaksere KP problemer er Lagrange relaksasjon. Ideen med Lagrange relaksasjon er å gjøre problemet enklere å løse ved å sette begrensningene inn i objektivfunksjonen. For problemet:

$$\begin{aligned} \text{Minimer} \quad & f(x) && (2.1) \\ \text{Forutsatt at} \quad & g_i(x) <= 0,\ i = 1 \ldots m && (2.2) \\ \text{og} \quad & h_i(x) = 0,\ i = 1 \ldots r && (2.3) \end{aligned}$$

blir Lagranges relakserte problem:

$$\begin{aligned} L(x, \lambda, \nu) \quad = \quad & f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{r} \nu_j h_j(x) \\ \text{med} \quad & \lambda_i \geq 0,\ i = 1 \ldots m \\ \text{og} \quad & \nu_j \in \Re,\ j = 1 \ldots r \end{aligned}$$

hvor λ og ν kalles Lagrange multiplikatorer.

2.2.1 Indikator funksjonen

En tolkning av Lagrange relaksasjon er at den er en lineær approksimasjon på indikatorfunksjonen [1]. Den er definert som

$$\text{minimer } I(x) = f(x) + \sum_{i=1}^{m} I_-(g_i(x)) + \sum_{i=1}^{r} I_0(h_i(x))$$

Hvor I_- og I_0 er indikator funksjonene:

$$\begin{aligned} I_-(u) = \quad & 0, \quad u \leq 0 \\ & \infty, \quad u > 0 \\ I_0(u) = \quad & -\infty, \quad u < 0 \\ & 0, \quad u = 0, \\ & \infty, \quad u > 0 \end{aligned}$$

Denne funksjonen er den samme som $f(x)$ når x tilfredsstiller begrensningene. Når de ikke er tilfredsstilt blir den ∞. Lagrange funksjonen er en ikke så veldig god approksimasjon på indikator funksjonen, men den vil være en underestimator på den siden $L(x, \lambda, \nu) \leq I(x)$.

2.2.2 Nedre grense på optimal verdi

Vi har at Lagrange relaksasjon gir en nedre grense for den optimale verdien til problemet i 2.1.

For en hver x^* som oppfyller begrensningene 2.2 og 2.3 har vi:

$$\sum_{i=1}^{m} \lambda_i(g_i(x^*)) + \sum_{i=1}^{r} \nu_j(h_j(x^*)) \leq 0$$

siden $g_i(x^*) \leq 0$, $h_j(x^*) = 0$ og $\lambda \geq 0$. Dette betyr at $L(x^*, \lambda, \nu) \leq f(x^*)$. Så når man antar at x er innen for begrensningene vil $L(x, \lambda, \nu)$ gi en nedre grense på optimeringsproblemet.

2.2.3 Lagrange duale problem

For hvert par (λ, ν) med $\lambda \geq 0$, har vi sett at Lagrange funksjonen gir oss en nedre grense for den optimale verdien til det opprinnelige eller primale problemet. Vi ønsker så å finne den beste nedre grensen som kan oppnås og definerer funksjonen:

$$g(\lambda, \nu) = inf_{x \in D} L(x, \lambda, \nu)$$

Da får vi det duale problemet:

$$\begin{aligned} \text{Maksimer} \quad & g(\lambda, \nu) \\ \text{Forutsatt at} \quad & \lambda \geq 0 \end{aligned}$$

Dette er et konvekst optimeringsproblem siden objektivfunksjonen er konkav og begrensningene er konvekse.

2.2.4 Karush-Kuhn-Tucker optimalitetskrav

La x^* være et lokalt minimum til problemet:

$$\begin{aligned} \text{minimer} \quad & f(x) \\ \text{forutsatt at} \quad & h_i(x) = 0 \quad i = 1 \dots m \\ & g_j(x) \leq 0 \quad j = 1 \dots r \end{aligned}$$

hvor f, h_i og g_j er kontinuerlige deriverbare funksjoner fra $\Re^n$ til $\Re$, og anta at x^* er et regulært punkt som vil si at $\nabla h_i(x^*), i = 1, \dots, m$ og $\nabla g_j(x^*), j \in J(x^*)$ er lineært uavhengige. Da eksisterer det unike Lagrange multiplikatorvektorer $\lambda^* = (\lambda_1^*, \dots, \lambda_m^*),\ \mu^* = \mu_1^*, \dots, \mu_r^*)$ slik at

$$\begin{aligned} &\nabla_x L(\mathbf{x}^*, \lambda^*, \mu^*) = 0 \\ &\mu_j^* \geq 0,\ j = 1, \dots, r, \\ &\mu_j^* = 0,\ \forall j \notin J(\mathbf{x}^*), \end{aligned}$$

hvor $J(\mathbf{x}^*)$ er settet av aktive begrensninger i $\mathbf{x}^*$.

2.3 Indrepunktsmetoden anvendt på lineært problem

Lineære problemer på formen

$$\begin{aligned} \text{maksimer} \quad & c^T x \\ \text{forutsatt at} \quad & Ax + w = b \\ & x \geq 0 \end{aligned}$$

kan løses effektivt ved hjelp av simpleks-algoritmen. Simpleks algoritmen vil begynne i et hjørne av den tillatte mengden som er et polyeder definert av kravene $Ax \leq b$.

Et alternativ til simpleksalgoritmen er indrepunktsmetoden. Som navnet tilsier så vil den bevege seg i det indre av den tillatte mengden i motsetning til simpleksalgoritmen som beveger seg fra hjørne til hjørne.

Først vil vi introdusere den logaritmiske barriere-funksjonen ved å erstatte ikkenegativitetskravet for x med den logaritmiske barrieren $\log x$ og barriere parameteren μ. Da får vi problemet:

$$\begin{aligned} \text{maksimer} \quad & c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i \\ \text{forutsatt at} \quad & Ax + w = b \end{aligned}$$

Elementene $\mu \sum_j \log x_j + \mu \sum_i \log w_i$ i objektiv funksjonen vil virke som en barriere for x og w mot 0 siden $\lim_{x \to 0} \log x = -\infty$. Dette er egentlig en

famile av problemer etter hvilken verdi μ har. Hvis μ er stor vil man havne langt fra hjørnene i polyederet, mens hvis man gradvis lar $\mu \to 0$ vil man følge en vei fra det indre og mot ett av hjørnene.

For å løse barriere problemet vil vi benytte oss av Lagrange teorien. Vi kan innføre Lagrange's multiplikator og flytte forutsetningene opp i objektivfunksjonen til barriere problemet. Da får vi Lagrange-funksjonen:

$$L(x, w, y) = c^T x + \mu \sum_j \log x_j + \mu \sum_i \log w_i + y^T(b - Ax - w)$$

I følge Karush-Kuhn-Tucker finner vi de førsteordens optimalitetsbetingelsene ved å deriverer m.h.p. variablene x_j, w_i og y og sette dem lik 0. Dette gir:

$$\begin{aligned}
\frac{\delta L(x,w,y)}{\delta x_j} &= c_j + \mu\frac{1}{x_j} - \sum_{i=1}^{m} y_i a_{ij} = 0, j = 1 \dots n \\
\frac{\delta L(x,w,y)}{\delta w_i} &= \mu\frac{1}{w_i} - y_i = 0, i = 1 \dots m \\
\frac{\delta L(x,w,y)}{\delta y_i} &= b_i - \sum_{j=1}^{n} a_{ij} x_j - w_i = 0, i = 1 \dots m
\end{aligned}$$

I matriseform blir dette:

$$\begin{aligned}
A^T y - \mu X^{-1} e &= c \\
y &= \mu W^{-1} e \\
Ax + w &= b
\end{aligned}$$

Her er X diagonalmatrisen med x på diagonalen. Det samme gjelder for W. e er en vektor av enere med tilpasset lengde. Ved å sette $z = \mu X^{-1} e$ kan vi redefinere likningene til en primal-dual symmetrisk form. Da får vi likningsystemet:

$$\begin{aligned}
Ax + w &= b \\
A^T y - z &= c \\
z &= \mu X^{-1} e \\
y &= \mu W^{-1} e
\end{aligned}$$

Ved å innføre samme notasjon for Z og Y som for X og W får vi:

$$\begin{aligned}
Ax + w &= b \\
A^T y - z &= c \\
XZe &= \mu e \\
WYe &= \mu e
\end{aligned}$$

De to siste likningene tilsvarer komplementær slakk kravet. Det betyr at når disse er 0 har vi en optimal løsning. Dette benytter vi oss av i algoritmen. Ved å la μ gå mot 0 vil vi gå fra et indre punkt til et punkt nærmere randen på det konvekse problemet derfor har det fått navnet indrepunktsmetoden.

2.4 Indrepunktsmetoden anvendt på kvadratisk problem

V ønsker å anvende teorien for indrepunktsmetoden på det kvadratiske problemet med lineære begrensninger:

$$\begin{aligned} \text{minimer} \quad & c^T x + \frac{1}{2} x^T Q x \\ \text{forutsatt at} \quad & Ax - w = b \\ & w, x \geq 0 \end{aligned}$$

Ved samme anaalogi som for det lineære problemet får vi først barriere problemet:

$$\begin{aligned} \text{maksimer} \quad & c^T x + \frac{1}{2} x^T Q x - \mu \sum_j \log x_j - \mu \sum_i \log w_i \\ \text{forutsatt at} \quad & Ax - w = b \end{aligned}$$

Vi flytter sa forutsetningene opp i objektivfunksjonen ved hjelp Lagrange's multiplikator og får lagrange-funksjonen:

$$L(x, w, y) = c^T x + \frac{1}{2} x^T Q x - \mu \sum_j \log x_j - \mu \sum_i \log w_i + y^T (b - Ax + w)$$

Første ordens optimalitetsbetingelser blir da:

$$\begin{aligned} \frac{\delta L(x,w,y)}{\delta x_j} &= c_j + \sum_{i=1}^{m} Q_{ij} x_j - \mu \frac{1}{x_j} - \sum_{i=1}^{m} y_i a_{ij} = 0, j = 1 \ldots n \\ \frac{\delta L(x,w,y)}{\delta w_i} &= -\mu \frac{1}{w_i} + y_i = 0, i = 1 \ldots m \\ \frac{\delta L(x,w,y)}{\delta y_i} &= b_i - \sum_{j=1}^{n} a_{ij} x_j + w_i = 0, i = 1 \ldots m \end{aligned}$$

Og vi får den Primal-Duale matriseformen:

$$\begin{aligned} Ax - w &= b \\ A^T y + z - Qx &= c \\ XZe &= \mu e \\ WYe &= \mu e \end{aligned}$$

2.4.1 Newtons metode

For å løse disse problemene bruker vi Newtons metode. Vi vil utlede Newtons metode på det kvadratiske problemet. Newtons metode for det lineære problemet vil følge den samme utledningen med den forskjellen at den mangler det kvadratiske leddet.

Først bytter vi ut (x, w, y, z) med $(x + \Delta x, w + \Delta w, y + \Delta y, z + \Delta z)$ og ordner likningene:

$$\begin{aligned} A\Delta x - \Delta w &= b - Ax + w := \sigma \\ A^T\Delta y + \Delta z - Q\Delta x &= c - A^T y - z + Qx := \rho \\ X\Delta z + Z\Delta x + \Delta X\Delta Ze &= \mu e - XZe \\ W\Delta y + Y\Delta w + \Delta Y\Delta We &= \mu e - YWe \end{aligned}$$

Gitt (x, w, y, z) ønsker vi å finne skritt retning $(\Delta x, \Delta w, \Delta y, \Delta z)$. Da må vi løse settet med likninger. Siden $\Delta X\Delta Ze$ og $\Delta Y\Delta We$ er ulineære byr de på problemer og vi dropper dem i likningssystemet. I tillegg setter vi høyresidene til de to første likningen lik h.h.v. σ og ρ for å forenkle uttrykkene senere.

Da får vi likningssystemet:

$$\begin{aligned} A\Delta x - \Delta w &= \sigma \\ A^T\Delta y + \Delta z - Q\Delta x &= \rho \\ X\Delta z + Z\Delta x &= \mu e - XZe \\ W\Delta y + Y\Delta w &= \mu e - YWe \end{aligned}$$

De to siste likningene gir:

$$\begin{aligned} \Delta z &= X^{-1}(\mu e - XZe - Z\Delta x) \\ \Delta w &= Y^{-1}(\mu e - YWe - W\Delta y) \end{aligned}$$

Eliminerer så Δz og Δw i de to første likningene:

$$\begin{aligned} & A\Delta x - Y^{-1}(\mu e - YWe - W\Delta y) = \sigma \\ \Rightarrow\ & A\Delta x + Y^{-1}W\Delta y = \sigma + \mu Y^{-1}e - w \\ \text{og}\ & A^T\Delta y + X^{-1}(\mu e - XZe - Z\Delta x) - Q\Delta x = \rho \\ \Rightarrow\ & A^T\Delta y - (X^{-1}Z - Q)\Delta x = \rho - \mu X^{-1}e + z \end{aligned}$$

Substituerer inn definisjonen av σ og ρ og setter opp systemet i matrise form:

$$\begin{bmatrix} -(X^{-1}Z + Q) & A^T \\ A & Y^{-1}W \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} c - A^T y - \mu X^{-1}e + Qx \\ b - Ax + \mu Y^{-1}e \end{bmatrix}$$

2.4.2 Barriere parameter

For hver iterasjon oppdaterer vi x y w og z. I tillegg må vi redusere barriere parameteren μ. Dette for at vi skal bevege oss fra et punkt i det indre til et punkt på randen.

Fra likning 2.4 og 2.4 ser vi at barriere parameteren er proporsjonal med den komplementære slakken $x_i z_i$ og $w_j y_j$. Gitt et punkt (x_i, z_i, w_j, y_j) finner vi $\mu = x_i z_j$ eller $\mu = w_i y_j$ for en i eller j. Ettersom vi nærmer oss en optimal verdi vil vi redusere betydningen av barriere leddet ved å redusere μ. I stedet for å velge et par x_i, z_i eller w_j, y_j, velger vi å bruke et gjennomsnitt av alle: $\frac{z^T x + y^T w}{n+m}$. Denne verdien reduserer vi med et tall, δ mellom 0 og 1. For eksempel foreslår Vanderbei i [16] $\frac{1}{10}$. Da får vi $\mu = \delta \frac{z^T x + y^T w}{n+m}$

2.4.3 Skritt lengde

Når vi oppdaterer x, y, z, w må vi sørge for at de nye verdiene fortsatt er større enn 0. Dvs. $x_j + \theta \Delta x_j > 0$. Dette gir at $\theta > -\frac{x_j}{\Delta x_j}$. eller $\frac{1}{\theta} < -\frac{\Delta x_j}{x_j}$ for alle x_j, z_j, w_i, y_i. Vi finner den minste θ og ganger den med et tall r litt mindre enn 1, f.eks. 0,9. I tillegg ønsker vi ikke at θ skal være større enn 1 og velger da det minste av dette og 1:

$$\theta = \min(1, r(\max_{i,j}\{-\frac{\Delta x_j}{x_j}, -\frac{\Delta w_i}{w_i}, -\frac{\Delta y_i}{y_i}, -\frac{\Delta z_j}{z_j}\})^{-1})$$

Da kan vi gjøre oppdateringene:

$$\tilde{x} = x + \theta \Delta x, \quad \tilde{y} = y + \theta \Delta y,$$
$$\tilde{w} = w + \theta \Delta w, \quad \tilde{z} = z + \theta \Delta z$$

2.4.4 Stoppkriterier

Vi vet at vi har funnet en optimal løsning når den er primalt tillatt, dualt tillatt og den komplementære slakken er null. For det lineære problemet har vi:

Primalt tillatt:	$\rho = b - Ax - w = 0$
Dualt tillatt:	$\sigma = c - A^T y + z = 0$
Komplementær slakk:	$\gamma = z^T x + y^T w = 0$

For det kvadratiske problemet blir dette:

Primalt tillatt:	$\rho = b - Ax + w = 0$
Dualt tillatt:	$\sigma = c - A^T y - z + Qx = 0$
Komplementær slakk:	som for det lineære

Vi velger et lite tall $\epsilon > 0$ og krever at $||\rho||_1 < \epsilon$, $||\sigma||_1 < \epsilon$, og $|\gamma| < \epsilon$ for at løsningen skal være optimal. I tillegg trenger vi en grense for hvor stor x kan være før vi sier at problemet er ubegrenset. Vi sier at problemet er primalt ubegrenset hvis $||x||_\infty > M$, der M er et valgt stort tall, og at problemet er dualt ubegrenset hvis $||y||_\infty > M$.

Kapittel 3

Klassifikasjonsteori

3.1 Klassifikasjon

Anta du har et sett med punkter x_i som er fordelt på to klasser, + og − representert ved hhv. $y_i = +1$ og $y_i = -1$. Ved hjelp av et treningssett der klassen er kjent fra før ønsker man å trene opp en funksjon som kan skille disse klassene fra hverandre. Vi kaller denne gjerne for en diskriminantfunksjon.

3.2 Valg av diskriminantfunksjon

Det er mange funksjoner, eller familier av funksjoner, man kan velge mellom som diskriminantfunksjon. Det er viktig at man velger en funksjon med riktig kompleksitetsgrad i forhold til treningsdataene. Hvis treningsdataene har en komplisert fordeling trenger man en komplisert funksjon. Samtidig må ikke funksjonen bli for komplisert for å unngå at den blir overtilpasset treningsettet.

Ofte har man et begrenset antall punkter i treningssettet. Det gjør at man kan få tilfeldige tendenser i treningssettet som ikke finnes i et testsett dvs. i den egentlige fordelingen av dataene. Hvis funksjonen blir for kompleks kan den tilpasse seg disse tendensene og gi feilklassifikasjoner i testsettet. Dette kan man unngå ved å begrense kompleksiteten på diskriminantfunksjonen.

Mange algoritmer er bygget på lineære diskriminantfunksjoner, hyperplan, som for eksempel “Support Vector Machine”. Denne type algoritmer kan benytte seg av såkalte kjernefunksjoner.

3.3 Kjernefunksjoner

Siden det ikke alltid er mulig å skille to klasser av egenskapsvektorer $x \in \Re^n$ med et hyperplan, ønsker vi å finne en transformasjon $\Phi(x)$ fra $\Re^n$ og inn i

et høyere dimensjonalt rom $\Re^m$ slik at dette er mulig.

Siden egenskapsvektorene opptrer i prikkprodukter i optimeringsproblemet kan man bytte ut $\langle x_i, x_j \rangle$ med $\langle \Phi(x_i), \Phi(x_j) \rangle$. Da kan man på samme måte løse optimeringsproblemet (for y_i) med de transformerte egenskapsvektorene og finne skillende hyperplan i dette nye rommet. Så bruker man den samme transformasjonen i diskriminantfunksjonen.

$$f(x) = sign(\sum_{i=1}^{l} \alpha_i y_i \langle \Phi(x_i), \Phi(x) \rangle + b).$$

Definisjon 1 (Kjernefunksjon) *En funksjon $K(x, y)$ kalles en kjernefunksjon hvis*

$$\int K(x, y) g(x) g(y) dx dy \geq 0$$

for alle funksjoner $g \in L_2$ (der L_2 er rommet av alle 2-ganger deriverbare funksjoner).

Hvis man da antar at det finnes en kjerne funksjon K slik at:

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

så kan vi bytte ut $\langle x_i, x_j \rangle$ med $K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$ i programmet. Dette har Mercer vist at finnes så lenge matrisen K oppfyller visse krav.

Teorem 1 (Mercer) *La X være et endelig innput rom og $K(x, z)$ er en symmetrisk funksjon på X. Da har vi at*

$$K(x, y) = \langle \Phi(x), \Phi(y) \rangle$$

for en transformasjon $\Phi : X \to H$, hvor H er et (mulig) høyere-dimensjonalt indreproduktrom.

Da er $K(x, y)$ en kjernefunksjon hvis og bare hvis matrisen

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

er positiv semidefinit (har ikkenegative egenverdier). Med andre ord er K en Gram matrise hvor

$$x^T K x \geq 0$$

Se "Support Vector Machines", side 33, av Christianino og Shawe-Taylor [2].

Ved hjelp av kjernefunksjoner unngår man å utføre transformasjonen av egenskapsvektorene inn i et høyere dimensjonalt rom for å ta indreproduktet der. Det er heller ikke nødvendig å kjenne det eksakte uttrykket for transformasjonen $\Phi(x_j)$ bare funksjonen oppfyller visse krav gitt av Mercer.

Det er brukt flere forskjellige kjernefunksjoner. Eksempler på dette er:

Polynomisk kjerne:

$$K(x_i, x_j) = \langle x_i, x_j \rangle^d \text{ eller } (\langle x_i, x_j \rangle + 1)^d$$

Gaussisk kjerne:

$$K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$$

Sigmoid kjerne:

$$K(x_i, x_j) = \tanh(\kappa \langle x_i, x_j \rangle + \Theta).$$

Man kan lett lage nye kjernefunksjoner ved å kombinere to eller flere. Her er for eksempel en kombinasjon av en sigmoid og en gaussisk kjerne:

Sigmoid-gaussisk kjerne:

$$K(x_i, x_j) = -\tanh(\kappa \|x_i - x_j\|^2 + \Theta).$$

Man kan si at kjernen representerer ett sett med funksjoner. Dvs. for hver parameterinnstilling får vi en ny funksjon der parameteren bestemmer kompleksiteten av funksjonen. Det er viktig at kompleksiteten til funksjonen er passe til problemet og dermed at parameteren(e) er riktig innstilt. Vapnik og Chervonenkis har utviklet en teori rundt dette som kalles VC-teori.

3.4 VC-Teori

Vapnick-chervonenkis (VC) teori sier at det er viktig å begrense antall mulige funksjoner som brukes for klassifikasjon/kjernetransformasjoner. Vi må ikke overtilpasse funksjonen for å få 100% riktig klassifikasjon. Vi må beholde en viss generalitet. Funksjonen må ha en riktig kapasitet i forhold til antall tilgjengelige treningsdata. Den mest kjente kapasitets målet i VC-teori er VC dimensjonen. Den er definert som det største antall punkter som funksjonen kan 'spre'. En funksjon kan 'spre' m punkter hvis den kan skille dem på alle mulige måter i to grupper. For eksempel kan et plan skille tre punkter i to grupper på alle mulige måter, men 4 punkter kan det ikke skille på denne måten. Se side 54-58 i [2] og side 9-10 (og 141) i [14].

3.5 Trening og validering

For å være sikker på at funksjonen får den riktige kompleksiteten kan man bruke et valideringssett. Man deler treningsettet i to. Den ene delen bruker man til å trene opp funksjonen og den andre til å teste eller validere treningen. Man gjentar treningen for forskjellige parameterinnstillinger og velger de verdiene som ga best klassifikasjon på valideringssettet. Ved bruk av et valideringssett unngår man å overtilpasse algoritmen testsettet.

3.6 Klassifikasjon av testsett

Man klassifiserer så testsettet som er disjunkt fra trening og valideringssettene.

Ettersom man ikke nødvendigvis er ute etter å finne den beste mulige klassifikasjonene av dette testsettet, men heller en gjennomsnittlig prestasjonsevne av algoritmen, kan det være naturlig å bruke flere trenigssett og så rapportere et gjennomsnitt og standardavvik av klassifikasjonene.

3.7 En enkel klassifikasjonsmetode: senter problemet

Anta du har et sett med punkter x_i som er fordelt på to klasser, + og − representert ved hhv. $y_i = +1$ og $y_i = -1$. En enkel måte å skille disse to klassene fra hverandre på er ved å først finne senteret til hver av klassene og så finne ut hvilket senter et nytt punkt er nærmest. Dette kan vi gjøre ved hjelp av et indreprodukt. Sentrene til klassene er gitt ved:

$$c_+ = 1/m_+ \sum_{i|y_i=+1} x_i \tag{3.1}$$

og

$$c_- = 1/m_- \sum_{i|y_i=-1} x_i \tag{3.2}$$

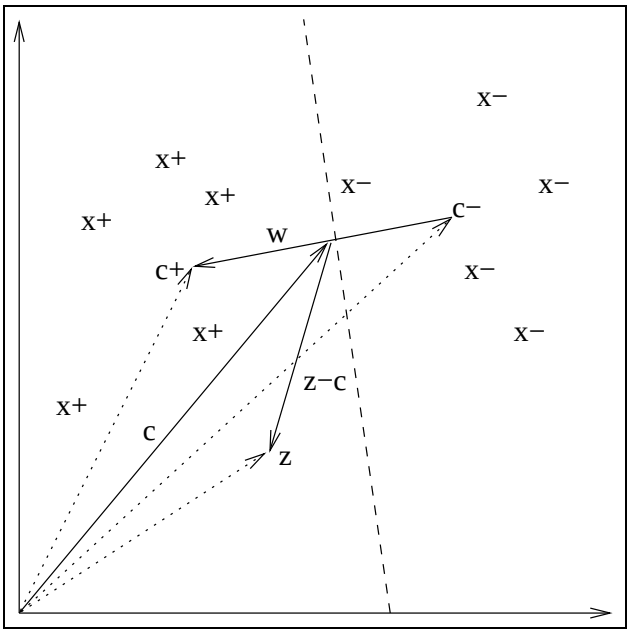
Så finner vi retningen fra c_- til c_+ ved

$$w = c_+ - c_-$$

Midtpunktet på denne linjen blir $c = (c_+ + c_-)/2$. Hvis vinkelen mellom $x - c$ og w er mindre enn $\pi/2$, tilhører x klassen + og hvis vinkelen er større enn $\pi/2$ klassen −. Dette kan uttrykkes med indreprodukt ved:

$$\begin{aligned} f(x) &= sign(\langle x - c, w\rangle) && (3.3)\\ &= sign(\langle x - (c_+ + c_-)/2, c_+ - c_-\rangle) && (3.4)\\ &= sign(\langle x, c_+\rangle - \langle x, c_-\rangle + b) && (3.5)\\ \text{hvor} & && (3.6)\\ b &= \frac{1}{2}(-\langle c_+, c_+\rangle + \langle c_-, c_-\rangle - \langle c_-, c_+\rangle + \langle c_+, c_-\rangle) && (3.7)\\ &= \frac{1}{2}(||c_-||^2 - ||c_+||^2 \text{ der } ||c|| = \sqrt{\langle c, c\rangle} && (3.8) \end{aligned}$$

I figur 3.1 kan vi se at en klassifikasjon av et nytt punkt z blir klassifisert til klasse $x+$.



Figur 3.1: Et enkelt klassifikasjons problem. Et testpunkt z blir klassifisert til den klassen med nærmest senterpunkt. Dette kan gjøres ved å beregne indreproduktet mellom $z - c$ og w. Her er $c = (c_+ + c_-)/2$ og $w = c_+ - c_-$. Vi kan se at $\langle z - c, w \rangle > 0$ siden vinkelen mellom w og $z - c$ er mindre enn $\pi/2$ og dermed blir z klassifisert til klassen med $x+$.

Ved å skrive om $f(x)$ og b til å uttrykkes ved x kan vi innføre kjernen $K(x,x)$ i isteden for indreproduktet $\langle x,x\rangle$. Vi setter inn for c_+ og c_- og får:

$$f(x) = sign(1/m_+ \sum_{i|y_i=+1} \langle x,x_i\rangle - 1/m_- \sum_{i|y_i=-1} \langle x,x_i\rangle + b) \quad (3.9)$$

$$= sign(1/m_+ \sum_{i|y_i=+1} K(x,x_i) - 1/m_- \sum_{i|y_i=-1} K(x,x_i) + b) \quad (3.10)$$

$$b = \frac{1}{2}(1/m_-^2 \sum_{i,j|y_i=y_j=-1} K(x_i,x_j) - 1/m_+^2 \sum_{i,j|y_i=y_j=+1} K(x_i,x_j)) \quad (3.11)$$

Neste steg i utviklingen av metoden kan være å eliminere punkter i beregningen av c_- og c_+. Dette kan gjøres ved for eksempel å fjerne uteliggere. Man kan også la punktene nærmest skille mellom klassene få større betydning. Dette kan gjøres ved å bytte ut $1/m_+$ og $1/m_-$ med en α_i for hvert punkt. Man kan da starte med $\alpha_i = 1/m_+$ og $\alpha_i = 1/m_+$ for hhv $y_i = +1$ og $y_i = -1$ og så minke α_i for punkter som ligger lengst fra middelpunktet, og øke α_i for de x_i som ligger nærme den andre klassens middelpunkt. Dette kan gjentas iterativt til middelpunktene nærmer seg grensen mellom fordelingene. Dette kan gi en nøyaktigere klassifikasjon i grenseområdene mellom klassene.

En annen måte å forbedre metoden på kan være å justere b slik at man minimere antall feilklassifiserte punkter. Dette tilsvarer å parallellforskyve hyperplanet som skiller klassene og på den måten kompansere for at klassene kan ha forskjellige varianser.

Kapittel 4

Klassifikasjon med Support Vector Machine

4.1 Hard margin problemet

SVM problemet vi så på i seksjon 1.2.3 kalles hard margin problemet. Vi fant ut at vi kunne beskrive det slik:

$$\begin{array}{ll} \text{Minimer} & \frac{1}{2}\|w\|^2 \\ \text{Forutsatt at} & y_i(\langle w, x_i\rangle + b) \geq 1,\ i = 1, \ldots, m \end{array} \tag{4.1}$$

med bestemmelses-funksjonen:

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

.

Vi kaller det hard margin problemet siden det ikke tillater at punkter havner på feil side av marginen. Klassene må være lineært separerbare for at problemet skal ha noen løsning. Isteden for å løse dette problemet direkte er det vanlig å omforme det til det duale problemet ved hjelp av Lagrange-funksjonen. Vi kaller det Lagrange relaksasjon av problemet.

4.1.1 Lagrange relaksasjon

Lagrange-funksjonen får vi ved å flytte kravene $y_i(\langle w, x_i\rangle + b) \geq 1$ opp i objektivfunksjonen og introdusere Lagrange-multiplikatorene $\alpha_i \geq 0$ som gir:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i(y_i(\langle w, x_i\rangle + b) - 1) \tag{4.2}$$

Denne funksjonen må minimeres mhp. de primale variablene w og b og maksimeres mhp. de duale variablene α_i. Dette tilsvarer å finne et sadelpunkt.

Hvis vi tenker oss at w og b er gitt, og så ønsker å maksimere lagrange-funksjonen 4.2, må vi øke α_i for de i der $y_i(\langle w, x_i\rangle + b) - 1 \leq 0$ og sette $\alpha_i = 0$ for de i der $y_i(\langle w, x_i\rangle + b) - 1 > 0$.

Når vi så har satt verdiene på α_i-ene ønsker vi så å minimere funksjonen med hensyn på w og b. Dette får vi til ved å justere w og b slik at de x_i som bryter kravet $y_i(\langle w, x_i\rangle + b) > 1$ og dermed gir et tillegg til objektivfunksjonen kommer på riktig side av marginen igjen. Da vil $y_i(\langle w, x_i\rangle + b) - 1 > 0$ og med $\alpha_i > 0$ vil det minke funksjonsverdien.

Når vi så skal maksimere funksjonen mhp. α_i igjen, vil de $\alpha_i > 0$ som ga et negativt tilskudd til funksjonen bli satt til 0 igjen, mens andre α_i for x_i som nå bryter kravet i 4.1 vil bli øket fra 0.

Dette fører til en iterativ prosess som konvergerer i et sadelpunkt.

Dette sadelpunktet vil gi den w og b som oppfyller kravene i likning 4.1 (om mulig) og samtidig gir størst mulig marginen.

De x_i hvor $\alpha_i > 0$ kaller vi support vektorer eller støtte-vektorer, siden de "støtter" marginen på hver side. Dette kommer fra Karush-Kuhn-Tucker's komplementæritets krav.

4.1.2 Karush-Kuhn-Tucker's komplemetæritetskrav

Karush-Kuhn-Tucker's komplemetæritetskrav sier at i den optimale løsningen av Lagrange-funksjonen vil følgende være oppfylt:

$$\alpha_i[y_i(\langle w, x_i\rangle + b) - 1] = 0,\ i = 1, \ldots, m \tag{4.3}$$

Denne likningen sier at i den optimale løsningen til Lagrange-funksjonen vil $\alpha_i = 0$ for de x_i hvor $y_i(\langle w, x_i\rangle + b) > 1$, og $\alpha_i > 0$ for de x_i hvor $y_i(\langle w, x_i\rangle + b) = 1$, altså på margin-kanten.

I følge Karush-Kuhn-Tucker's optimalitetskriterier finner vi dette sadelpunktet når de partiellderiverte er 0:

$$\frac{\delta}{\delta b} L(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y_i = 0$$

og

$$\frac{\delta}{\delta w} L(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y_i x_i - w = 0$$

som gir oss kravene:

$$\sum_{i=1}^{m} \alpha_i y_i = 0,\ \text{og } w = \sum_{i=1}^{m} \alpha_i y_i x_i.$$

4.1.3 Det dual problemet

Ved å sette disse likningene inn i Lagranges funksjon får vi det duale optimeringsproblemet:

$$\max_{\alpha} \quad W(\alpha) = \sum_{i,j=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{4.4}$$

$$\text{f.a.} \qquad \alpha_i \geq 0 \; i = 1, \ldots, m \tag{4.5}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{4.6}$$

Diskriminantfunksjonen kan da skrives:

$$f(x) = sign(\sum_{i=1}^{m} y_i \alpha_i \langle x, x_i \rangle + b)$$

og b finner man fra de primale begrensningen:

$$b = -\frac{\max_{y_i=-1}(\langle w, x_i \rangle) + \min_{y_i=1}(\langle w, x_i \rangle)}{2} \tag{4.7}$$

4.2 Myk margin: C-SVM problemet

Hvis dataene ikke er lineært separerbare finnes det ingen tillatt løsning av 4.1. I dette tilfellet kan man "myke opp" hyperplanet ved at man legger til et feilledd ϵ_i i kravet $y_i(\langle w, x_i \rangle + b) \geq 1$. I tillegg kan man legge til $C \sum_{i=1}^{m} \epsilon_i$ i objektivfunksjonen slik at disse feilleddene vil være minimert i den optimale løsningen. Dette optimeringsproblemet er kjent som C-SVM problemet i litteraturen:

$$\min \qquad \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{m} \epsilon_i \tag{4.8}$$

$$\text{forutsatt at} \quad y_i(\langle w, x_i \rangle + b) + \epsilon_i \geq 1, \quad i = 1 \ldots m \tag{4.9}$$

$$\epsilon_i \geq 0, \qquad i = 1 \ldots m \tag{4.10}$$

Her er $C > 0$ en parameter som vekter betydningen av punktene som bryter kravet $y_i(\langle w, x_i \rangle + b) \geq 1$. Punktene med $\epsilon_i > 0$ kan være av to typer. De kan være på riktig side av hyperplanet men innenfor marginen eller på feil side av hyperplanet og dermed bli feilklassifisert. Er C stor vil hvert punkt med $\epsilon_i > 0$ ha stor betydning og marginen $1/\|w\|$ vil bli liten. Er C liten kan programmet godta flere punkter på feil side av marginen og marginen kan økes. Dette øker generaliteten til programmet ved at flere punkter er med på å bestemme hyperplanet. Samtidig som man ønsker en viss generalitet, ønsker man heller ikke for mange feilklassifiserte treningspunkter.

4.2.1 Lagrange relaksasjon

For C-SVM problemet blir Lagrange relakserte problem:

$$
\begin{aligned}
L(w,b,\epsilon,\alpha,r) &= \frac{1}{2}\langle w,w\rangle + C\sum_{i=1}^{m}\epsilon_i \\
&\quad -\sum_{i=1}^{m}\alpha_i[y_i(\langle x_i,w\rangle + b) - 1 + \epsilon_i] - \sum_{i=1}^{m} r_i\epsilon_i
\end{aligned}
$$

Det duale er da gitt ved å derivere med hensyn på w, ϵ og b som gir likningene:

$$
\frac{\delta L(w,b,\epsilon,\alpha,r)}{\delta w} = w - \sum_{i=1}^{m} y_i\alpha_i x_i = 0, \tag{4.11}
$$

$$
\frac{\delta L(w,b,\epsilon,\alpha,r)}{\delta \epsilon_i} = C - \alpha_i - r_i = 0, \tag{4.12}
$$

$$
\frac{\delta L(w,b,\epsilon,\alpha,r)}{\delta b} = \sum_{i=1}^{m} y_i\alpha_i = 0. \tag{4.13}
$$

Ved å bruke disse likningene kan vi substituere ut de primale variablene w og b og får det duale problemet i α:

$$
L(w,b,\epsilon,\alpha,r) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle x_i, x_j\rangle.
$$

Dette er likt som hos maksimal margin. I tillegg har vi kravet $C-\alpha_i-r_i = 0$, som sammen med $r_i \geq 0$, gir $\alpha_i \leq C$.

$\epsilon_i \neq 0$ bare når $r_i = 0$ som gir $\alpha_i = C$ fra likning 4.12. Dette gir oss Karush-Kuhn-Tucker's komplementæritets krav:

$$
\alpha_i[y_i(\langle w, x_i\rangle + b) - 1 + \epsilon_i] = 0, \qquad i = 1,\ldots,m \tag{4.14}
$$

$$
\epsilon_i r_i = \epsilon_i(\alpha_i - C) = 0, \qquad i = 1,\ldots,m \tag{4.15}
$$

Dette gir oss tre muligheter for α_i:

$$
\begin{aligned}
\alpha_i = 0 \quad & y_i(\textstyle\sum_{j=1}^{m} y_j\alpha_j\langle x_j,x_i\rangle + b) \geq 1 \text{ og} \quad \epsilon_i = 0 \\
0 < \alpha_i < C \quad & y_i(\textstyle\sum_{j=1}^{m} y_j\alpha_j\langle x_j,x_i\rangle + b) = 1 \text{ og} \quad \epsilon_i = 0 \\
\alpha_i = C \quad & y_i(\textstyle\sum_{j=1}^{m} y_j\alpha_j\langle x_j,x_i\rangle + b) \leq 1 \text{ og} \quad \epsilon_i \geq 0
\end{aligned}
$$

Den første likningen gjelder for punkter som ligger utenfor marginen (på riktig side). De har ingen ting å si for plasseringen av hyperplanet. Den andre likningen gjelder for de punktene som ligger på marginen. Dette er support vektorene. Den siste av de tre likningene gjelder for punkter som ligger innenfor marginen. De har fått et tillegg $\epsilon_i \geq 0$ for at margin kravet skal holde samtidig som α_i har nådd "taket" med C/m. Man sier at marginen

er myk for disse punktene. Siden det bare er noen få av punktene som er med i løsningen kan SVM løse store problemer ved å utnytte denne sparsomheten i løsningen.

Det duale problemet blir da:

$$\max_{\alpha} \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{4.16}$$

$$\text{forutsatt at} \quad C \geq \alpha_i \geq 0 \; i = 1, \ldots, m \tag{4.17}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{4.18}$$

4.2.2 Beregning av b

For å beregne avstanden b bruker vi support vektorene (SV). Dvs. de $x_i \in I := \{i : 0 < \alpha_i < C\}$. Da har vi for hver $x_i \in I$ at:

$$y_i(\sum_{j=1}^{m} y_j \alpha_j \langle x_j, x_i \rangle + b) = 1$$

Tar vi gjennomsnittet over disse får vi en numerisk stabil løsning [11]:

$$b = \frac{1}{|I|} \sum_{i \in I} (y_i - \sum_{j=1}^{m} y_j \alpha_j \langle x_j, x_i \rangle)$$

4.3 Myk margin 2: ν-SVM

Da C ikke er en så intuitiv variabel er det blitt introdusert en annen parameter $\nu \in (0, 1]$ (side 204-211 i [14] og [13]), slik at vi får problemet:

$$\min \quad \tfrac{1}{2}\|w\|^2 - \nu\rho + \tfrac{1}{m}\textstyle\sum_{i=1}^{m} \epsilon_i \tag{4.19}$$

$$\text{forutsatt at} \quad y_i(\langle w, x_i \rangle + b) + \epsilon_i \geq \rho, \quad i = 1 \ldots m \tag{4.20}$$

$$\epsilon_i \geq 0, \quad i = 1 \ldots m \tag{4.21}$$

$$\rho \geq 1 \tag{4.22}$$

$$\tag{4.23}$$

Det tilsvarende duale blir da :

$$\max_{\alpha} \quad W(\alpha) = -\tfrac{1}{2}\textstyle\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{4.24}$$

$$\text{forutsatt at} \quad 0 \leq \alpha_i \leq \tfrac{1}{m} \quad i = 1, \ldots, m \tag{4.25}$$

$$\textstyle\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{4.26}$$

$$\textstyle\sum_{i=1}^{m} \alpha_i \geq \nu \tag{4.27}$$

Fordelen med parameteren ν er at den kan tolkes som en nedre grense på antall support vektorer og en øvre grense på antall punkter på feil side av marginen (margin feil)[11].

Vi får den samme bestemmelsesfunksjonen:

$$f(x) = sign(\sum_{i=1}^{m} \alpha_i y_i \langle x_i, x \rangle + b).$$

4.4 "Proximal" eller regularisert SVM

Glenn Fung og Olvi L. Mangasarian [5] har gjort noen forandringer til C-SVM problemet slik at den kan bli tolket som en regularisert minste kvadraters metode.

De har byttet ut ulikheten i $y_i(\langle w, x_i \rangle + b) + \epsilon_i \geq 1$ til en likhet. Dette medfører at man dytter planene $\langle w, x \rangle + b = \pm 1$ utover fra å danne en margin til å være lineære aproksimasjoner på de to klassene:

$$\begin{aligned} \min \quad & \tfrac{1}{2}\|w\|^2 + \tfrac{1}{2}b^2 + \tfrac{C}{2}\|\epsilon_i\|^2 && (4.28) \\ \text{forutsatt at} \quad & y_i(\langle w, x_i \rangle + b) + \epsilon_i = 1, \quad i = 1 \dots m && (4.29) \\ & && (4.30) \end{aligned}$$

I tillegg unngår de kravet $\epsilon_i \geq 0, i = 1 \dots m$ ved å kvadrere ϵ_i i objektivfunksjonen. Det er da ikke nødvendig med et ikkenegativitetskrav på disse variablene siden hvis en ϵ_i er negativ kan vi minke objektivfunksjonen ved å sette denne til 0 og samtidig tilfredstille det tilhørende kravet $y_i(\langle w, x_i \rangle + b) + \epsilon_i = 1$.

De har også lagt til $1/2b^2$ i objektivfunksjonen. Det viser seg at dette tillegget gir en like god klassifikasjon og i tillegg gir sterk konveksitet i objektivfunksjonen.

Dette problemet kan tolkes som en regularisert minste kvadraters metode til likningssystemet $y_i(\langle w, x_i \rangle + b) + \epsilon_i = 1, i = 1 \dots m$.

Fordelen ved denne måten å sette opp problemet på er at det bare krever løsningen av et likningssett som er mye raskere enn å måtte løse et kvadratisk eller lineært optimeringsprogram. Fung og Mangasarian har også vist at det gir gode test sett resultater på noen datasett. De har også foreslått noen forbedringer i "Multicategori Proximal SVM" artikkelen [4] med bl. annet en vekting av klasser med forskjellig antall treningspunkter og en "Newton refinement" hvor de paralellforskyver hyperplannet for å minimere klassifikasjonsfeil.

4.5 Indrepunktsmetoden anvendt på det duale av SVM-problemet

Med indrepunktsmetoden har vi utviklet et program som løser problemer på formen:

$$\begin{aligned} \min \quad & c^T x + \frac{1}{2} x^T Q x \\ \text{subject to} \quad & Ax \geq b \\ & w, x \geq 0 \end{aligned}$$

Jeg har kalt dette programmet pathQuad programmet. For å kunne bruke dette programmet på de duale SVM-problemene må disse omformes til denne formen.

SVM-hard margin problemet, likning 4.4 til 4.6, kan omskrives på følgende måte: For objektivfunksjonen velger vi $x = \alpha$, $Q_{i,j} = y_i y_j K(x_i, x_j)$ og $c = [-1... - 1]^T$. For å få kravet $\sum_{i=1}^{m} \alpha_i y_i = 0$ over på formen $Ax \geq b$ bruker vi en vanlig omskrivning fra optimeringsliteraturen. Vi erstatte likheten med to ulikheter, $\sum_{i=1}^{m} \alpha_i y_i \geq 0$ og $-\sum_{i=1}^{m} \alpha_i y_i \geq 0$. På matriseform blir dette $A = \begin{bmatrix} y \\ -y \end{bmatrix}$ og $b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, der y er en radvektor av lengde m.

For C-SVM problemet, likning 4.16 til 4.18, har vi i tillegg at $\alpha_i \leq C$. Dette kravet kan vi inkorporere i systemet over ved å utvide A med den negative identitetsmatrisen $-I$ og b med $-C$ m ganger. Da får vi:

$$c = [-1... - 1]^T,\ A = \begin{bmatrix} y \\ -y \\ -I \end{bmatrix},\ b = \begin{bmatrix} 0 \\ 0 \\ -C \\ \vdots \\ -C \end{bmatrix} \text{ og } Q_{ij} = y_i y_j K(x_i, x_j) \text{ som før.}$$

For ν-SVM problemet, likning 4.24 til 4.27, får vi $c = [0 \dots 0]$, og for A og b kan vi gjøre tilsvarende utvidelser som for C-SVM problemet, men med $-\frac{1}{m}$ i steden for C i b. For kravet $\sum_{i=1}^{m} \alpha_i \geq \nu$ får vi et tillegg på en radvektor av 1'ere i A og et tillegg på ν i b. Oppsummert blir dette:

$$c = [0 \dots 0],\ A = \begin{bmatrix} y \\ -y \\ -I \\ 1 \dots 1 \end{bmatrix},\ b = \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{m} \\ \vdots \\ -\frac{1}{m} \\ \nu \end{bmatrix} \text{ og } Q \text{ er som før.}$$

Foruten hard-margin, C-SVM og ν-SVM problemet har jeg benyttet meg av en variant jeg har kalt blandingsvarianten som er gitt ved:

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathrm{K}(x_i, x_j) \tag{4.31}$$

$$\text{forutsatt at} \quad \alpha_i \geq 0,\ i = 1, \ldots, m \tag{4.32}$$

$$\text{og} \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{4.33}$$

som gir:

$$c = [0 \ldots 0],\ A = \begin{bmatrix} y \\ -y \end{bmatrix} \text{ og } b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Jeg har kalt det blandingsproblemet, siden det er en blanding av hard margin problemet og ν-SVM problemet. A og b er som i hard margin problemet, mens c er som i ν-SVM problemet. Dette tilsvarer å droppe det første leddet i hard margin problemet.

I begynnelsen skapte det første leddet i objektivfunksjonen, $\sum_{i=1}^{m} \alpha_i$, problemer. Siden ν-SVM problemet ikke hadde dette leddet i objektivfunksjonen var det nærliggende å prøve å slette dette leddet i hard margin problemet også.

Da det viste seg at denne modellen ga bedre klassifikasjonsresultater enn de andre med indrepunktsmetoden, valgte jeg å bruke denne varianten selv om det opprinnelige problemet ikke ble løst. Jeg kalte det blandingsproblemet, da det var en blanding av hard margin problemet og ν-SVM problemet.

4.6 Multiklasse problemet

I hyperspektrale data er det vanlig med flere enn to klasser, si K. Her er to enkle måter å generalisere SVM klassifikatoren:

1. Tren K binære klassifikatorer, en for hver klasse, ved å ta treningspunkter fra en av klassene mot en gruppe av treningspunktene fra de $K-1$ resterende klassene. Siden den ene klassen vil ha færre treningspunkter enn samlingen av de resterende klassene kan man vekte hvert treningspunkt med antall treningspunkter i klassen i steden for å velge ut en gruppe. Dette gjelder spesielt hvis man bruker en myk margin eller 'proximal'-klassifikator. Se Glenn Fung og O. L. Mangasarian [4]. Prøv ut alle K klassifikatorene på testdataene og velg den klassen med størst margin.

2. Tren $\binom{K}{2} = K(K-1)/2$ klassifikatorer ved å bruke alle par av klasser. Prøv ut alle klassifikatorene på testdata og gi en stemme til hver klasse som bestemmes. Velg så den klassen som får flest stemmer. Denne metoden er brukt av Gualtieri og Cromp i [6].

Det er viktig at de to "klassene" som skal skilles har like mange treningspunkter. Har den ene klassen mange flere treningspunkter enn den andre, vil den med flest bli favorisert av algoritmen. Det er to løsninger på dette problemet. Enten kan man bruke et utvalg slik at det blir like mange fra hver del, eller man kan vekte de to klassene slik at man får en balanse i algoritmen.

Jeg har valgt å bruke metode 2. slik at jeg trener $K(K-1)/2$ klassifikatorer. Hvis K er stor, dvs hvis det er mange forskjellige klasser, må man trene opp mange flere klassifikatorer enn om man hadde brukt metode 1. Da kan det være tidsbesparende å bruke metode 1. Siden det bare er 3 klasser i det største problemet jeg skal klassifisere vil jeg bare trenge å trene opp 3 klassifikatorer som er like mange som for metode 1. Da slipper jeg også å ta hensyn for at det blir en skjevhet i antall treningspunkter.

4.7 Beskrivelse av algoritmer

4.7.1 Senter algoritmen

Senter algoritmen implementerer det enkle senter problemet beskrevet i seksjon 3.7.

4.7.2 PathQuad algoritmen

PathQuad algoritmen implementere indrepunktsmetoden på blandingsproblemet som er gitt av likningene 4.31 Den inneholder ikke noen C som Senter algoritmen og har dermed èn parameter mindre å optimalisere enn OnLine og QuadProg algoritmene.

4.7.3 OnLine algoritmen

OnLine algoritmen er hentet fra [2] og er implementert i matlab. Den løser C-SVM problemet og tar tre parametere, Kjernematrise K, en vektor y av -1 og +1 elementer som angir klassetilhørighet for dataene i kjernematrisen og C som angir margin-mykheten. $C = \infty$ gir hard margin modellen.

OnLine er en iterativ metode som oppdaterer en α_i om gangen derav navnet on-line. Den unngår kravet om at $\sum_i y_i \alpha_i = 0$ ved å bestemme $b = 0$ fra starten av. Maks antall iterasjoner har blitt satt til å være 10.000.

4.7.4 QuadProg1 algoritmen

QuadProg1 algoritmen benytter seg av matlab sin kvadratiske optimeringsrutine quadprog til å løse C-SVM problemet. Matlab sin quadprog rutine kjører sin medium skala metode som er en aktiv sett algoritme hvor maks antall iterasjoner er 200.

4.7.5 QuadProg2 algoritmen

I QuadProg2 algoritmen er det brukt en forenkling av problemet.

$$\max_{\alpha} \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (K(x_i, x_j) + \frac{1}{r} I) \tag{4.34}$$

$$\text{forutsatt at} \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{4.35}$$

$$\text{og der} \quad \alpha_i,\ i = 1, \ldots, m \text{ er frie variable.} \tag{4.36}$$

Her står I for identitetsmatrisen. Ved å la α_i være frie variable kan quadprog å løse problemet med en prekondisjonert konjugert gradienters metode som er meget rask og skalerer godt. Den klarer også å oppnå gode klassifikasjonsresultater.

Kapittel 5

Klassifikasjon med kjernefunksjoner

I dette kapitlet skal vi først se på hvor godt den polynomiske, gaussiske og sigmoide kjernefunksjonen klassifiserer sjakkbrettdatasettet og så se videre på hvordan parameterinnstillingen i den gaussiske kjernefunksjonen spiller inn på klassifikasjonen av virveldatasettet.

5.1 Klassifikasjon med forskjellige kjernefunksjoner av sjakkbrettdatasettet

Sjakkbrettdatasettet er et relativt komplisert sett der 2 klasser er fordelt i annenhver rute i et sjakkbrett mønster. Det er laget ved å generere 50 tilfeldig normalfordelte punkter for hver rute. Halvparten av punktene er så tatt ut til et treningssett og den siste halvparten er brukt i et testsett.

Trenings- og testsettet er så normalisert slik at de har middelverdi 0 og varians 1. De er så transformert av kjernefunksjonen før de er overlatt til klassifikasjonsalgoritmen.

I tabell 5.2 kan vi se bilder av fire klassifikasjoner med hver sin kjernefunksjon. Treningspunktene er plottet som stjerner. Det kreves en nokså komplisert diskriminantfunksjon for å skille de to klassene fra hverandre.

I figur 5.1 kan vi se klassifikasjonsresultater av testsettet klassifisert av 2 klassifikasjonsalgoritmer med forskjellige kjernefunksjoner.

Den beste klassifikasjonen ble gjort av pathQuad og onLine algoritmen med polynomisk kjernefunksjon, mens den gaussiske kjernen ga gjennomsnittlig best resultat for alle klassifikasjonsalgoritmene.

Fra tabellen kan vi lese at med riktig valgt kjernefunksjon kan problemet løses nesten like godt med den enkle senter algoritmen som med den mer kompliserte C-SVM metoden løst av de andre algoritmene.

Vi kan trekke den konklusjonen at det er kjernefunksjonen og ikke klassifikatoren som spiller størst rolle for dette problemet, men vi kan ikke si

Test av kjernefunksjoner				
Gaussisk kjerne:				
Parametere:	senter	pathQuad	onLine	quadProg1
sigma	0.1	0.3	0.3	0.1
C	-	-	100	25
% rett	93.75	95.75	95.75	92.75
Polynomisk kjerne:				
Parametere:	senter	pathQuad	onLine	quadProg1
d	3	9	6	8
C	-	-	25	100
% rett	53.5	96.75	96.75	59.25*
Sigmoid kjerne:				
Parametere:	senter	pathQuad	onLine	quadProg1
k	0.4	0.1	0.1	0.1
q	2-	1	1	1
C	-	-	100	10
% rett	49	63	54*	54*

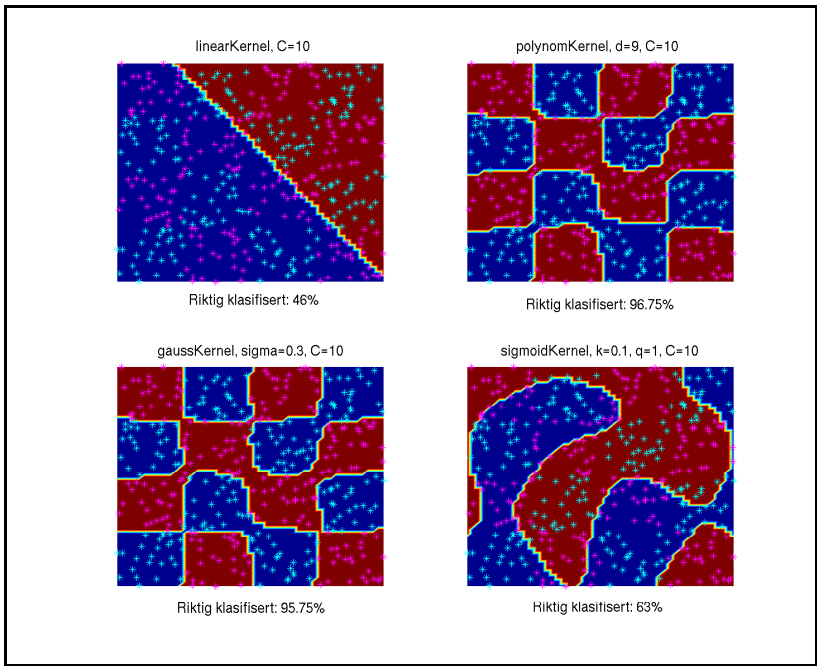
Figur 5.1: Tabellen viser klassifikasjonsresultater av testsettet til sjakkbrett-datasettet. Det er brukt forskjellige kjernefunksjoner og klassifikatorer. Med gaussisk kjernefunksjon klarte alle klassifikatorene å klassifisere testsettet med god nøyaktighet. Med polynomisk kjernefunksjon klarte to av klassifikatorene, pathQuad og onLine klassifikasjonen godt, mens med sigmoid kjernefunksjon klarte ingen av klassifikatorene det. *Disse klassifikasjonene terminerte på maks antall iterasjoner før optimal verdi ble funnet.

at det vil være slik for andre problemer. Dette understreker i det minste diskriminant-kraften i kjernefunksjonen og man kan tenke seg at det kan være tjenlig å designe og kombinere forskjellige kjernefunksjoner etter hva slags type data man jobber med. Dette er gjort i [10].

5.2 Klassifikasjon med gaussisk kjernefunksjon av virvelfunksjonen

Et annet kjent problem fra litteraturen er å skille to klasser dreid rundt hverandre i en virvel kalt virvelfunksjonen. Vi skal med dette eksemplet prøve å vise hvordan kompleksiteten på diskriminantfunksjonen forandrer seg når vi forandrer parameterverdiene til kjernefunksjonen.

De to klassene er dannet av de parametriserte kurvene k1(t) og k2(t):



Figur 5.2: Sjakkbrettdatasettet klassifisert av pathQuad algoritmen med forskjellige kjernefunksjoner. Øverst til venstre er det klassifisert med lineær kjernefunksjon, øverst til høyre med polynomisk kjernefunksjon, nederst til venstre med gaussisk kjerne funksjon og nederst til høyre med sigmoid kjernefunksjon. Treningspunktene er plottet som stjerner, mens riktig klassifisert angir hvor mange prosent av testpunktene som ble riktig klassifisert. Dette gir et mål på hvor godt diskriminantfunksjonen klarer å etterlikne det opprinnelige sjakkbrettmønsteret og ikke bare om den klarer å klassifisere treningspunktene riktig.

$$
\begin{aligned}
t &\in (0, \ldots, 4\pi] \\
\mathrm{k1}(t) &= t * \sin(t)\mathbf{i} + t * \cos(t)\mathbf{j} \\
\mathrm{k2}(t) &= t * \sin(t + pi)\mathbf{i} + t * \cos(t + pi)\mathbf{j}
\end{aligned}
$$

Vi lager 50 punkter på hver av kurvene med lik avstand i tetta. Så velger vi ut annet hvert punkt til et treningssett og de resterende til et testsettet. Da får vi 25 punkter fra hver av de to funksjonene til å trene klassifikatoren med, og så klassifiserer vi de $25 * 2$ punktene i testsettet.

Før vi trener opp klassifikatoren regner vi ut kjernefunksjonen. Siden det var den gaussiske kjernen som ga de beste resultatene for sjakkbrettdatasettet velger vi å bruke den på virveldatasettet i sammen med quadProg1 algoritmen.

Vi skal nå se på hvilken effekt parameteren σ har på diskriminantfunksjonen i klassifikasjonen. Vi skal se at ved å justere σ kan vi forandre på kompleksiteten på diskriminantfunksjonen.

I figur 5.3 er det plottet fire klassifikasjoner av virvelfunksjonen med forskjellige verdier for σ i kjernefunksjonen. Plottet viser diskriminantfunksjonen som deler bildet inn i to områder eller klasser, ett blått og ett rødt.

Punkter innenfor det røde området blir klassifisert til den røde klassen og punkter innenfor det blå området blir klassifisert til den blå klassen. I tillegg kan vi se treningspunktene til den blå klassen plottet som lyseblå (cyan) stjerner og treningspunktene til den røde klassen plottet som rosa punkter (magenta).

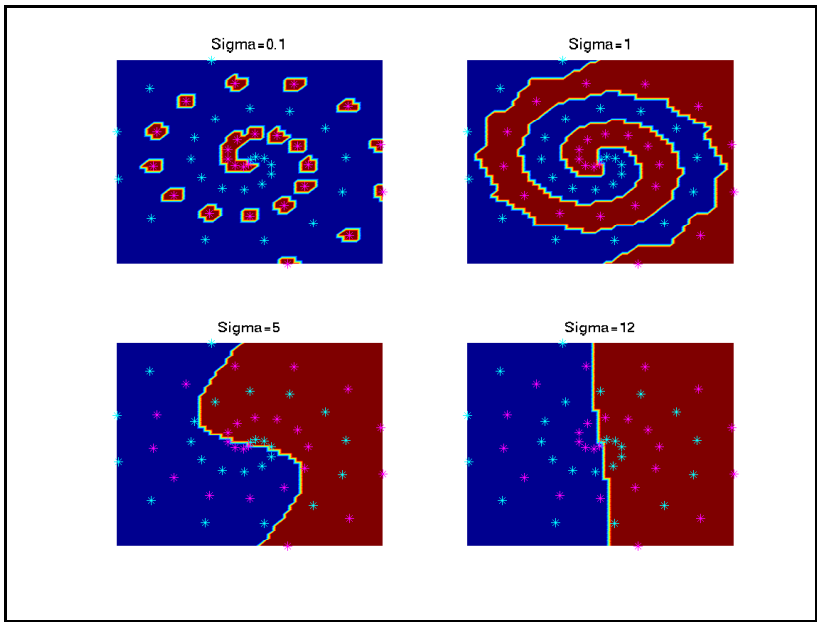
Vi kan se fra bildet øverst til venstre, der $\sigma = 0.1$, at diskriminantfunksjonen klassifiserer treningspunktene riktig, men en del av testpunktene til den ene klassen som ligger mellom treningspunktene blir feilklassifisert. Bare 62% av testpunktene blir riktig klassifisert. Diskriminantfunksjonen har for stor kompleksitet og feiler i å approksimere virvelfunksjonene.

I bildet øverst til høyre, der $\sigma = 1$, ser vi at diskriminantfunksjonen følger virvelfunksjonen rundt og 100% av testpunktene blir riktig klassifisert. Diskriminantfunksjonen har fått en passende kompleksitet.

I bildet nederst til venstre, der σ er øket til 5, har bare 52% av testpunktene blitt riktig klassifisert. Kompleksiteten på diskriminantfunksjonen har blitt for liten og den klarer ikke å følge virvelmønsteret. Vi kan se at både treningspunkter og testpunkter blir feilklassifisert.

I bildet nede til høyre, der $\sigma = 12$, er diskriminantfunksjonen nesten lineær og bare 48% av testpunktene er riktig klassifisert.

For dette problemet og av disse verdiene for σ, ga $\sigma = 1$ den beste og mest riktige klassifikasjonen.



Figur 5.3: Figuren illustrerer resultatet av forskjellige parameterverdier i gaussisk kjerne klassifisert av quadProg1 algoritmen. Klassifikasjonsnøyaktigheten av testpunktene er for hhv. $\sigma = 0.1$, 1, 5 og 12 : 62%, 100%, 52% og 48%. Vi kan se at for $\sigma = 0.1$ blir diskriminantfunksjonen for komplisert. Treningspunktene blir "memorert" av algoritmen og vi oppnår ikke stor nok generalitet i funksjonen. For $\sigma = 1$ får vi en riktigere etterlikning av virvelfunksjonene, mens for $\sigma = 5$ og 12 blir kompleksiteten for liten og diskriminantfunksjonen klarer ikke å følge virvelen rundt. Jeg har brukt SVM parameter $C = 1$ i disse plottene. For $C = 100$ ga nesten ingen endring av diskriminantfunksjonen og klassifikasjonsnøyaktigheten av testsettet, mens for $C < 0.4$ ble alle punktene klassifisert til den ene av klassene.

Kapittel 6

Klassifikasjon av Fontainebleau datasettet

6.1 Fontainebleau datasettet

Fontainebleau datasettet er tatt med en flybåren sensor 10 mai 1994 under "European Multisensor Airborne Campaigne" (EMAC-94). Det er tatt med en sensor kalt "Reflective Optics System Imaging Spectrometer" (ROSIS).

Det er tatt over et skogområde sør for Paris som inneholder verifiserte områder av eik-, bjørk- og furuskog. Datasettet har 81 spektrale bånd, ofte referert til som egenskaper, samplet med 12nm båndbredde i den nedre delen av spekteret (430-550nm) til 4nm i den øvre delen av spekteret(554-830nm), med en pikselstørrelse på 5,6m.

Videre er data settet først delt i to, et trenings/valideringssett på 8085 piksler og et testsett på 8777 piksler. Av trenings/ valideringssettet er det så laget 30 oppdelinger. Det består av 10 sett med 90 treningspunkter for hver klasse, 10 sett med 300 og 10 sett med 600. For hvert av de 30 settene er resten av de 8085 punktene som ikke var med i treningsettet så brukt i valideringen. Datasettet inneholder tre klasser.

Denne oppdelingen jeg har jobbet med er gjort av Asbjørn Berge, stipendiat innen bildebehandling på Institutt for Informatikk ved Universitetet i Oslo.

6.2 Normalisering

Før dataene brukes til å trene opp algoritmen, må de normaliseres. Det vil si at hver egenskap blir skalert til å ha varians én og middelverdi null. Dette forebygger mot numerisk ustabilitet ved flyt-tall operasjoner siden man jobber med tall i samme størrelsesorden. Man unngår at noen av egenskapene dominerer over de andre som er helt vesentlig for en god klassifikasjon med SVM.

6.3 Validering av parameterverdier

Før man klassifiserer testsettet trenger man å stille inn noen parametre. Siden man helst ikke skal justere parametrene til testsettet, er treningssettet delt i to. En del for å trene opp klassifikatoren og en del for å teste eller validere parameterinnstillingene. Man trener så opp klassifikatoren med forskjellige parameterverdier og ser hvilke verdier som gir best resultat på validering-settet. Når man er blitt fornøyd med parameterinnstillingene trener man klassifikatoren med disse før man så klassifiserer testsettet.

Jeg har brukt de parameterinnstillingene som ga best gjennomsnittlig resultat på de 10 realiseringen for hhv. 90, 300 og 600 treningspunkter pr klasse.

For å få et mål på hvor god klassifikasjonen var kan man regne ut prosent riktig klassifiserte punkter, men ettersom det ikke er like mange valideringspunkter for hver klasse vil dette gi et skjevt bilde av hvor god klassifikasjonen var. Ved først å regne ut prosentvis riktig klassifisert for hver klasse og så ta gjennomsnitt over disse vil alle klassene få like mye uttelling i klassifikasjonsprosenten uansett antall validering/testpunkter. Ved å optimalisere parameterinnstillingene på dette kriteriet i isteden for den enklere prosent riktig klassifisert unngår man at en klasse med mange testpunkter dominerer over andre klassene.

6.4 Forvirringsmatrise og klassifikasjonsmål

I tabell 6.1 ser vi en forvirringsmatrise over en klassifikasjon av et valideringssett. Radnummeret angir sann klasse eller den riktige klassen, og søylenummeret angir estimert klasse som er den klassen punktene ble klassifisert til.

Dette betyr for eksempel at i rad én og i søyle tre ser vi antall punkter tilhørende klasse én men som er feilklassifisert til klasse tre. Vi får de riktig klassifiserte punktene på diagonalen mens de feilklassifiserte punktene havner utenfor diagonalen. Med andre ord kan vi si at jo større verdier vi får på diagonalen, jo bedre ble klassifikasjonen, og jo større verdier vi får utenfor diagonalen, desto større er "forvirringen" i klassifikasjonen derav navnet forvirringsmatrise.

Isteden for å summere opp tallene på diagonalen og så få antall rett klassifiserte punkter, får vi et bedre mål ved å ta hensyn til at det er forskjellig antall punkter i hver klasse.

Dette kan vi gjøre ved å beregne prosent rett for hver klasse først og så ta gjennomsnittet av disse. Dette har vi gjort i tabell 6.2. Søylen "totalt antall" tilsvarer radsummene i tabell 6.1, og søylen "antall rett" tilsvarer tallene på diagonalen i den samme tabellen. "prosent rett" er så beregnet for hver av klassene og til slutt ender man opp med gjennomsnittet av disse

Forvirringsmatrise			
	Estimert klasse		
Sann klasse	1	2	3
1	3221	368	1006
2	382	1039	62
3	0	0	207

Figur 6.1: Forvirringsmatrisen viser klassifikasjonen av valideringssettet til treningsett nr 6 med 600 treningspunkter for hver klasse. Det er 6285 punkter som er klassifisert. Klassifiseringen er gjort av on-Line algoritmen med Gaussisk kjerne med $\sigma = 8$. Totalt tok trening og klassifikasjon ca 7 minutter.

Klassifikasjonsnøyaktighet			
klasse	totalt ant.:	ant. rett:	prosent rett:
1	4595	3221	70.10
2	1483	1039	70.06
3	207	207	100.00
Gjennomsnitt:			80.05

Figur 6.2: Tabellen viser utregning av gjennomsnittlig klassifikasjonsnøyaktighet for klassene. Tallene er hentet fra forvirringsmatrisen i tabell 6.1, der søylen "totalt ant." er radsummene og "ant. rett" er tallene fra diagonalen i forvirringsmatrisen.

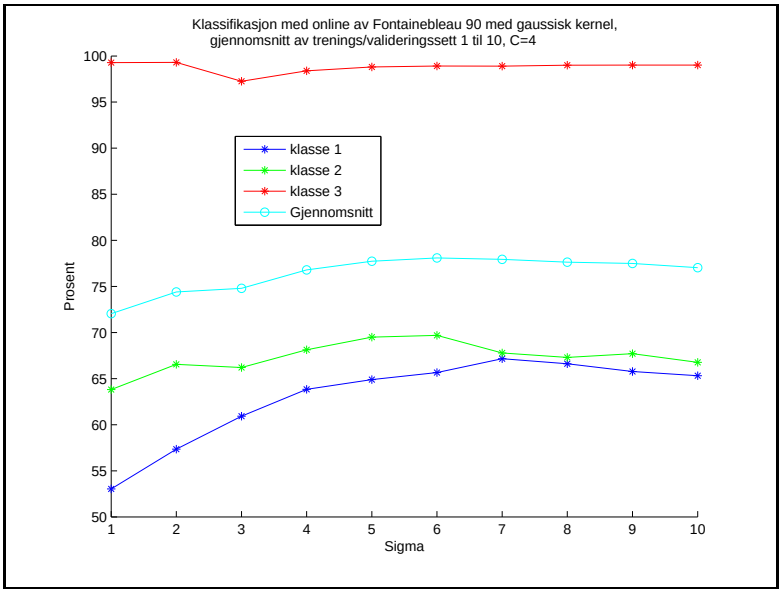
som et mål på kvaliteten av klassifikasjonen. Det er dette kvalitetsmålet vi optimaliserer i valideringsprosessen. Ved å bruke dette målet unngår vi at den klassen med flest testpunkter dominerer optimaliseringen og vi får en klassenøytral klassifikasjon av testsettet.

6.5 Validering: Innstilling av parameterverdier

Validering av parameterverdier går ut på å finne de parameterverdiene som gir best klassifikasjon etter klassifikasjonsmålet vi har definert på valideringssettene.

For pathQuad og senter metodene må vi bare optimalisere σ i gausskjernen. For onLine må vi i tillegg optimalisere C verdien til C-SVM problemet og for quadProg2 algoritmen må vi optimalisere r i tillegg til σ.

Da det er 10 par av trenings- og valideringssett for hhv 90, 300 og 600 settene må vi trene opp algoritmen og teste på valideringssett 10 ganger for hver parameterverdi. Dette må vi så gjøre for 90, 300 og 600 settene. Dette gjør at det blir tidkrevende å gjøre et utfyllende grid-søk for σ og C/r.



Figur 6.3: Figuren viser prosent riktig klassifisert av online algoritmen for hver av klassene og gjennomsnittet av disse for $\sigma = 1, \ldots, 10$.Vi får høyeste gjennomsnittverdi for $\sigma = 6$ og velger denne verdien for σ i den videre innstillingen av C.

Ved først å gjøre en grovinnstilling av C og så bruke denne verdien av C i fininnstillingen av σ ved at man tester på et litt finere grid og så bruke denne verdien av σ i fininnstillingen av C, viser det seg at man ikke trenger å stille inn σ noe mer da den fortsatt er optimal for den nye C-verdien. C og σ påvirker algoritmen uavhengig av hverandre.

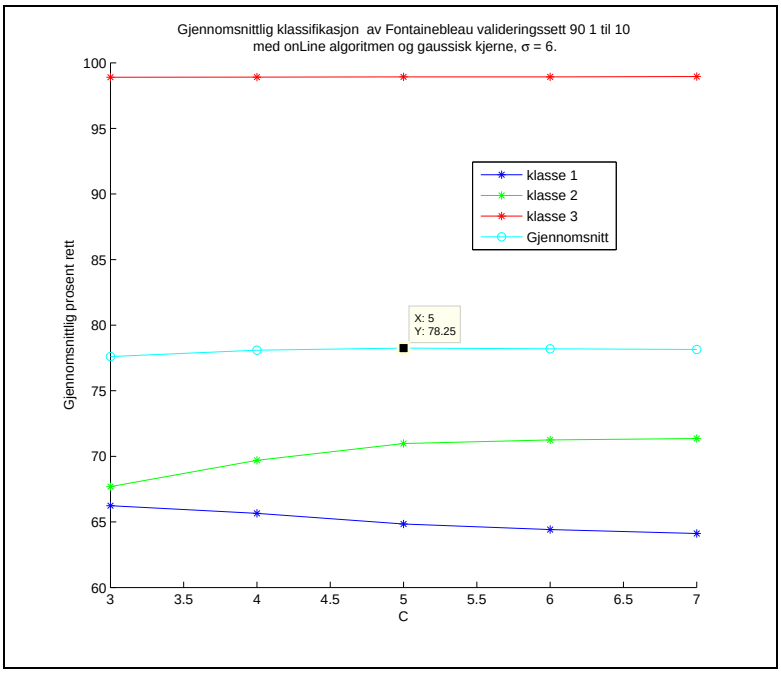
I figur 6.3 der vi kan se et plott av gjennomsnittlige klassifikasjonsresultater for Fontainebleau valideringssett 90, 1 til 10 for forskjellige σ-verdier. Vi kan se at toppen på kurven gjennomsnitt av klassene er gitt i $\sigma = 6$. Denne σ verdien er så brukt i den videre innstillingen av C. Dette er gjort i figur 6.4, som gir oss $C = 5$ som beste innstilling.

Når så σ og C er funnet, trener vi opp algoritmen med disse parameter innstillingene og klassifiserer testsettet. Dette gjør vi for alle de 10 treningsettene og så beregner gjennomsnittet og standardavviket for disse.

Vi kan se resultater for hver av algoritmene og parameterinnstillingene som er brukt i tabell 6.5.

6.6 Klassifikasjonsresultater

Tabell 6.5 viser en oversikt over klassifikasjonsnøyaktigheten for 4 klassifikatorer: senter, pathQuad, onLine og quadProg2.



Figur 6.4: Figuren viser gjeneomsnittlig klassifikasjonsresultater for valideringsett 90, 1 - 10 av de tre klassene og gjennomsnittet av disse for forskjellige C-verdier klassifisert med onLine algoritmen hvor den optimale σ verdien vi fant i figur 6.3 er brukt i gausskjernen. Vi kan se at $C = 5$ gir best gjennomsnittlig klassifikasjonsnøyaktighet på valideringssettene. Vi bruker så denne verdien for å klassifisere testsettet.

Fra tabellen kan vi se at jo flere punkter pr klasse som er brukt i treningen jo bedre klassifikasjonsnøyaktighet får vi. Dette kan tyde på stor kompleksitet i klassene og at det kreves en kompleks funksjon for å skille klassene som igjen krever mange treningspunkter for å få en god nøyaktighet.

Vi kan se at standardavviket synker som en følge av flere treningspunkter pr. klasse. Dette tyder også på at treningen blir mer stabil jo flere treningspunkter man bruker.

Den høye treningstiden på onLine algoritmen skyldes at den konvergerer sakte med opptil 9000 iterasjoner for hver av de 10 600-treningsettene. For eksempel for 600 - 1 settet brukte den $7541 + 685 + 490 = 8716$ iterasjoner. Siden onLine algoritmen oppdaterer én α_i om gangen har den en dobbel forløkke som ikke kan vektoriseres. Det gjør at den blir noe tregere i matlab enn om den hadde vært implementert i et annet programmeringsspråk som for eksempel i C++.

QuadProg1 algoritmen klarte bare treningen med 90 settene men ikke med 300 og 600 settene. Den terminerte på maks antall iterasjoner og klarte ikke å klassifisere testsettet tilfredstillende. Ved å droppe begrensningene på α_i og dermed forenkle problemet, viste det seg at matlab sin quadprog funksjon kunne benytte seg av en meget rask "Prekondisjonert Konjugerte Gradienters"-metode (PCG). PCG er en iterativ metode for å løse et positivt definitt lineært system av likninger [9].

I tillegg til å droppe kravene på α_i var det anbefalt å legge til ett lite tillegg $1/r$ på diagonalen ev kjernematrisen i boka til Nello Christianino og John Shawe-Taylor [2] da det kunne forbedre klassifikasjonen noe. Dette ble opphavet til den nye quadProg2 algoritmen som viste seg å gi gode klassifikasjonsresultater til tross for at den avvek noe fra det opprinnelige C-SVM problemet. I tillegg hadde den en meget kort treningstid selv for 600 settene.

PathQuad algoritmen har nesten like god klassifikasjons-nøyaktighet som onLine men er mye raskere siden den kunne i større grad vektoriseres.

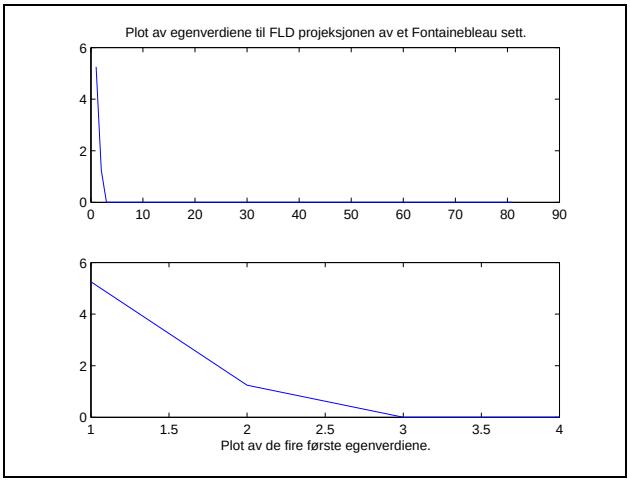
Senter algoritmen klarer ikke å skille klassene så godt som de SVM baserte metodene. Dette var forventet men det er også interessant å se hvor godt en enkel algoritme klarer å klassifisere et komplisert problem ved hjelp av kjernefunksjonen. Siden den er en enkel algoritme bruker den kort tid men allikevel litt lenger tid enn quadProg2 algoritmen. Dette skyldes nok at PCG metoden quadProg2 benytter seg av er nøye implementert og optimalisert for matlab.

6.7 Egenskapstransform med Fishers Lineære Diskriminant (FLD)

Siden SVM ikke lider under høy dimensjon, kan man spørre seg om egenskapsreduksjon spiller noen rolle. Jeg har prøvd Fishers projeksjon (FLD)

Gjennomsnittlige klassifikasjonsresultater av Fontainebleau testsettet over 10 treningsett for 90, 300 og 600 treningspunkter pr. klasse med gaussisk kjernefunksjon.			
	Ant. treningspunkter pr. klasse		
	90	300	600
Senter			
σ	2	2	2
Klassesnitt $\pm$ std.:	73.65 ± 0.61	73.77 ± 0.40	73.78 ± 0.22
Prosent rett $\pm$ std.:	60.21 ± 2.49	61.91 ± 0.87	62.22 ± 0.79
Treningstid:	2 sek	9 sek	21 sek
Pathquad			
σ	10	10	10
Klassesnitt $\pm$ std.:	78.07 ± 0.98	79.21 ± 0.80	82.77 ± 0.61
Prosent rett$\pm$ std.:	66.50 ± 2.19	72.30 ± 0.90	74.86 ± 0.45
Treningstid:	4 sek	30 sek	2 min, 26 sek
OnLine			
C	5	8	30
σ	6	7	7
Klassesnitt $\pm$ std.:	77.17 ± 1.71	81.47 ± 1.23	82.30 ± 1.17
Prosent rett$\pm$ std.:	68.94 ± 2.73	75.23 ± 3.11	75.52 ± 3.28
Treningstid:	4 sek	1 min, 15 sek	8 min, 36 sek
QuadProg2			
r:	10	15	35
σ:	6	8	15
Klassesnitt$\pm$ std.:	76.94 ± 0.73	81.19 ± 0.54	83.04 ± 0.74
Prosent rett$\pm$ std.:	66.94 ± 1.67	73.07 ± 0.88	74.79 ± 1.57
Treningstid:	1 sek	6 sek	14 sek

Figur 6.5: Klassifikasjonsresultater for Fontainebleau testsettet. Det er på 3 klasser. Det er 4 forskjellige klassifikatorer som er brukt.



Figur 6.6: I delplott 1 ser vi plott av de 81 egenverdiene til FLD projeksjonen. I delplott 2 ser vi plott av de 4 første egenverdiene. Vi ser at det er bare de to første egenverdiene som er signifikant større enn null. Dette stemmer overens med teorien om at projeksjonen har rang én mindre enn antall klasser, dvs 2.

som maksimerer variansen mellom klassene og minimerer variansen innenfor klassene, og Prinsipal Komponent Analyse (PCA).

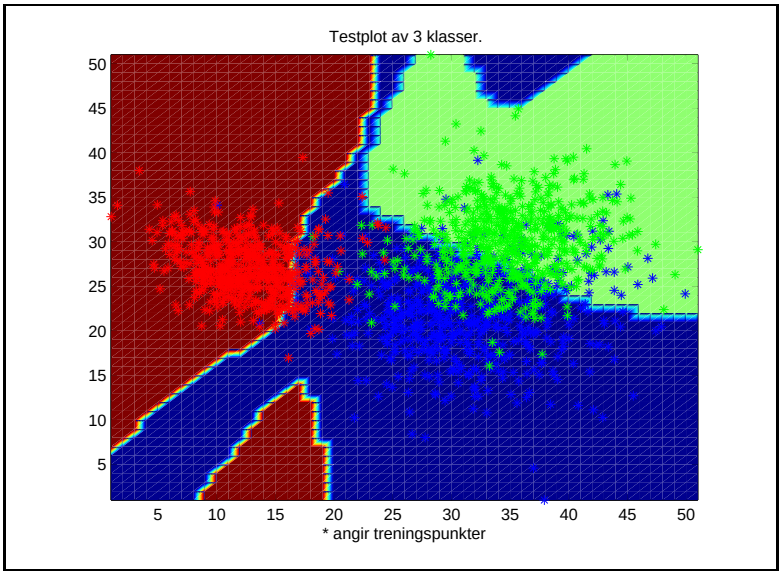
FLD projiserer dataene ned i et rom én dimensjon mindre enn antall klasser som beskrevet i avsnitt 1.4.2. For Fontainebleau datasettet viste testresultatene at det ikke spiller noen rolle om jeg bruker to, tre eller fire egenvektorer i projeksjonen. Men derimot fem eller flere ga et dårligere klassifikasjonsresultat. Dette stemmer overens med teorien som sier at rangen til projeksjonen er én mindre enn antall klasser. Så egentlig er det ikke mulig å få mer enn $M - 1$ antall egenvektorer med FLD der M er antall klasser. I dette tilfelle, da vi har 3 klasser gir det 2 egenvektorer, men siden matlab regner numerisk vil vi få flere egenvektorer, men der egenverdiene er nesten lik 0. Dette kan vi se fra figur 6.6 der vi ser at hovedsakelig all energi er samlet i de to første egenverdiene. I resultatene for FLD er det bare brukt de to egenvektorene tilhørende de to største egenverdiene.

I tabell 6.7 kan vi se klassifikasjonsresultater for fontainebleau testsettet klassifisert med pathQuad algoritmen hvor de 81 oprinnelige egenskapene har blitt projisert av FLD metoden inn i 2 nye egenskaper. Ved å sammenligne med resulatatene for pathQuad algoritmen i tabell 6.5 kan vi se at for "Snitt klasseprosent riktig"/"klassesnitt" ligger FLD ca 3% under for 90 og 300 settene og ca 2% for 600 settet. Standard avvikene er også høyere for FLD som betyr at variasjonen av klasssifikasjonen for FLD er større enn for de opprinnelige 81 egenskapene.

I figur 6.8 kan vi se et plott av klassifikasjonsgrensene sammen med tre-

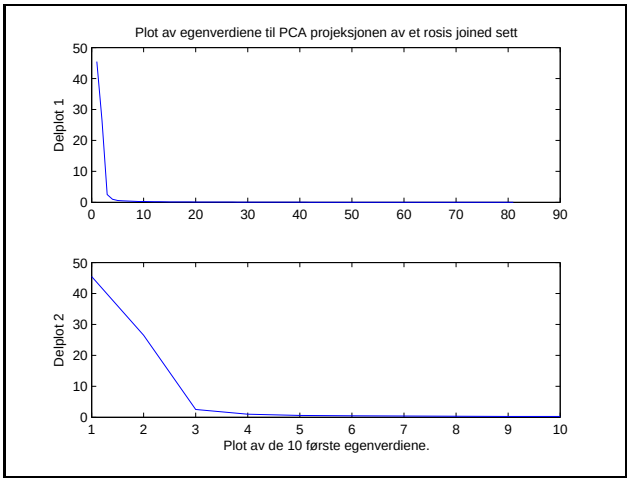
Klassifikasjon av FLD projeksjon av Fontainebleau datasettet klassifisert av pathQuad algoritmen med gaussisk kernel.						
	Prosent rett klassifisert			*Snitt klasseprosent riktig*		
σ:	1	3	2	1	3	2
treningssett	90	300	600	90	300	600
1	60,8864	74,3534	75,8232	70,3476	80,7100	81,2310
2	64,5209	71,8469	78,3525	74,1716	78,2213	81,3102
3	64,4639	65,9907	73,4761	72,3404	76,8761	80,5060
4	69,0099	71,3228	77,7145	75,1609	78,7786	81,4795
5	61,7409	67,7794	76,0055	72,7114	76,0808	80,2027
6	58,8926	71,4823	80,6654	70,0824	79,6696	83,2483
7	64,9311	70,1265	77,6347	73,0195	77,6961	81,5748
8	62,8347	72,0861	71,3911	71,8703	78,4017	78,2547
9	56,5113	69,1125	73,4533	69,1021	77,1304	78,8818
10	66,8452	63,6550	68,2921	75,2451	76,7775	72,7342
snitt	63,0637	69,7756	75,2808	72,4051	78,0342	79,9423
Std. avvik:	3.68	3.2	3.67	2.11	1.42	2.9

Figur 6.7: Fra tabellen kan vi lese at treningssett 6 med 600 treningspunkter ga best klassifikasjonsnøyaktighet på testsettet med 80,7 %. Det samme settet ga også maksimum for snitt klasseprosent riktig med 83,2 %. Ved å sammenligne gjennomsnittet av prosent rett klassifisert med verdiene for pathQuad i tabell 6.5 ser vi at resultatene ligger ca 3 % under for 90 og 300 settene mens for 600 settene ligger FLD ca 2% under. Dette tyder på at FLD projeksjonen tar vare på det meste av separerbarheten i bare to egenskaper!



Figur 6.8: Bildet viser klassifikasjonsgrensene og treningspunktene til Fontainebleau treningssett nr 6 med 600 treningspunkter for hver klasse projisert ned i 2 dimensjoner av FLD. Bakgrunnsfargene illustrerer hvilke områder som blir klassifisert til de forskjellige klassene. Siden treningspunktene fra de forskjellige klassene overlapper hverandre er det ikke mulig å skille de 100%. Isteden prøver algoritmen å finne en separasjon med minst mulig feilklassifisering. Klassifikasjonen er gjort av pathQuad algoritmen med blandingsvarianten av SVM med gaussisk kjerne med $\sigma = 2$.

ningspunktene for Fontainebleau treningssett nr 6 med 600 treningspunkter for hver klasse projisert ned i 2 dimensjoner av FLD. Det er litt uttydelig å se at de blå og grønne treningspunktene overlapper ganske mye siden de grønne punktene er plottet oppå de blå. De røde treningspunktene skiller seg ganske godt fra de andre klassene. Dette gjør at den røde klassen blir lett godt klassifisert mens den blå og grønne har motsatt korrelerte klassifikasjonsresultater. Flytter man klassifikasjonsgrensene mellom dem lengre over i det grønne området vil man få flere riktig klassifiserte punkter fra den blå klassen, men færre fra den grønne. Hvis man flytter grensen lenger ned i det blå feltet vil det motsatte skje. Dette gjør klassifikasjonsoppgaven mere krevende i og med at man må finne en riktig balanse mellom den blå og den grønne klassen.



Figur 6.9: Som for FLD ser vi at egenverdiene faller raskt mot null. Det vil si at det meste av informasjonen med hensyn på korrelasjonen ligger i egenvektorene tilsvarende de første egenverdiene. Egenverdiene holder seg litt lengre over null enn hos FLD, noe som tyder på et flere egenvektorer vil ha noe å si for klassifikasjonen, noe som testresultatene også viser.

6.8 Egenskapstransform med prinsipalkomponentanalyse

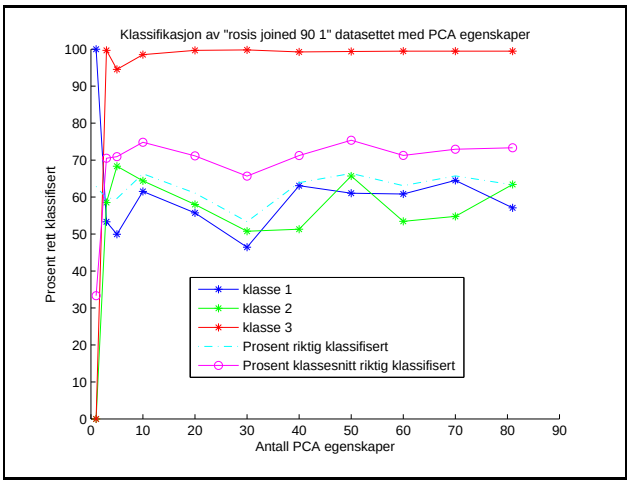
Prinsipalkomponentanalyse (PCA) benytter seg av korrelasjonen mellom egenskapene til dataene og projiserer dataene inn i et rom som minimerer korrelasjonen mellom egenskapene som beskrevet i avsnitt 1.4.1. I motsetning til FLD som bruker informasjon om klassene til å øke eller beholde separerbarheten, kan man se på PCA som en kompresjon av dataene som minimerer korrelasjonen mellom egenskapene.

I figur 6.9 kan vi tydelig se hvordan energien er samlet i de første egenverdiene.

I figur 6.10 har vi plottet klassifikasjonsnøyaktigheten for Fontainebleau datasettet med varierende antall PCA-egenskaper. Vi kan se at med relativt få egenskaper, mindre enn 10, kommer klassifikasjonen opp på et akseptabelt nivå. Dette stemmer overens med at bare de 8 - 10 første egenverdien er signifikant over null og vi kan da anta at de 8 - 10 første egenskapene inneholder mesteparten av informasjonen i datasettet.

I tabell 6.11 kan vi se gjennomsnittlige klassifikasjonsresultater for Fontainebleau testsettet trenet med 90-settene og klassifisert med forskjellig antall PCA egenskaper og forskjellige klassifikatorer.

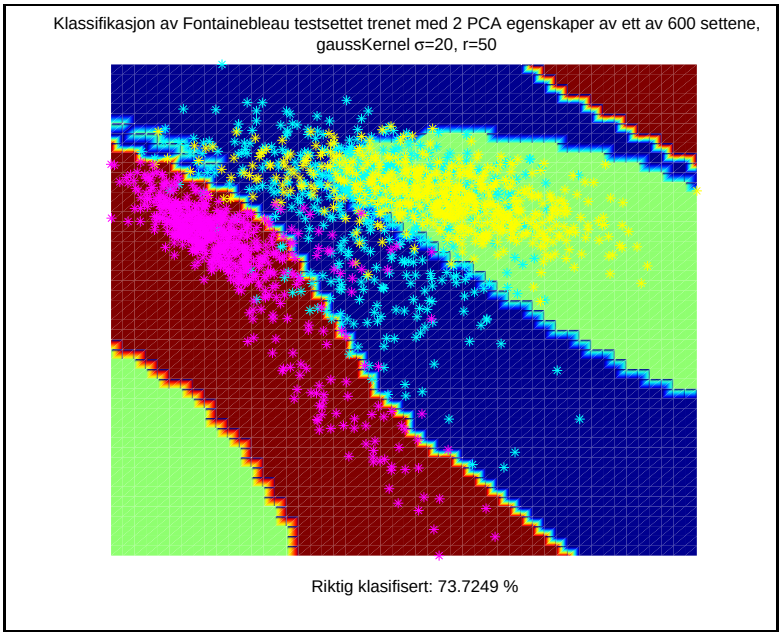
I figur 6.12 kan vi se klassifikasjonsresultat for testsettet klassifisert av quadProg2 algoritmen trent med ett av 600 treningsettene med 2 PCA egen-



Figur 6.10: Her ser vi hvordan klassifikasjonsnøyaktigheten varierer for de tre klassene i Rosis-Joined datasettet med antallet PCA-egenskaper brukt i klassifikasjonen. Det viser også prosent riktig klassifisert og snittet av de tre klassene. Jeg ville ventet at vi ville sett en jevnere stigning og mindre variasjon ettersom det ble brukt flere av egenskapene. Den store variasjonen kan skyldes at det er forholdsvis få treningspunkter og derfor ble konsekvensene av små forandringer av egenskapene store på klassifikasjonen av testsettet. Det som kommer tydelig frem er at med forholdsvis få egenskaper, ca 10, kommer klassifikasjonen opp på et akseptabelt nivå. Dette stemmer overens med tolkningen av egenverdiene der bare de ti første er signifikant større enn null.

Gjennomsnittlige klassifikasjonsresultater for Fontainebleau testsettet over 10 treningsett for 90 treningspunkter pr. klasse med gaussisk kjernefunksjon.			
	Antall PCA egenskaper		
Algoritme og parametre	10	40	81
Senter			
$\sigma \pm 1$:	2	2	2
Klassesnitt ± std.:	72.27 ± 0.42	72.57 ± 0.58	72.63 ± 0.60
Prosent rett:	59.91 ± 2.32	60.12 ± 2.47	60.20 ± 2.48
Treningstid	1 sek	1 sek	1 sek
PathQuad			
$\sigma \pm 2$	6	8	10
Klassesnitt ± std.:	71.69 ± 0.95	75.63 ± 0.98	76.42 ± 0.98
Prosent rett:	61.41 ± 1.66	61.55 ± 2.09	66.51 ± 2.19
Treningstid:	4 sek	4sek	4 sek
OnLine			
C ± 1:	5	8	5
$\sigma \pm 2$	8	6	6
Klassesnitt ± std.:	75.20 ± 1.51	76.86 ± 1.80	77.26 ± 1.98
Prosent rett:	66.80 ± 2.49	68.01 ± 3.12	68.72 ± 3.24
Treningstid:	4 sek	4 sek	4 sek
QuadProg2			
r ± 10:	40	40	50
$\sigma \pm 2$:	14	10	10
Klassesnitt: ± std.:	73.89 ± 0.56	77.0462 ± 0.95	78.19 ± 0.86
Prosent rett:	64.15 ± 1.38	66.95 ± 1.72	68.27 ± 1.23
Treningstid:	1 sek	1 sek	1 sek

Figur 6.11: Klassifikasjonsresultater for Fontainebleau testsettet trenet med 90 treningspunkter pr. klasse projisert ned fra 81 egenskaper til 10, 40 og 81 PCA-egenskaper. Ved å sammenlikne med tabell 6.5 kan vi se at PCA-egenskapene ga stort sett dårligere resultater enn de originale. Med alle 81 PCA-egenskapene ga det likevel tilnærmet like gode resultater. ± etter parameternavnene angir grid-størrelsen for parametersøket.



Figur 6.12: Her kan vi se resultatet av klassifikasjonen av Fontainebleau testsettet av quadProg2 algoritmen trent med 2 PCA-egenskaper av ett av 600 treningssettene.

skaper. Punktene som er plottet er treningspunktene mens bakgrunnsfargen angir områdene som klassifiserer til hver klasse. Prosent rett er gjennomsnitt riktig klassifisert for klassene i testsettet. Parametrene σ og r er ikke optimale men er grovinnstilt for 600 settene med 81 PCA egenskaper. Allikevel gir plottet et representativt bilde av en klassifikasjon med de to første PCA egenskapene.

Kapittel 7

Oppsummering og videre arbeide

Vi har nå sett på klassifikasjonsresultater for forskjellige algoritmer.

Vi kan se at OnLine algoritmen bruker lang tid på trening. Dette skyldes ikke at dette er en dårlig algoritme men at den ikke egner seg så godt i matlab da den i denne formen ikke kan vektoriseres i noe særlig grad. Hadde den vært implementert i C++ for eksempel kunne vi regnet med en betydelig hastighetsøkning. På den annen side ga den god klassifikasjonsnøyaktighet.

QuadProg2 algoritmen som benytter seg av den prekondisjonerte konjugerte gradienters metode (PCG), var den raskeste med en ordens forskjell i forhold til OnLine og PathQuad. Den var til og med litt raskere enn senter algoritmen. Det kan være interessant å studere nærmere hvilke konsekvenser det har for det primale problemet å droppe begrensningene på α_i'ene i det duale problemet slik som quadProg2 gjorde for at matlab skulle kjøre PCG-metoden. Ettersom den også ga de beste klassifikasjonsresultater på testsettet av Fontainebleau datasettet gjør det at dette er ekstra interessant.

Det kan være interessant å studere nærmere modeller som kan løses ved et lineært uavhengig likningssett da de kan løses meget raskt med f.eks. PCG metoden. Proximal-SVM modellen er en slik modell som bare krever at man må løse et likningssett. De viser i [5] og [4] at den klarer å klassifisere en god del testsett med god klassifikasjonsnøyaktighet. Mine foreløpige resultater med en implementasjon av den kvadratiske psvm algoritmen gitt i [5] viser at den klarer å klassifisere min versjon av virvelsettet med 86% rett med gaussisk kjernefunksjon, $\sigma = 0.5$ og $\nu = 1$, sjakkbrettdatasettet klarer den med 80.25% med $\sigma = 0.5$ og $\nu = 5$, men på Fontainebleau testsettet klarer den ikke mere enn ca 60% gjennomsnittlig for de tre klassene med gaussisk kjernefunksjon.

De foreløpige klassifikasjonsresultatene av Fontainebleau testsettet med FDR og PCA projeksjoner sier at man ikke oppnår bedre klassifikasjonsresultater med FDR eller PCA enn uten. Selv med alle 81 PCA egenskapene

ga det ikke noe bedre resultater enn med de originale 81 egenskapene. Allikevel var ikke klassifikasjonen så mye dårligere enn at det kan oppveie for en mulig gevinst i kjøretid om man kan benytte seg av dette i algoritmen.

Videre arbeid kan bestå i å prøve ut flere algoritmer for eksempel en "Nærmeste punkt"-algoritme for SVM som er beskrevet i [7].

Bibliografi

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[2] Nello Christianino and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University press, Cambridge, 2000.

[3] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

[4] Glenn Fung and O. L. Mangasarian. Multicategory proximal support vector machine classifiers. In *Data Mining Institute Report*, 2001. ftp://ftp.cs.wise.edu/math-prog/tech-reports/01-06.ps.

[5] Glenn Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In F. Provost and R. Srikant, editors, *Knowledge Discovery and Data Mining*, pages 77–86. Asscociation for Computing Machinery, August 2001. ftp://ftp.cs.wise.edu/math-prog/tech-reports/01-02.ps.

[6] J. Anthony Gualtieri and R. F. Cromp. Support vector machines for hyperspectral remote sensing classification. In *Advances in Computer Assisted Recognition*, volume 3585, Washington D.C., 1998. Proceedings of the SPIE.

[7] S. S. Keethi, S. K. Shevade, C. Battacharyya, and K. R. K. Murthy. A fast iterativ nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks*, 11(1), januar 2000.

[8] David Landgrebe. Information extraction principles and methods for multispectral and hyperspectral image data. *http://dynamo.ecn.purdue.edu/˜landgreb/Principles.pdf*, 1998.

[9] Tom Lyche. Lecture notes for mod200. *Institutt for Informatikk, Universietet i Oslo*, 2002.

[10] Grégoire Mercier and Marc Lennon. Support vector machines for hyperspectral image classification with spectral-based kernels. *Geoscience and Remote Sensing Symposium, 2003. IGARSS '03. Proceedings. 2003 IEEE International*, 1:288–290, 2003. http://perso-iti.enst-bretagne.fr/~mercierg/articles/igarss03/Paper_03-1432.pdf.

[11] Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–202, 2001.

[12] P. M. Narendra and K. Fukunage. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, C-26:917–922, 1977.

[13] Bernhard Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithm. In *Neural Computation*, number 5 in 12, pages 1207–1245. The MIT Press, 2000.

[14] Bernhard Schölkopf and Alexander J. Smola. *Learning with kernels : Support Vector Machines, Regularisation Optimization, and Beyond*. The MIT Press Massachuset Institute of Technology press, Cambridge, Massachusetts 02142, http://mitpress.mit.edu, 2002.

[15] S. Theodoridis and K. Koutroumbas. *Pattern recognition*. Academic Press., 1998.

[16] Robert J. Vanderbei. *Linear Programming, Foundations and Extensions*. Kluwer Academic Publishers, 2001.

Tillegg A

Programmkode

A.1 Klassifikatorer

A.1.1 Indrepunktsmetoden

```
function [alpha,b] = pathQuad2(K,y,nu,valg)
% function [alpha,b] = pathQuad(K,y)
% Path-following algoritme figur 17.1 side 296 i Linear
% Programming, Foundations and extensions av R. J. Vanderbei
% Løser problemet:
% Minimize  c'x + 1/2 x'Qx
% s. t.     Ax >=b
% Ved Barrier problemet:
% Maximize c'x + 1/2 x'Qx + mu*sum_j(log x_j) + mu*sum_i(log w_i)
% S. t. Ax-w=b

TESTUTSKRIFT = 1;
if nargin <4
  valg =4;
end
while ~(valg ==1 || valg==2||valg==3||valg==4)
  disp('****Velg SVM-type****');
  disp('1: nu-SVM');
  disp('2: C-SVM');
  disp('3: SVM-Hard Margin');
  disp('4: Blanding');
  valg=input(': ');
  if ~(valg ==1 || valg==2||valg==3||valg==4)
    disp('Du tastet feil! Prøv igjen.');
  end
end
```

```
if valg==1
  if nargin == 2
    nu=input('Oppgi nu: ');
  end
  A=[y'; -y'; ones(size(y')); -eye(length(y))];
  b=[0;0; nu; (-1/length(y))*ones(length(y),1)];
  [m,n]=size(A);
  c=zeros(n,1);
elseif valg == 2
  %C=input('Hva er C? (0..1): ');
  if nargin > 2
    C=nu;
    fprintf('C=%i\n',C);
  else
    C=1;
  end
  A=[y'; -y'; -eye(length(y))];
  b=[0;0; -C*ones(length(y),1)];
  [m,n]=size(A);
  c=-1*ones(n,1);
elseif valg == 3
  A=[y'; -y'];
  b=[0;0];
  [m,n]=size(A);
  c=-1*ones(n,1);
elseif valg == 4
  A=[y'; -y'];
  b=[0;0];
  [m,n]=size(A);
  c=zeros(n,1);
end
%c=zeros tilsvarer myk margin. Gir bedre klassifikasjon
Q=(y*y').*K;
klasse=y;
%Initsierer startverdier
x=ones(n,1);
z=ones(n,1);
y=ones(m,1);
w=ones(m,1);

optimal=0;

%Parametre til algoritmen
eps=1/1000000; %stopp verdi for tilatt feil
```

```
%M=1000;     %maks verdi for ubegrenset løsning
M=inf;
r=0.9;
d=0.1;% 0<d<1
ro=b-A*x+w;
si=c-A'*y-z+Q*x;
ga=z'*x+y'*w;
ga_gammel=ga;

% (1) Finner Barriere parameteren mu:
mu=d*ga/(n+m);
i=0;
if TESTUTSKRIFT
  fprintf('Avstand fra primalt tillatt: %f\n',sum(abs(ro)));
  fprintf('Avstand fra dualt tillatt: %f\n',sum(abs(si)));
  fprintf('komplemtær slakk: %f\n',ga);
%  fprintf('Barriere parameter mu: %f\n',mu);
  fprintf('Iterasjon: %i komplementær slakk: %g',i,ga);
end
while(~optimal)
  i=i+1;

  X=spdiags(x,0,n,n);
  Z=spdiags(z,0,n,n);
  Y=spdiags(y,0,m,m);
  W=spdiags(w,0,m,m);

  % (2) Finner skritt retningen dx, dy, dz og dw ved:
  % (a) Karush-Kuhn-Tucker systemet:

  % likn. 18.5 Linear Programming boka.
  % B=[-X*inv(Z) zeros(n,m) -eye(n,n)   zeros(n,m); ...
  %    zeros(m,n+m)         A           eye(m,m); ...
  %    -eye(n,n) A'         zeros(n,n)  zeros(n,m); ...
  %    zeros(m,n) eye(m,m)   zeros(m,n) A*inv(W)];
  % h=[-mu.*inv(Z)*ones(n,1)+x; ro; si; mu.*inv(W)*ones(m,1)-y];
  % dzdydxdw=B\h;

  % eller
  % (b) Karush-Khun-Tucker reduserte system
  B2=[Y\W A; ...
      A' -(X\Z+Q)];
  h2=[b - A*x + mu*(Y\ones(m,1)); ...
      c - A'*y - mu*(X\ones(n,1))+Q*x];
```

```
  dydx=B2\h2;
  dy = dydx(1:m);
  dx = dydx(m+1:m+n);
  dz = X\( mu*ones(n,1) - X*Z*ones(n,1) - Z*dx );
  dw = Y\( mu*ones(m,1) - Y*W*ones(m,1) - W*dy );
  dzdydxdw= [dz; dy; dx; dw];

  % (3) Beregner skritt lengden tetta slik at det nye punktet fortsatt
  % har possitive verdier
    tetta=min(r*inv((max(-dzdydxdw./[z;y;x;w]))),1);

  % (4) Finner ny løsning [z;y;x;w]
  zyxw=[z;y;x;w]+tetta.*dzdydxdw;
  z=zyxw(1:n);
  y=zyxw(n+1:n+m);
  x=zyxw(n+m+1:2*n+m);
  w=zyxw(2*n+m+1:2*n+2*m);
  ro=b-A*x+w;
  si=c-A'*y-z+Q*x;
  ga=z'*x+y'*w;
  if TESTUTSKRIFT
%   fprintf('Avstand fra primalt tillatt: %f\n',sum(abs(ro)));
%   fprintf('Avstand fra dualt tillatt: %f\n',sum(abs(si)));
    fprintf(['\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\' ...
             'b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\bIterasjon: %i ' ...
              'komplementær slakk: %g'],i,ga);
  end
  % (1) Finner Barrier parameteren mu:
  mu=d*ga/(n+m);
  %fprintf('barrierparameter mu: %f \n',mu);

  if(max(abs(x))>M)
    error('Problemet er primalt ubegrenset.');
  elseif(max(abs(y))>M)
    error('Problemet er dualt ubegrenset.');
  %  elseif(sum(abs(ga))<eps && i>10)
  elseif(sum(abs(ro))<eps && sum(abs(si))<eps &&sum(abs(ga))<eps)
    optimal=true;
  end
  if ((i>150) ||((ga_gammel-ga)<1/1000000000000))
    break;
  end
  ga_gammel=ga;
```

```
  %fprintf('ro: %g si: %g ga: %g \n', sum(abs(ro)),sum(abs(si)),sum(abs(ga)));
  %fprintf(' %g',x);
  %toc
  %x
end %while
fprintf('\nAntall Itarasjoner: %i\n',i);

alpha = x;
b=-0.5*( max((alpha.*klasse)'*K(:,find(klasse==-1)))+min((alpha.* ...
         klasse)'*K(:,find(klasse==1))));
if isempty(b)
  disp('Warning: b is a empty metrix resetting b=0');
  b=0;
end
```

A.1.2 Matlabs kvadratiske optimerer

A.1.3 On-line algoritmen

```
function [alpha,b] = on_line(K,y,C);
%OPTIMERING
%Hard margin: C=inf; 1-norm soft margin: C=en konstant.
%obj funksjon:
%sum(alpha)- 0.5*sum(sum((alpha*alpha').*(y*y').*(K+1/C*delta(i,j))));
%der delta =1 hvis i=j og 0 ellers
%delta=spdiags(ones(m,1),0,m,m);

  %Balanserer klassene
  %  m1=sum(y==-1);
  %  m2=sum(y==1);
  %  y(y==-1)=-1/m1;
  %  y(y==1)=1/m2;
  % blir ikke helt riktig ????

alpha=zeros(length(y),1);
W1=sum(alpha)- 0.5*sum(sum((alpha*alpha').*(y*y').*K));
%fprintf('\nObjektivfunskjon W(alpha) = %f',W1);
forandring=inf;
%C=4; %1 og inf ga ingen forskjell på Poly nu=1 og d=2 506 iterasjoner
grense= 0.001/(y'*y);
iter=1;
%K=K+1/C*speye(size(K));
aa1=alpha'*alpha;
vektorisert=0;
```

```
fprintf('Iterasjon: %5i',iter);
while (forandring>grense) && (iter<10000) %stoppkriterie ikke oppfylt
%iter
  fprintf('\b\b\b\b\b%5i',iter);
  if vektorisert==0
  for i=1:length(y) %for alle punkter
      alpha(i)=alpha(i)+ 1/K(i,i)*(1-y(i)*sum(alpha.*y.*K(:,i)));
      if alpha(i) < 0
         alpha(i)=0;
      elseif alpha(i)>C
         alpha(i)=C;
      end;
   end;
  else
  %vektoriserer: -kan denne brukes da den oppdaterer
  % hele alpha om gangen i motsetning til for løkken
  % som tar en om gangen?
  alpha=alpha + 1./diag(K).*(1-y.*((alpha.*y)'*K)');
  a0=alpha<0;
  %alpha=alpha-alpha.*a0;
  alpha(a0)=0;
  aC=alpha>C;
  %alpha= alpha-(alpha.*aC-C);
  alpha(aC)=C;
  end;
  %tic
   %W2=sum(alpha)- 0.5*sum(sum((alpha*alpha').*(y*y').*K));
 %toc
   %fprintf('\nObjektivfunskjon W(alpha) = %f',W2);
   %forandring=abs(W2-W1)
   %ay=alpha'*y;
   %fprintf('\nsum(alpha(i)*y(i))==0: %f',ay);
   %W1=W2;
   iter=iter+1;
   aa2=alpha'*alpha;
   forandring=abs(aa2-aa1);
   aa1=aa2;
end;
%fprintf('\nAntall iterasjoner: %i\n',iter);
fprintf('\n');
fprintf('Ant treningspunkter: %i\n',length(alpha));
fprintf('Ant sv == 0: %i\n',sum(alpha==0));
fprintf('Ant sv > 0 og < C : %i\n',sum((alpha>0).*(alpha<C)));
fprintf('Ant sv == C : %i\n',sum(alpha==C));
```

```
fprintf('Ant sv >0 : %i\n',sum(alpha>0));
b=-(max((alpha.*y)'*K(:,find(y==-1)))+min((alpha.*y)'*K(:,find(y==1))))/2;
```

A.1.4 Senter algoritmen

```
function [alpha,b]=senter3(K,y,C);
plotOn=0;
refinement = 0;
%figure;
if plotOn
  h=figure;
end
m=length(y);
mx=sum(y==1);
my=sum(y==-1);
delta=1/(mx*my);
alpha=((y==1)/mx+(y==-1)/my);
b=-0.5*sum(y'*((alpha*alpha').*K));
b_old=inf;
%f = y'*((alpha*alpha').*K) +b;
f=K*(y.*alpha)+b;

fprintf('delta: %g \n',delta)
%for i=1:10
i=100;
while((abs(b-b_old)>delta || i < 9)&&i<50)%min 9, max 100 iterasjoner
fprintf('Iterasjon: %i, |b-bold|= %g\n',i,abs(b-b_old));
  i=i+1;
  if plotOn
    %alpha er den inverse av avstanden til hyperplanet:
    % subplot(10,1,i,'align');
    plotResult(f,y,b,i);
    hold on;
    %plot
  end
 f(y==1)
 f(y==-1)
  alpha=1./(f.^2);

  % c1=sum(alpha_i x_i) \in konv(x_i: i \in I) konvekskombinasjon
  alpha(y==1) = alpha(y==1)./sum(alpha(y==1));

  alpha(y==-1) = alpha(y==-1)./sum(alpha(y==-1));
```

```
  %  alpha(alpha < 1/mx)=0;
  %  alpha(y==1) = alpha(y==1)./sum(alpha(y==1));
  %  alpha(y==-1) = alpha(y==-1)./sum(alpha(y==-1));
  b_old=b;
  b=-0.5*sum(y'*((alpha*alpha').*K));
  %f = y'*((alpha*alpha').*K) +b;
  f = K*(y.*alpha) +b;

  %alpha er skalert til sum alpha =1;
  %Det betyr at cI = sum alpha_i x_i, i \in I er en
  %konvekskombinasjon av x_i, i\in I
  %cJ=sum alpha_j x_j, j \in J er en konvekskombinasjon av x_j, j \in J
end
fprintf('Iterasjoner: %i\n',i);

% [m,n]=size(data);
%mx=sum(y==1);

%barx=1/mx*(data(labels==1))
%my=sum(y==-1);

%bary=1/my*(data(labels==-1))
%startløsning:
%w= barx-bary

%alpha=((y==1)/mx+(y==-1)/my);

%b=-0.5*sum(y'*((alpha*alpha').*K));

%skal være samme som:
%b=0.5*(-1/(my^2)*sum(sum(K(y==-1,y==-1))) +1/(mx^2)*sum(sum(K(y==1,y==1))));
%fprintf('b: %g',b);
%b=0;
%f = y'*((alpha*alpha').*K) +b;
%b=-0.5*(max(f(y==-1)) + min(f(y==1)))
%[y f']
%err=sum (y.*f'<0);
%fprintf('Feilklasssifisert: %i\n',err);
%delay
%for i=1:10000
%  for j=1:10000
%  i*j;
%  end;
%end;
```

```
%if plotOn
%  h=figure;
%  plotResult(f,y,b,i);
%end
if refinement
  C=0.9;%konvergensraten

  z=(y.*f <= max(y.*f));
  for i=1:10
    %punktene innenfor en viss avstand til hyperplanet
    %brukes til å beregne nytt senter for fordelingene.

    max(y.*f)
    z=(y(z).*f(z))<C*max(y(z).*f(z));
    sum(z)
    alpha(~z)=0;

    mx=sum(y(z)==1);
    my=sum(y(z)==-1);
    if (mx==0 || my==0)
      break;
    end;
    alpha(z)=(y(z)==1)/mx+(y(z)==-1)/my;
    b=0.5*(1/(my^2)*sum(sum(K(y(z)==-1,y(z)==-1))) -1/(mx^2)*sum(sum(K(y(z)==1,
    f = (y'*((alpha*alpha').*K))' + b;
  size(y)
  size(f)
    err(i)=sum (y.*f<0);
    fprintf('Feilklasssifisert: %i\n',err(i));
    if plotOn
      plotResult(f,y,b,1);
    end
  end
  plot(err);
end



function plotResult(f,y,b,i)

  %hold off
```

```
  plot(f(y==1),i*ones(sum(y==1),1),'ro',f(y==-1),i*ones(sum(y==-1),1),'bd');
  hold on;
  plot(b,i,'yx');
  %hold off;
```

A.1.5 Proximal SVM algoritmen

```
function [alpha,b] = psvmK(K,y,nu)
% Proximal Support Vector Machine Classifier
% Glenn Fung and Olvi L. Mangasarian
% linear and nonlinear classification
% INPUT: K, Y, nu.
% OUTPUT: alpha, b
% [alpha,b] = psvmK(K,Y,nu);
%
% Løser problemet:
% min nu/2*e'e+1/2*(alpha'alpha + b^2)
% s.t. Y(K(A,A')Y*alpha +b*e)+d=e
%
% 'Y' er en diagonalmatrise med +/-1 på diagonalen
% ettersom A(i,:) tilhører klasse +/-1.
% 'd' er en vektor med feilledd som minimeres.
% 'e' er en vektor av enere.
%
% bestemmelsesfunksjonen blir da:
% f(x)=K(x,A')Y*alpha+b;

[m,n]=size(K);
D=spdiags(y,0,m);
e=ones(m,1);

G=D*[K -e];
%v=inv(I/nu + G*G')*e;
%[j,k]=size(G*G')

I=speye(m);
v=(I/nu + G*G')\e;
u=D*K'*D*v;
gamma = -e'*D*v;
alpha =u;
b= -gamma

%r=sum(H)'; %r=H'*e;
%r=(speye(n+1)/nu+H'*H)\r; %solve (I/nu+H'*H) r=H'*e
```

```
%alpha=nu*(1-(H*r));
%s=D*alpha;

%w=(s'*A)'; %w=A'*D*u

%gamma = -sum(s); %gamma=-e'*D*u
%b=-gamma
b=-(max((alpha.*y)'*K(:,1:4))+min((alpha.*y)'*K(:,5:8)))/2
```

A.2 Kjernefunksjoner

A.2.1 Gaussisk kjerne

```
function K=gaussKernel(A,B,sigma,dummy)
% function K=gaussKernel(A,B,sigma)
% A er m*n, B er n*k matriser
% Den gaussiske kjernen er gitt ved:
% K_ij=e^-||A_i'-B_j||^2/(2*sigma^2), i=1..m, j=1..k
[m,n]=size(A);
[l,k]=size(B);

if n~=k
  error('A og B må ha like mange kolonner');
  return;
end
%B=B'
%preallokerer minne for k slik at den doble for-loopen kan gå raskere
%K1=zeros(m,k);
%cp=cputime;
%for i=1:m
%  for j=1:k
%    ab=A(i,:)'-B(:,j);
%    K1(i,j)=ab'*ab;
%  end
%end
%K1=exp(-K1/(2*sigma^2));

%cputime-cp
%cp=cputime;
%||x-y||^2=
%<x-y,x-y>=<x,x>-2<x,y>+<y,y>
%<A'-B,A'-B>=<A',A'> -2<A',B>+<B,B>=
%size(A)
%size(B)
```

```
A2=(sum((A.^2)'))';
B2=sum((B.^2)')';
AB=A*B';
%size(repmat(A2,length(B2),1))
%size(repmat(B2,1,length(A2)))
%size(AB)
K=repmat(A2,1,length(B2)) -2*AB + repmat(B2',length(A2),1);
K=exp(-K/(2*sigma^2));

%cputime-cp
%K2= K-K1;
%K2(1:10,1:10)
%sum(sum(K-K1))

%doble for-løkke brukte 2 min 8 sekunder på rosis
%Matrise versjonen brukte 4 sekunder!!
```

A.2.2 Sigmoid kjerne

```
function K=sigmoidKernel3(A,B,sigma)
% function K=sigmoidKernel(A,B,k,q)
% K(xi,xj)=tanh(k*(xi'*xj)+q);
% eller K= tanh(||A-B||^2/(2*sigma^2))


[m,n]=size(A);
[o,p]=size(B);
if(n~=p)
 error('A og B må ha samme antall søyler.');
 return;
end

A2=sum(A.^2,2);
A2=repmat(A2,1,o);
B2=sum(B.^2,2);
B2=repmat(B2,1,m)';
AB=A*B';

K=A2 -2*AB + B2;

%K=exp(-K/(2*sigma^2));

%K=tanh(k*(A*B')+q*ones(m,o));
%K=-tanh(k*K+q*ones(m,o));
```

```
K=tanh(K/(2*sigma^2));%+q*ones(m,o));
```

A.2.3 Polynomisk kjerne

A.2.4 Spektral Angle kjerne

```
function K=SAKernel(A,B,r)
%radial basis function with Spectral Angle measure
%K(x,y)=arccos(-x'y/(||x||*||y||+r))
[m,n]=size(A);
[k,l]=size(B);
AB=A*B';
A2=sqrt(sum(A.^2,2));
A2=repmat(A2,1,k);
B2=sqrt(sum(B.^2,2));
B2=repmat(B2,1,m)';
K=AB./(A2.*B2+r*ones(m,k));
K=acos(-K);
```

A.3 Preprosessering

A.3.1 Normalisering

```
function data_norm=normalisering2(data)
% Normaliserer dataene til middelverdi 0
% og enhets varians.
%sigma=sqrt(sum(x-x_mean)^2/(N-1))
%x_norm=(x-x_mean)/sigma

[N,k]=size(data);
%N er antall objekter/pixler.
%k er antall features.

%feature normalisation
xmean=sum(data)/N;
%tic
%for i = 1:k
%  sigma2(i)=1/(N-1)*sum((data(:,i)-xmean(i)).^2);
%end
%for i = 1:k
%  data_norm(:,i)=(data(:,i)-x_mean(i))./sqrt(sigma2(i));
%end
%toc
%figure(1);
```

```
%imagesc(data_norm(1:100,:));

%tic
%vektorisering ca 7 ganger raskere

xmean2=(ones(N,1)*xmean);%segmentation fault???
data_xmean=data-xmean2;
sqr_sigma2=sqrt(sum((data_xmean).^2)/(N-1));
sqr_sigma2_2=(sqr_sigma2'*ones(N,1)')';
data_norm=data_xmean./sqr_sigma2_2;

%data_norm=data_xmean./(sqrt(sum((data_xmean).^2)/(N-1))'*ones(N,1)')';
%toc
%figure(2);
%imagesc(data_norm(1:100,:));
%sum(sum(data_norm-data_norm2))
```

A.3.2 Egenskapsutvelgelse med Fisher Lineare Diskriminant

```
function [C,d] = fdr(data,labels,k)
%function C = fdr(data,labels)
%fdr (fishers discriminant ratio) is a feature extraction function.
% I.e. fdr gives a transformation of data => datafdr
% which maximizes the between class separability (Sb) and minimizes the
% within class scatter (Sw).
% the number of features is redused from n to l<M, wehere M is the
% number of classes.
% It does this by finding k eigenvectors of inv(Sw)*Sb
% corresponding to the k argest eigenvalues,
% where k is the number of classes.

[m,n]=size(data);
%within class scatter Sw:
Sw=zeros(n,n);
%Between class scatter Sb:
Sb=zeros(n,n);

globalmean=mean(data);
for i=1:k
  Pi=sum(labels==i)/m;
  Si=cov(data(labels==i,:));
  Sw=Sw+Pi*Si;

  meani=mean(data(labels==i,:));
```

```
  mg=meani-globalmean;
  Sb=Sb+Pi*(mg'*mg);
end
S=inv(Sw)*Sb;
[C,D]=eig(S);
d=diag(D);

%[ds,di]=sort(d,'descend');
%...d er sortert :-)

C=C(:,1:k-1);
%datafdr=data*C;
plotd=0;
if plotd
  figure;
  subplot(2,1,1);
  plot(abs(d));
  Title('Plot av egenverdiene til Fisher projeksjonen av et rosis joined sett')
  ylabel('Delplot 1')
  subplot(2,1,2);
  plot(abs(d(1:4)));
  xlabel('Plot av de fire første egneverdiene.');
  ylabel('Delplot 2');
end;
```

A.3.3 Egenskapsutvelgelse med Prinsipal Komponent Analyse

```
function [A,d]=pca(data,l)
%function A=pca(data)
%pca = Principal Component Analysis
% A is the projection matrix y= A^T x that diagonalizes the
% correlation matrix R_y=E[yy^T]=E[A^T xx^T A]=E[A^T R_x A]
% The collumns of A consist of the eigenvectors of R_x.
% By choosing the l<n eigenvectors corresponding to the l largest
% eigenvalues we can reduse the input space from n to l while
% keeping the most information.
[m,n]=size(data);
%l=n;
%Rx=data'*data;
Rx=corrcoef(data);
[A,D]=eig(Rx);
d=diag(D);
%alle egenverdiene er 1
```

```
%A'*Rx*A=I
%A'*A=I
[ds,di]=sort(d,'descend');

A=A(:,di(1:l));

plotd=0;
if plotd
  figure;
  subplot(2,1,1);
  plot(abs(d(di)));
  Title('Plot av egenverdiene til PCA projeksjonen av et rosis joined sett');
  ylabel('Delplot 1')
  subplot(2,1,2);
  plot(abs(d(di(1:10))));
  xlabel(sprintf('Plot av de %i første egenverdiene.',l));
  ylabel('Delplot 2');
end;
```

A.4 Kjøringsalgoritmer

A.4.1 Datautvelgelse og preprosessering for Fontainebleau datasettet kalt rosis joined i matlabfilene

```
clear;
%******INITSIALISERING*****************
prosent=[];
snittklasseprosent=[];
klassifikator = cell(5,1);
klassifikator= {'on_line','psvmK','pathQuad2','senter2','quad_prog','senter3'};
antklassifikatorer=length(klassifikator);

kernels= {'linearKernel','polynomKernel','gaussKernel','sigmoidKernel', ...
        'SAKernel','gaussSAKernel','sigmoidGaussKernel','sigmoidGauss2Kernel'};
klassifikatornr=1
antklasser=3;
kernelnr=3;

PCA=0;
l=81; %antall egenvektorer som brukes i PCA projeksjonen
FDR=0;
sigma=[6];
kernelparam2=1;
%sigma=100;
```

```
diary on;
for n=[90]% 300 600]%[90 300 600]; %90 300 eller 600 punkter pr klasse
%Cv=[40 50];%[1 5 10 20 30 40 50 60 70 80 100] ;
Cv=[3 4 5 6 7];
for t=1:length(Cv)
  %l=Cv(t);
  C=Cv(t);
 %C=50;
  %testplott=0;           % =1 plotter prosent rett for alle sigma
  % verdier, =0 gir ingen plot
  test=0;               %=1 klassifiserer rosis_n_test, =0 rosis_n_val_%1_%2
  utskrift =1;
  r=10;% antall realiseringer 1-r, max r=10

p2=zeros(r,1);%prosent rett klassifisert
snittklasseprosent=zeros(r,1);%snitt av prosentrett for hver klasse
klasseprosent=zeros(r,antklasser);%prosent riktig for hver klasse
klasseprosent_sigma=zeros(length(sigma),antklasser);
snittprosent_sigma=zeros(length(sigma),1);
snittklasseprosent_sigma=zeros(length(sigma),1);

%diary senter2_rosis90_sigma1til3_fdr

%*********UTVELGELSE AV TRENINGS OG TEST DATA*************
for i= 1 : length(sigma)
  kernelp=[kernelnr sigma(i) kernelparam2];
  %n er antall treningspunkter pr klasse, m er realiseringsnr.
  %for n=[90 300 600];
  j=1;
  for m=1:r
    %*********treningsdata***************
    s1=['rosis_train_n_' int2str(n) '_' int2str(m)];
    load('./../data/rosis_joined_struct.mat',s1);
    trendata = getfield(eval(s1),'data');
    trendata=trendata(:,3:83);
    trenlabels=getfield(eval(s1),'nlab');
    clear(s1);
    %**********valideringssett****************
    if ~test
      s2=['rosis_val_n_' int2str(n) '_' int2str(m)];
      load('./../data/rosis_joined_struct.mat',s2);
      testdata=getfield(eval(s2),'data');
      testdata=testdata(:,3:83);
      testlabels=getfield(eval(s2),'nlab');
```

```
  clear(s2);
end
%**********testsett********************
if test
  %trendata=[trendata;testdata];
  %trenlabels=[trenlabels;testlabels];
  load('./../data/rosis_joined_struct.mat','rosis_n_test');
  testdata=rosis_n_test.data(:,3:83);
  testlabels=rosis_n_test.nlab;
  clear('rosis_n_test');
end

%**********supertren**************
%load('./../data/rosis_joined_struct.mat','rosis_train');
%trendata = rosis_train.data(:,3:83);
%trenlabels=rosis_tren.nlab;
%clear('rosis_train');

%load('./../data/rosis_joined_struct.mat','rosis_test');
%testdata = rosis_test.data(:,3:83);
%testlabels=rosis_n_test.nlab;
%clear('rosis_test');

%******normalisering**************
trendata=normalisering2(trendata);
testdata=normalisering2(testdata);

%********Fisher projection****************
if FDR==1
  F= fdr(trendata,trenlabels,antklasser);%evt l gir l-1
  trendata=trendata*F;
  testdata=testdata*F;
  datasett=sprintf('Fontainebleau_n_%i_%i_fdr_test%i',n,m,test);
elseif PCA==1
  F=pca(trendata,l);%l angir antall egenvektorer som brukes i
                    %projeksjonen
  trendata=trendata*F;
  testdata=testdata*F;
  datasett=sprintf('Fontainebleau_n_%i_%i_pca_test%i',n,m,test);
else
  datasett=sprintf('Fontainebleau_n_%i_%i_test%i',n,m,test);
end
% diary off;
size(trendata)
```

```
  [p2(m),snittklasseprosent(m),klasseprosent(m,1:antklasser)] = ...
      klassifiser7(trendata,trenlabels,testdata,testlabels, ...
                   antklasser,datasett,klassifikatornr,kernelp,C);
%diary on;
end
if length(sigma)>1
  snittprosent_sigma(i)=sum(p2,1)/r;
  snittklasseprosent_sigma(i)=sum(snittklasseprosent,1)/r;
  klasseprosent_sigma(i,:)=sum(klasseprosent,1)/r;
end
if utskrift==1
  fprintf('Snitt prosent for %i testkjøringer: %3.4f%%\n',r, ...
          snittprosent_sigma(i));
  fprintf('Snitt klasseprosent for %i testkjøringer: %3.4f%%\n', ...
          r, snittklasseprosent_sigma(i));
  s=sprintf(['/ifi/fenris/h39/bjorntau/Hovedfag/SVM/LOG2/' ...
             'testkjoring_%s_%s_sigma%g.log'],datasett,klassifikator{ ...
                 klassifikatornr},sigma(i));
  fid = fopen(s,'A');
  fprintf(fid,'sigma: %1.1f, n=%i \n',sigma(i),n);
  fprintf(fid,'Snitt prosent for %i testkjøringer: %3.4f%%\n', ...
          m,snittprosent_sigma(i));
  fprintf(fid,'Snitt klasseprosent for %i testkjøringer: %3.4f%%\n', ...
          m,snittklasseprosent_sigma(i));
  fprintf(fid,'prosent for hver kjoring:\n');
  fprintf(fid,'%g\n',p2);
  fprintf(fid,'klasseprosent:\n');
  fprintf(fid,'%g\t%g\t%g\n',klasseprosent_sigma(i));
  fclose(fid);
  prosent=[prosent sigma(i) snittprosent_sigma(i) snittklasseprosent_sigma(i)
  [z,u]=size(klasseprosent);
  %snittklasseprosent=[snittklasseprosent; sum(klasseprosent,1)/z];
end
fprintf('Prosent rett: %g %% \n',sum(p2)/length(p2));
%Standard avvik prosent rett:
fprintf('Standardavvik for prosentrett: %g', ...
        std(p2));

snittklasseprosent
%Standard avvik snitt klasseprosentene:
fprintf('Gj.snitt snittklasseprosent: %g \n',sum(snittklasseprosent)/length(s
fprintf('Standardavvik for Snitt klasseprosent: %g', ...
        std(snittklasseprosent));
klasseprosent
```

```
end
if length(Cv)>1
  snittklasseprosent_C(t)=sum(snittklasseprosent,1)/r;
  snittprosent_C(t)=sum(p2,1)/r;
  klasseprosent_C(t,:)=sum(klasseprosent,1)/r;

end
if (length(sigma)>1)
    %klasseplott(l,'Antall PCA egenskaper', klasseprosent, p2, snittklasseprosent,
    %antklasser, klassifikator{klassifikatornr},n,sigma);
    sigmaplott(klasseprosent_sigma,snittprosent_sigma, ...
               snittklasseprosent_sigma,antklasser,klassifikator{klassifikatornr},sigma,
end
filename=sprintf('/ifi/fenris/h39/bjorntau/Hovedfag/SVM/LOG3/sigmaplott_%s_%s_val_sigma%
               min(sigma),max(sigma),date);
tittel=sprintf('Klassifikasjon med %s av rosis joined %i med gaussisk kernel',klassifika
save(filename,...
     'klasseprosent_sigma', ...
     'snittprosent_sigma', ...
     'snittklasseprosent_sigma', ...
     'antklasser', ...
     'klassifikatornr',...
     'sigma', ...
     'n',...
     'C',...
      'tittel', ...
     'klassifikator');



s2=sprintf('/ifi/fenris/h39/bjorntau/Hovedfag/SVM/LOG3/%s_%s_sigma%g-%g%s.mat',klassifik
          min(sigma),max(sigma),date);
save(s2,'p2','snittklasseprosent','klasseprosent','klassifikator','klassifikatornr','n',
     'kernelnr','C');%saves variables to filename given by s

end
%diary off;
diary([s2 '.diary']);
end
if length(Cv)>1
  s3=sprintf('/ifi/fenris/h39/bjorntau/Hovedfag/SVM/LOG3/Cplott_%s_%s_C%g-%g_sigma%g_%s
          min(Cv),max(Cv),sigma(1),date);
```

```
  save(s3, ...
       'Cv', ...
       'snittklasseprosent_C', ...
       'klasseprosent_C', ...
       'snittprosent_C', ...
       'antklasser', ...
       'klassifikatornr',...
       'sigma', ...
       'n',...
       'Cv',...
       'tittel', ...
       'klassifikator');
   Cplott(klasseprosent_C,snittprosent_C, ...
               snittklasseprosent_C,antklasser,klassifikator{klassifikatornr},C

  %figure;
  %plot(Cv,snittklasseprosent_C,Cv);
  %title(sprintf('Gjennomsnittlig klassifikkasjon av Fontainebleau test og \n v
  %xlabel('C');
  %ylabel('Gjennomsnitt prosent rett av klassene');
end;
```

A.4.2 Datasettutvelgesle og preprosessering for andre datasett

```
%function kjoring(valg)
valg=0;

datasett = cell(7,1);
datasett= {'rosis', 'diabetes', '5klasse_2dim','chekkerboard16_2dim' ...
           '9klasse_2dim','2klasse_2dim','rosis_joined_90_1'};
antdatasett=length(datasett);
while ~(sum(valg==1:length(datasett))==1)
  disp('Velg datasett:');
  for i=1:antdatasett
    fprintf('%i: %s\n',i,datasett{i});
  end
  valg=input('Velg nr.: ');
end
load(sprintf('./../data/%s.mat',datasett{valg}));


[m1,n]=size(trendata)
m2=length(testdata);
```

```
fprintf(['Det er %i treningseksempler og %i testeksempler med %i ' ...
         'egenskaper og %i klasser.\n'],m1,m2,n,antklasser);

klassifikator= {'on_line','psvmK','pathQuad2','senter2','quadProg3','senter3'};
klassifikatornr=2;
kernels = cell(8,1);
kernels= {'linearKernel','polynomKernel','gaussKernel','sigmoidKernel', ...
        'SAKernel','gaussSAKernel','sigmoidGaussKernel','sigmoidGauss2Kernel'};
kernelnr=3;
%kv=0.1:0.1:1
kv=0.5
Cv=[5];
for i=1:length(kv)
  k=kv(i)

  for j=1:length(Cv)
    %q=qv(j)
    C=Cv(j)


    norm2=1;
    if norm2==1
      %******normalisering**************
      trendata=normalisering2(trendata);
      testdata=normalisering2(testdata);
    end
    FLD=0;
    if FLD
      %F=pca(trendata);
      F= fdr(trendata,trenlabels,antklasser);
      trendata=trendata*F;
      testdata=testdata*F;
    end
    [prosentrett(i,j),snittklasseprosent(i,j),klasseprosent]= ...
        klassifiser7(trendata,trenlabels,testdata,testlabels, ...
                     antklasser,datasett{valg},klassifikatornr, ...
                     [kernelnr,k,q],C);
  end
end
%[p,p2,resultat] = klassifiser7(trendata,trenlabels,testdata, ...
%                               testlabels,antklasser,datasett{valg});
% for sigma = 1:10
%    [p,p2(m),pkl(m),klasseprosent(m,:),f5] = klassifiser6(trendata,trenlabels,testdata
```

```
%skrivLogg(resultat,datasett,kernel,klassifikator);
```

A.4.3 Kjerneutregning, Klassifikasjon og utskrifter

```
function [p2,snittklasseprosent,klasseprosent] = klassifiser7( trendata, ...
   trenlabels, testdata, testlabels, kl, datasett, valg, kernel, C);
% function [p2,pkl,klasseprosent] = klassifiser7( trendata, ...
%  trenlabels, testdata, testlabels, kl, datasett, valg, kernel, C);
% trener k*(k-1)/2 klassifikatorer.
% En klassifikator for hvert  sett av 2 klasser.
% 'trendata' er en matrise med rader av treningspunkter
% 'trenlabels' angir klassetilhørigheten til punktene
% tilsvarende for 'testdata' og 'testlabels'
% 'kl' er antall klasser
% 'valg' angir nr på klassifikator
% 'kernel' inneholder kernelinformasjon
% 'C' > 0 angir SVM hardheten inf er hard margin, ellers myk
% margin.
% 'p2' er prosent rett klassifisert
% 'pkl' prosent rett for hver klasse
TESTUTSKRIFT = 0;
tid=0;
tic;
k=cell(kl,1);
[m,n]=size(testdata)
%hvis det er to dimmensjoner plottes løsningsrommet i 2-d
if n==2
  testdata2=plotdata(trendata);
end
if nargin < 8
  kernel=0;
end
if nargin < 7
  valg=0;
end

%****************************************
%****Velger klassifikator algoritme*********
%****************************************
klassifikator = cell(5,1);
klassifikator= {'on_line','psvmK','pathQuad2','senter2', ...
                'quadProg3', ... %'kjoring_quadProg', ...
```

```
                'senter3'};
antklassifikatorer=length(klassifikator);
while ~(sum(valg==1:antklassifikatorer)==1)%så lenge klassifikator
                                           %ikke er valgt
  disp('Velg klassifikator:');
  for i=1:antklassifikatorer
    fprintf('%i: %s\n',i,klassifikator{i});
  end
  valg=input('Velg nr.: ');
  if (valg ~=1 && valg ~=2 && valg ~=3 && valg ~=4 && valg ~=5)
    disp('Du må taste 1, 2, 3, 4 eller 5!');
  end
end
if nargin ==9
  klassifikatorparameter=C;
else
  if (valg==1)
    C=input('Hva er C (3.5) ? (inf gir hard margin): ');
    %C=3.5;
    klassifikatorparameter=C;%for on-line på rosis_joined
  elseif(valg==2)
     C=input('Hva er klassifikator parameter C (0.5): ');
    klassifikatorparameter=C;
  elseif(valg==3)
    C=input('Hva er klassifikatorparameter C for pathQuad ?: ');
    klassifikatorparameter=C
  elseif (valg==4)
    C=input('Hva er klassifikator parameter C for senter?: ');
    klassifikatorparameter=C;
  elseif (valg==5)
    C=input(['Hva er klassifikatorparameter C for quadprog(23)?(inf ' ...
             'gir hard margin): ']);
    klassifikatorparameter=C;%quadprog C=23
  end
end;

%***********************************
%**** Meny for valg av Kernel ******
%***********************************
%kernels = cell(5,1);
%kernels= {'polynomKernel','gaussKernel','sigmoidKernel2','SAKernel', ...
%          'PIDKernel'};
kernels = cell(8,1);
kernels= {'linearKernel','polynomKernel','gaussKernel','sigmoidKernel','SAKernel','gauss
```

```
antkernels=length(kernels);
while ~(sum(kernel(1)==1:antkernels)==1)
  disp('Velg kernel:');
  for i=1:antkernels
    fprintf('%i: %s\n',i,kernels{i});
  end
  kernel(1)=input('Velg nr.: ');
  if (kernel ~=1 && kernel(1) ~=2 && kernel(1) ~=3 && kernel(1) ~=4 ...
      && kernel(1) ~=5)
    disp('Du må taste 1, 2, 3, 4 eller 5!');
  elseif (kernel(1)==1)
    d=0;
    while ~((d<=20) && (d>0))
      d=input('Angi graden til polynomiell kernel (1-20): ');
      kernel(2)=d;
    end
  elseif(kernel(1)==2)
    sigma=input(['Hva er sigma? K(x,y)=exp(-||x-y||^2/(2sigma^2)) ' ...
                 '(sigma>0): ']);
    kernel(2)=sigma;
  elseif(kernel(1)==3)
    %sigmoid kernel
    k=input('Hva er k: ');
    kernel(2)=k;
    %k=kernel(2);
    q=input('Hva er q: ');
    kernel(3)=q;
    %q=1;
  else
    kernel(2)=input('Kernelparameterverdi: ');
  end
  if length(kernel)<3
    kernel(3)=0;
  end
end
if TESTUTSKRIFT
  fprintf('Tid brukt: %f\n',tid);
end;
time = cputime;

%*********************************************
%**** Klassifiserer klasse i mot klasse j ****
%**** Iter = kl*(kl-1)/2  ****
%*********************************************
```

```
f=zeros(kl-1,kl,length(testlabels));
if n==2
  fplot=zeros(kl-1,kl,length(testdata2));
end
diary off;
for i=1:kl
  for j=i+1:kl
    fprintf('Klassifikasjon klasse %i mot klasse %i: %i av %i\n', ...
            i, j,kl*(kl-1)/2-(kl-i)*(kl-i+1)/2+j-i,kl*(kl-1)/2);
    %      nu=0.1;
    pack;    fprintf('Beregner treningskernel K...');
    A=[trendata(trenlabels==i,:); trendata(trenlabels==j,:)];
    if kernel(1)==4
      K=feval(kernels{kernel(1)},A,A,kernel(2),kernel(3));
    else
      K=feval(kernels{kernel(1)},A,A,kernel(2));
    end
    fprintf('ferdig!\n');
    %positiv definit test
    disp('sum(sum(K-(K+K^T)/2)):')
    sum(sum(K-((K+K')/2)))

    if det(K)>=0
      disp('Kernelen er positiv semidefinit.');
    else
      disp('Kernelen er ikke positiv semidefinit!');
    end

    y=[ones(sum(trenlabels==i),1); -ones(sum(trenlabels==j),1)];
    %klasse i får verdi 1, j får -1
    %D=10;
    %K=K+D*speye(size(K));
    lagR2=0;
     if lagR2
        disp('mean_a');
        %cpu=cputime;
        Mean_A=ones(length([trendata(trenlabels==i,:); trendata(trenlabels==j,:)]),1)*me
        %cputime-cpu;
        fprintf('S....');
        S=[trendata(trenlabels==i,:); trendata(trenlabels==j,:)]-Mean_A;%sentrerer punkt
        fprintf('done\n');

        fprintf('R...');
        R2=max(sum(S'.*S')); %2-norm^2
```

```
    fprintf('done\n');
    cpu=cputime;
    K= K + ones(size(K))*R2;

   % if n==2
   %   X2=X2+ones(size(X))*R2;
   % end;
    %cputime-cpu;
  end;
[alpha,b] = feval(klassifikator{valg},K,y,klassifikatorparameter);
clear K;
%[alpha,j]=svo(K,y,10);%alpha er supportvektorer, j er indeksen
%b=-0.5*(
%max((alpha.*y(j))'*K(:,find(y==-1)))+min((alpha.*y)'*K(:,find(y==1))));
%Support-vector statistikk:
if valg~=4
  sv=(alpha<(C-1000*eps) & (alpha>(0+1000*eps)));
  fprintf('max(alpha): %4.16f\n',max(alpha));
  fprintf('min(alpha): %4.16f\n',min(alpha));
 % fprintf('min(abs((alpha)): %4.16f\n',min(abs(alpha)));
 % fprintf('C: %g\n',C);
  fprintf('Antall SV: %i\n',sum(sv));
 % fprintf('Optimal verdi: %g, b: %g\n', (alpha'.*y')*K*
 % (alpha.*y)- sum(alpha),b);
end
%pack;
%Lagrer memory til disk og leser fra disk
%for å ungå fragmentering i memory ved store datamengder.
fprintf('Beregner testkernel X...');
if kernel(1)==4
  X=feval(kernels{kernel(1)},testdata,A,kernel(2),kernel(3));
else
  X=feval(kernels{kernel(1)},testdata,A,kernel(2));
end
fprintf('ferdig!\n');

if TESTUTSKRIFT
  disp('test2 kernelmatrise ferdig');
  fprintf('trener opp  klassifikator for klasse %i og %i\n',i,j);
end;
%Klassifiserer testpunkter
if lagR2
  X= X + ones(size(X))*R2;
end
```

```
    f(i,j,:)=X*(y.*alpha)+b;
    clear X;
    if n==2
      fprintf('Beregner testkernel2 X...');
      if kernel(1)==4
        X2=feval(kernels{kernel(1)},testdata2,A,kernel(2),kernel(3));
      else
        X2=feval(kernels{kernel(1)},testdata2,A,kernel(2));
      end
      %Klassifiserer punkter for plotting
      if lagR2
         X2=X2+ones(size(X))*R2;
      end
      fplot(i,j,:)=X2*(y.*alpha)+b;
      clear X2;
    end
  end
end
diary on;
time = cputime-time;

% *********************************
% m-klasser er parvis klassifisert
% hvert punkt gis den klassen den er flest ganger klassifisert til
% Dvs. velger den mest sannsynlige klassen
[m,n]=size(testdata);
f3=zeros(m,kl);
f5=zeros(m,1);


for k= 1:m
  f(:,:,k)=(f(:,:,k)>0).*([1:kl-1]'*ones(1,kl)) + (f(:,:,k)<0).* ...
           (ones(kl-1,1)*[1:kl]);
  % summerer opp antall ganger hver klasse er valgt
  for i = 1:kl
    f3(k,i)=sum(sum(f(:,:,k)==i));
  end
  [x,f5(k,1)]=max(f3(k,:));
end

if n==2
  [m2,n2]=size(testdata2);
  f32=zeros(m2,kl);
  f52=zeros(m2,1);
```

```
  for k= 1:m2
    fplot(:,:,k)=(fplot(:,:,k)>0).*([1:kl-1]'*ones(1,kl)) + ...
        (fplot(:,:,k)<0).* (ones(kl-1,1)*[1:kl]);
    % summerer opp antall ganger hver klasse er valgt
    for i = 1:kl
      f32(k,i)=sum(sum(fplot(:,:,k)==i));
    end
    [x2,f52(k,1)]=max(f32(k,:));
  end
  disp('testplot');
  [snittklasseprosent, klasseprosent] = visProsentrapport(0,f5,testlabels,kl);
  testplot(testdata2,f52,trendata,trenlabels,kl,kernel,kernels,C,snittklassepro
end

p=sum((f5-testlabels)==0);   % ant rett klassifisert

p2=p/length(testlabels)*100; % %riktig klassifisert

fprintf('Riktig klassifisert: %f %%\n',p2);

%***************Skriv Logg*********************
antTren=length(trenlabels);
antTest=length(testlabels);
M=forvirring(f5,testlabels,kl);
fid = fopen(sprintf('/ifi/fenris/h39/bjorntau/Hovedfag/SVM/LOG2/%s.log', ...
                    datasett),'A');
skrivLogg(fid,datasett,klassifikator{valg},kernel,antTren,antTest,C);
vistid(fid,time);
visForvirringsmatrise(fid,M,kl);
[snittklasseprosent, klasseprosent] = visProsentrapport(fid,f5,testlabels,kl);
fclose(fid);
skrivLogg(1,datasett,klassifikator{valg},kernel,antTren, antTest,C);
vistid(1,time);
visForvirringsmatrise(1,M,kl);
visProsentrapport(1,f5,testlabels,kl);

%***********Forvirringsmatrise******************
function M=forvirring(resultat,testlabels,antklasser)
M=zeros(antklasser,antklasser);
for i=1:antklasser
  for j=1:antklasser
    M(i,j)=sum(resultat(testlabels==i)==j);
    %summere opp alle som var av klasse i og som ble klassifisert
    %til klasse j
```

```
  end;
end;

%********skrivLogg*********************************
function skrivLogg(fid,datasett,klassifikator, kernel, antTren, ...
                   antTest,C)
fprintf(fid,'\n%s\n',datestr(now));
fprintf(fid,'Klassifikasjon med:  %s, C= %g\n',klassifikator,C);
fprintf(fid,'Klassifikasjon av:  %s\n',datasett);
%fprintf(
kernels= {'linearKernel','polynomKernel','gaussKernel','sigmoidKernel','SAKernel','gauss
if kernel(1) ==1
  fprintf(fid,'linearKernel: K(x,y)=<x,y>\n');
elseif  kernel(1) ==2
   fprintf(fid,['PolynomKernel:  K(x,y)=(<x,y>+1)^d,d=%g\n'],kernel(2));
elseif kernel(1) ==3
  fprintf(fid,['Gaussisk kjerne: K(x,y)=exp(-||x-y||^2/(2*sigma^2), ' ...
               'sigma=%g\n'],kernel(2));
elseif kernel(1) ==4
  fprintf(fid,['Sigmoid kjerne: K(x,y)=tanh(k<x,y>+q) ' ...
               'k=%g, q=%g\n'],kernel(2),kernel(3));
elseif kernel(1) ==5
  fprintf(fid,['Spectral Angle kjerne: K(x,y)=exp(-arccos(<x,y>/||x||*' ...
               '||y||)/(2*sigma^2), sigma=%g\n'],kernel(2));

else
  fprintf(fid,'Kjerne: %s %g\n',kernels(kernel(1)),kernel );
  for i=1:length(kernel)-1
    fprintf(fid,'Parameter%i: %g \n',i,kernel(i+1));
  end;
end
fprintf(fid,'Datasett: %s\n',datasett);
fprintf(fid,'Antall treningspunkter:\t %i\n',antTren);
fprintf(fid,'Antall testpunkter:\t %i\n',antTest);

%************************************************
function visForvirringsmatrise(fid,forvirringsmatrise,kl)
% visForvirringsmatrise(forvirringsmatrise)
  fprintf(fid,'***********Forvirringsmatrise**********\n');
  for i=1:kl
    for j=1:kl
      fprintf(fid,'%i\t',forvirringsmatrise(i,j));
    end
    fprintf(fid,'\n');
```

```
  end

%************* visProsentrapport ************************************
 function [snittklasseprosent, klasseprosent] = ...
    visProsentrapport( fid, resultat, testlabels,kl)
% function [snittprosent, prosentriktig] = ...
%    visProsentrapport( fid, resultat, testlabels,kl)
% 'klasseprosent' er prosent riktig klassifisert for alle klassene
% 'snittklasseprosent' er snittet av klassenprosentene
fprintf(fid,'Klassenr:\t ant test: \t ant rett: \t prosent rett:\n');
antrett=zeros(kl,1);
klasseprosent=zeros(kl,1);
for i = 1:kl
  antrett(i)=sum(resultat(testlabels==i)==testlabels(testlabels==i));
  klasseprosent(i)=  antrett(i)/sum(testlabels==i)*100;
  fprintf(fid,'%i \t\t %1.0i \t\t %1.0i \t\t %3.4f  \n',i, ...
          sum(testlabels==i),antrett(i),klasseprosent(i));
end;
snittklasseprosent=sum(klasseprosent)/kl;
fprintf(fid,'Snitt av klasseprosent for klassene: %f\n', ...
        snittklasseprosent);
totaltriktig=sum(resultat==testlabels)/length(testlabels)*100;
fprintf(fid,'Prosentriktig totalt: %3.4f\n',totaltriktig);

%*******************************
function vistid(fid,tid)
minutter=floor(tid/60);
sek=floor(mod(tid,60));
fprintf(fid,'Tid brukt: %i min, %i sek\n',minutter,sek);

function testplot(data,lab1,data2,lab2,kl,kernel,kernels,C, ...
                  snittklasseprosent,klassifikator);
% function testplot(data,lab1,data2,lab2,kl)
%figure;
la=cell(kl,1);
for i=1:kl
  la(i)={find(lab1==i)};
end
%str=['b.'; 'g.'; 'r.'; 'c.'; 'm.'; 'y.'; 'k.';'b.'; 'g.'];
%hold on;
data_la=zeros(length(lab1),1);
for i =1:kl%  plot(data(la{i},1),data(la{i},2),str(i,:));
  data_la(la{i})=i;
end
```

```
kl
[lab1(1:100:400)';data_la(1:100:400)']'

for i=1:kl
  la(i)={find(lab2==i)};
end
str2=['bo'; 'g*'; 'r+'; 'cs'; 'md'; 'yv'; 'kp';'bo'; 'g*'];
%for i =1:kl
%  plot(data2(la{i},1),data2(la{i},2),str2(i,:));

%end
%title(sprintf('Testplot av %i klasser.',kl));
%xlabel('* angir treningspunkter');
%hold off;
size(data(:,1))
size(data(:,2))
size(data_la)



%pcolor(data2(:,1),data2(:,2),data2_la);

i=1:51;
j=1:51;
i2=repmat(i,1,length(j));
j2=[];
for k=1:length(j)
  j2=[j2 j(k)*ones(1,length(i))];
end
B=[i2; j2]';

c2=zeros(51,51);
for i=1:51*51

    c2(B(i,1),B(i,2))=data_la(i);
end;

figure;

%subplot(2,2,4);

hold off;
pcolor(c2');
shading('interp');
```

```
%colormap gray
hold on;

%transformerer dataene til intervallet 1-51
data2=data2-repmat(min(data2),length(data2),1);
data2=data2./repmat(max(data2),length(data2),1).*50+1;

str2=['c*'; 'y*'; 'm*'; 'c*'; 'm*'; 'y*'; 'k*';'b*'; 'g*'];
for i =1:kl
  plot(data2(la{i},1),data2(la{i},2),str2(i,:));
end
%title(sprintf('Testplot av %i klasser.',kl));
%title('Sigmoid kjerne, k=1, q=0.6, C=30');
s=[];
%s=sprintf('Klassifikator: %s \n',klassifikator);
s=[s sprintf('%s',kernels{kernel(1)})];
for i=1:length(kernel)-1
  s=[s sprintf(' Param%i: %g',i,kernel(i+1))];
end
s=[s sprintf(' C=%g',C)];
title(s);

%xlabel(fprintf('Riktig klasifisert: %g\% \n',snittklasseprosent));
%axis equal;
axis off;
text(10, -3,sprintf('Riktig klasifisert: %g %%',snittklasseprosent));
hold off;
%text(20, -3, '83%rett');
max(data2)
min(data2)
```